Teendők listája

☐ Erlang LS 7

Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozáselmélet és Szoftvertechnológiai?? Tanszék

Dolgozat címe

Témavezető:
Dr. Tóth Melinda
Egyetemi docens (PhD)

Szerző:
Fikó Róbert
programtervező informatikus BSc

Budapest, 2022

EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

SZAKDOLGOZAT TÉMABEJELENTŐ

Hallgató adatai:
Név: Fikó Róbert
Neptun kód: G55OFZ

Képzési adatok:
Szak: programtervező informatikus, alapképzés (BA/BSc/BProf)
Tagozat : Nappali

Belső témavezetővel rendelkezem

Témavezető neve: *Dr. Tóth Melinda, Bozó István*
munkahelyének neve, tanszéke: ***ELTE-IK, Programozási nyelvek és Fordítóprogramok Tanszék***
munkahelyének címe: ***1117, Budapest, Pázmány Péter sétány 1/C.***
beosztás és iskolai végzettsége: ***Egyetemi docens (PhD), egyetemi adjunktus (PhD)***

A szakdolgozat címe: RefactorErl elemző képességeinek integrálása az Erlang LS nyelvi kiszolgálóba

A szakdolgozat témája:
(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását)

Az Erlang egy futási idejű szemétgyűjtővel ellátott, konkurens funkcionális programozási nyelv. A nyelv széleskörben elterjedt a telekommunikációs alkalmazásokban, köszönhetően a gazdag OTP (Open Telecom Platform) futási-idejű környezetnek, ami rengeteg használatra kész komponenst tartalmaz. Az Erlang alkalmas nagymennyiségű adatfeldolgozásra, mert képes elosztott módban futni, továbbá hibákkal is igen toleráns.

A RefactorErl az ELTE Informatikai karán futó projekt, mely a Erlang nyelven írt források elemzésével és átalakításával foglalkozik. Legfontosabb célkitűzések a kódmegértés támogatása, a programozó segítése biztosabb, stabilabb Erlang programok írásában. Az eszköz erőssége a hatékony gráf-reprezentációban és a széleskörű szemantikus lekérdező nyelvben rejlik.

Az Erlang Language Server, vagyis nyelvi kiszolgáló (továbbiakban ELS), a Microsoft Language Server Protokolljára épülő cross-platform és cross-editor elemző rendszer, ami a legtöbb népszerű kódszerkesztővel kompatibilis. Annak ellenére, hogy az ELS egy hasznos elterjedt eszköz, a RefactorErl elemző lehetőségeivel még többre lenne képes.

Szakdolgozatomban a RefactorErl diagnosztikai és információ nyújtási képességeit kívánom integrálni az ELS nyelvi kiszolgálóba. A Language Server, az LSP protokollnak köszönhetően már sok népszerű fejlesztői környezetben támogatott, továbbá a protokollban jól definiált megjelenítési és bemenet adási lehetőségek rejlenek, aminek segítségével a programozó a szerkesztőjéből közvetlenül tud egyedi lekérdezéseket adni interaktív módon, továbbá az esetleges hibáit is egy helyen látja a forrásban.

Munkám során elsősorban a VSCode fejlesztő környezetben való alkalmazhatóságra koncentrálok. Azonban az ELS már támogatja az Emacs, SublimeText, Vim környezeteket is, így a RefactorErl funkcionalitásai itt is elérhetőek lesznek a későbbiekben.

Budapest, 2021. 11. 16.

Tartalomjegyzék

1. Bevezetés **3**

1.1. Statikus kódelemzés, illetve refaktorálás 3

1.2. RefactorErl 4

1.3. Fejlesztő környezeti nyelvi támogatás 4

1.4. Erlang LS 5

1.5. Visual Studio Code 5

1.6. Feladat 5

2. Felhasználói dokumentáció **6**

2.0.1. Rendszerkövetelmények 6

2.0.2. Telepítés 6

2.0.3. 6

3. Fejlesztői dokumentáció **7**

3.1. Megoldandó feladat 7

3.2. Használt eszközök 7

3.2.1. RefactorErl 7

3.2.2. Az Erlang nyelv 7

3.2.3. Erlang LS 7

3.2.4. A TypeScirpt nyelv 7

3.2.5. Visual Studioc Code API 7

3.3. Komponensek viszonya egymáshoz 7

3.4. LSP fölött megvalósított funkcionalitások 8

3.5. Bővítménybe ágyazott funkcionalitások 8

3.6. Forráskódok 8

3.6.1. Algoritmusok 8

4. Összegzés **10**

A. Szimulációs eredmények **11**

Irodalomjegyzék **13**

Ábrajegyzék **14**

Táblázatjegyzék **15**

Algoritmusjegyzék **16**

Forráskódjegyzék **17**

1. fejezet

Bevezetés

1.1. Statikus kódelemzés, illetve refaktorálás

Mi az a kód elemzés? Nem a programozó feladata a kódot megérteni? Ezen kérdések mind felmerülhetnek bennünk. A kódelemező egy olyan szoftver, ami képes a forráskód elemzésére, annak futtatása nélkül. Ugyan egy kicsi projektben egy elemző program használata még nem feltétlen szükséges, azonban egy ipari méretű szoftver esetében a programozók feladatát nagyban megkönnyítik.

Egy statikus elemző szoftver segítségével rengeteg információt tudhatun meg még a fejlesztés során, mint például: *kód-metrikákat* és *függőségi információkat*, melyek olyan kódrészek tekintében felettébb fontosak, amiket nem mi írtunk.

Szintén nagy méretű projekteknél fontos a refaktoráló, azaz kód átalakító eszközök használata is, hiszen manuálisan egy függvény vagy egy változó átnevezése igen nagy kihívást jelente. Azonban nem csak ilyen méretű szoftvereknél hasznos, amit esetlegesen több ember is használt, hanem egy projekt fejlesztésében, fejlődésében is jelentősége van egy ilyen, refaktoráló eszköznek. Gondoljunk bele: elkezdünk egy projektet, (ami lehet egy független szofter, vagy akár egy új funkció egy már meglévő projektben) először egy prototípust készítünk belőle, majd az alapján folytatjuk a fejlesztést, jellemzően már egy tervet követve, azonban a legalaposabb tervezés ellenére is előfordulhat, hogy egy kódot át kell alakítani.

1.2. RefactorErl

A RefactorErl az ELTE Informatikai Karán futó kutatás-fejlesztési projekt melynek fő célja az, hogy Erlang forrásokhoz statikus elemző szoftvert készítsen. Az eszközzel az elemzés mellett kódátalakítást, refaktorálást is elvégezhezünk. Főbb funkcionalitásaiként kiemelném a: függőségi gráf vizualizációját, illetve a szematnikus lekérdező nyelvét, amelyen keresztül többek közzött olyan információk érhetőek el, mint nem használt makró definicók, változó lehetséges értékei vagy biztonsági sérülékenységek[1]. Az eszköznek jelenleg több felhasználói felülete van, amelyből kiemelném a webes felületet és a Erlang Shell-en keresztül elérhető parancssori felületét.

1.3. Fejlesztő környezeti nyelvi támogatás

Napjainkban nagyon elterjedtek az integrált fejlesztői környezetek, avagy IDE[1]-k. Az ilyen fejlesztői környzetben egy szoftveren belül tudjuk szerkeszteni a forráskódot, konzolt kezelni, verziót kezelni, s a kódunkról diagnosztikákat is kapunk, szinte valós időben, amennyiben az adott programozsi nyelvhez rendelkezésre áll ilyen IDE-kompatibilis rendszer. (*Például: Visual Studio, IntelliJ, Emacs*)

A nyelvi támogatás nem jelent mást, mint egy a fejlesztői környzetbe integrált szoftvert, ami segíti a programozó munkáját a hibák és figyelmeztetések vizualizációjában, kód-kiegészítési javaslatokat is tesz, illetve a dokumentáció egy részét is elérhetővé teszi a szerkesztőn belül (*Például: az adott függvény paraméterezése, leírása*)

Sok esetben az ilyen nyelvi támogató rendszerek kifejezetten egy rendszerhez készültek, egy másik szerkesztőben pedig nem használhatóak, pedig a forrást ugyanúgy kell elemezni és a fejlesztőnek is közel azonos diagnosztikákat, visszajelzéseket kell nyújtani. Erre ad megoldást a Microsoft Language Server Protocolja (LSP), ami egy fejlesztői környezet független nyelvi kiszolgáló protokollt jellemez. Ennek az előnye, hogy a kiszolgálót csak egyszer kell megírni, és onnantól kezdve könnyedén integrálható, bármely kliensbe (amennyiben az megvalósítja a protokollt)

[1] *Integrated Developement Environment*

1.4. Erlang LS

Az Erlang Language Server (röviden: Erlang LS) a nyelvi kiszolgáló az Erlang progrmaozási nyelvhez. *Roberto Aloi* vezetésével indult el a nyílt forráskódú projekt, aminek mára már jelentős karbantartói és fejlesztői csapata van, a világ minden pontjáról, Magyarországról is. Az Erlang LS alapvetően egy szerkesztő független megvalósítás, azonban a csapat a Visual Studio Code-hoz fejlesztett egy interfész kiegészítőt is, ami segítségével az LS, szépen betud épülni a szerkesztőbe. Az elemző eszköz elérhető továbbá például Emacs, IntelliJ és még sok más IDE-ben is. Az Erlang LS támogotja a DAP[2]-ot és a legfrissebb kiadása már a RefactorErl diagnosztikák integrációját is támogtja.

1.5. Visual Studio Code

A Visual Studio Code manapság az egyik legelterjetebb kódszerkesztő szoftver, ami köszönhető nyílt forrásának, illetve annak elérhető macOS, Linux és Windows operációs rendszereken, sőt akár (béta verzióban) az interneten is. Az eszköz beépített grafikus verzió kezelést és integrált terminált is biztosít, a plugin[3] architektúrája végett, pedig szinte a kedvünkre bővíthető és személyre szabható[4].

1.6. Feladat

[2]Debugger Adapter Protocoll
[3]kiegészítő, beépülő modul
[4]Ezen kiegészítőket az alkalmazáson belül elérhető Marketplaceből tudjuk letölteni

2. fejezet

Felhasználói dokumentáció

2.0.1. Rendszerkövetelmények

2.0.2. Telepítés

Erlang/OTP telepítése

Visual Studio Code telepítése

Erlang LS bővített változatának telepítése

RefactorErl telepítése

...

RefactorErl Visualiser telepítése

2.0.3.

3. fejezet

Fejlesztői dokumentáció

3.1. Megoldandó feladat

3.2. Használt eszközök

3.2.1. RefactorErl

3.2.2. Az Erlang nyelv

3.2.3. Erlang LS

3.2.4. A TypeScirpt nyelv

3.2.5. Visual Studioc Code API

3.3. Komponensek viszonya egymáshoz

Erlang LS

RefactorErl

Visualiser

RPC

Websocket

3.4. LSP fölött megvalósított funkcionalitások

3.5. Bővítménybe ágyazott funkcionalitások

3.6. Forráskódok

```cpp
#include <stdio>

int main()
{
  int c;
  std::cout << "Hello World!" << std::endl;

  std::cout << "Press any key to exit." << std::endl;
  std::cin >> c;

  return 0;
}
```

3.1. forráskód. Hello World in C++

3.6.1. Algoritmusok

Az 1. algoritmus egy általános elágazás és korlátozás algoritmust (*Branch and Bound algorithm*) mutat be. A 3. lépésben egy megfelelő kiválasztási szabályt kell alkalmazni. Példa forrása: Acta Cybernetica (ez egy hiperlink).

1. algoritmus A general interval B&B algorithm

Funct IBB(S, f)

Set the working list $\mathcal{L}_W := \{S\}$ and the final list $\mathcal{L}_Q := \{\}$
while ($\mathcal{L}_W \neq \emptyset$) **do**
    Select an interval X from $\mathcal{L}_W$ ▷ Selection rule
    Compute $lbf(X)$ ▷ Bounding rule
    if X cannot be eliminated **then** ▷ Elimination rule
        Divide X into $X^j,\ j = 1, \ldots, p$, subintervals ▷ Division rule
        for $j = 1, \ldots, p$ **do**
            if X^j satisfies the termination criterion **then** ▷ Termination rule
                Store X^j in $\mathcal{L}_W$
            else
                Store X^j in $\mathcal{L}_W$
            end if
        end for
    end if
end while
return $\mathcal{L}_Q$

4. fejezet

Összegzés

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod. Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus porttitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. Praesent eu consequat purus.

A. függelék

Szimulációs eredmények

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque facilisis in nibh auctor molestie. Donec porta tortor mauris. Cras in lacus in purus ultricies blandit. Proin dolor erat, pulvinar posuere orci ac, eleifend ultrices libero. Donec elementum et elit a ullamcorper. Nunc tincidunt, lorem et consectetur tincidunt, ante sapien scelerisque neque, eu bibendum felis augue non est. Maecenas nibh arcu, ultrices et libero id, egestas tempus mauris. Etiam iaculis dui nec augue venenatis, fermentum posuere justo congue. Nullam sit amet porttitor sem, at porttitor augue. Proin bibendum justo at ornare efficitur. Donec tempor turpis ligula, vitae viverra felis finibus eu. Curabitur sed libero ac urna condimentum gravida. Donec tincidunt neque sit amet neque luctus auctor vel eget tortor. Integer dignissim, urna ut lobortis volutpat, justo nunc convallis diam, sit amet vulputate erat eros eu velit. Mauris porttitor dictum ante, commodo facilisis ex suscipit sed.

Sed egestas dapibus nisl, vitae fringilla justo. Donec eget condimentum lectus, molestie mattis nunc. Nulla ac faucibus dui. Nullam a congue erat. Ut accumsan sed sapien quis porttitor. Ut pellentesque, est ac posuere pulvinar, tortor mauris fermentum nulla, sit amet fringilla sapien sapien quis velit. Integer accumsan placerat lorem, eu aliquam urna consectetur eget. In ligula orci, dignissim sed consequat ac, porta at metus. Phasellus ipsum tellus, molestie ut lacus tempus, rutrum convallis elit. Suspendisse arcu orci, luctus vitae ultricies quis, bibendum sed elit. Vivamus at sem maximus leo placerat gravida semper vel mi. Etiam hendrerit sed massa ut lacinia. Morbi varius libero odio, sit amet auctor nunc interdum sit amet.

Aenean non mauris accumsan, rutrum nisi non, porttitor enim. Maecenas vel tortor ex. Proin vulputate tellus luctus egestas fermentum. In nec lobortis risus,

sit amet tincidunt purus. Nam id turpis venenatis, vehicula nisl sed, ultricies nibh. Suspendisse in libero nec nisi tempor vestibulum. Integer eu dui congue enim venenatis lobortis. Donec sed elementum nunc. Nulla facilisi. Maecenas cursus id lorem et finibus. Sed fermentum molestie erat, nec tempor lorem facilisis cursus. In vel nulla id orci fringilla facilisis. Cras non bibendum odio, ac vestibulum ex. Donec turpis urna, tincidunt ut mi eu, finibus facilisis lorem. Praesent posuere nisl nec dui accumsan, sed interdum odio malesuada.

Irodalomjegyzék

[1] .

Ábrák jegyzéke

Táblázatok jegyzéke

Algoritmusjegyzék

1. A general interval B&B algorithm 9

Forráskódjegyzék

3.1. Hello World in C++ 8