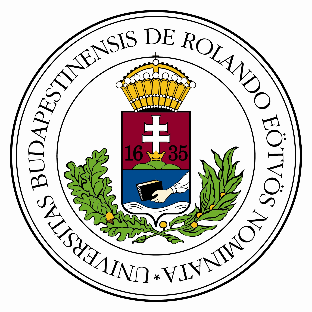
**Eötvös Loránd Tudományegyetem**

Informatikai Kar

Programozási Nyelvek és Fordítóprogramok Tanszék

**P4 programok helyességének ellenőrzése bővíthető szabályrendszer alapján**

Szerző:

Nagy Rebeka

Programtervező Informatikus BSc

Témavezető:

Tóth Gabriella

Doktorandusz

Budapest, 2020

Tartalomjegyzék

[Bevezetés 4](#_Toc39184723)

[Felhasználói dokumentáció 5](#_Toc39184724)

[Fejlesztői dokumentáció 5](#_Toc39184725)

[Szabályrendszer 5](#_Toc39184726)

[Elemzés 5](#_Toc39184727)

[Verifikáció 5](#_Toc39184728)

[Parser.hs 6](#_Toc39184729)

[Felhasználói felület 6](#_Toc39184730)

# Bevezetés

Egy szakdolgozat elkészítése remek lehetőség arra, hogy az egyetemi évek alatt megszerzett tudást kamatoztassuk, valamint, hogy új eszközöket és módszereket sajátítsunk el.

Mindig is közel éreztem magamhoz a funkcionális nyelveket, így a projektem modelljét is abban szerettem volna implementálni. A felhasználó felületek elkészítésére sok módszert tanultam különböző kurzusok alkalmával, ezek közül terveztem választani a szakdolgozatomhoz is egy architektúrát.

A P4 – gyel néhány projektmunka alkalmával ismerkedtem meg.

Ez egy napjainkban fontos, hálózati csomagok feldolgozására szolgáló eszközök programozására készített programozási nyelv. Az ellenőrzése és a tesztelése ebből kifolyólag nehéz, a fejlesztők könnyen hibát tudnak véteni a fejlesztés során.

A témám tehát egy olyan komplex szoftver elkészítése, amely képes ellenőrizni ezeket a programokat, és valamilyen visszajelzést adni arról, hogy mely részek okozhatnak nem elvárt viselkedést.

A szakdolgozatom ez alapján két nagyobb részre osztható: a verifikációs részre, és megjelenítésre.

Előbbihez tartozik a P4 szoftver szintaktikus elemzése, az előkészítése, végül az ellenőrzése adott szabályrendszer alapján. Ez a rész Haskell funkcionális nyelven írodott, így lehetőségem volt elmélyíteni a nyelvről megszerzett tudásomat, új könyvtárakat, implementálási módszereket és nyelvi szerkezeteket ismerhettem meg.

Utóbbit az egyetemen elsajátított MVVM architektúrában építettem fel, ahol az üzleti logika a verifikációs résztől kapott infomációt dolgozza fel.

# Felhasználói dokumentáció

....

# Fejlesztői dokumentáció

## Szabályrendszer

A verifikáció működésének a Tóth Gabriella és Tejfel Máté által szerzett cikk[[1]](#endnote-1) az alapja. Ez egy axiomatikus szemantikához hasonló szabályrendszer, amelynek összetettebb a környezeti struktúrája, és mellékfeltételekkel van kiegészítve.

Ezeket a feltételeket tudja a felhasználó módosítani, ezzel bővítve a programon végzett ellenőrzéseket.

....

## Elemzés

A dolgozat célja egy összetett verifikációs szoftver elkészítése, amely képes egy szintaktikailag helyes, leforduló P4 programot ellenőrizni, egy adott szabályrendszer alapján.

A programnak könnyen használhatónak, és átláthatónak kell lennie. Fontos a helyes és gyors működés is.

Mivel az ellenőrzés és a felhasználói felület két különböző nyelven és keretrendszerben van implementálva, így a köztük lévő gördülékeny kommunikációt is meg kell valósítani.

....

## Verifikáció

Az ellenőrzés két szöveget kap a felhasználói felülettől, ezeket dolgozza fel, majd a belőlük kiszámított fontos információkat küldi vissza, szintén szövegként.

A megkapott stringek közül az egyik maga a P4 program, a másik a felhasználó által megadott mellékfeltételek.

### Parser.hs

Az első lépésben a program feldolgozása történik, amelyet a Text.ParserCombinators.Parsec[[2]](#endnote-2) könyvtár, valamint egy egyszerűbb példa[[3]](#endnote-3) segítségével oldottam meg.

A nyelvtani elemzésehez szükség van a nyelv definíciójára, vagyis a kommentek jelölésének, a változónevek leírásának, az operátorok és a kulcsszavak meghatározására.

Ezzel az is definiálásra kerül, hogy a P4 milyen résznyelvével foglalkozik az ellenőrzés.

Implementálásra kerültek még bizonyos tokenek, amelyek magát az elemzést és annak olvashatóságát segítik.

#### Típusok

##### Statement

A szintaktikus elemző célja, hogy a megkapott P4 programot megfelelő részekre bontsa, amelyet az előkészítő folyamat már könnyen át tud alakítani a verifikációhoz szükséges típusokra.

Ezeket a részeket a *Statement* típus foglalja magába.

##### Variable

A *Variable* típus segítségével írjuk le mind a mezőket és headereket, mind pedig a táblák kulcsait és a *control* függvények paramétereiben szereplő változókat.

##### FunctionExpression

A programban történő függvényhívásokat *FunctionExpression*-ként írjuk le, melyek a *FuncExpr* konstruktorral alakíthatóak át *Statement*-re.

Ezek a függvényhívásokhoz tartózó változókból, valamint a függvény fajtájából tevődnek össze.

##### ArithmeticExpression

Az aritmetikai kifejezések leírására szolgáló típus. Ezt a *ParserAssignment* konstruktorral használjuk, amely a programban lévő értékadásokat írja le. Ehhez tartozik egy *string*, amely az egyenlőség bal oldalán szereplő változó nevét takarja, valamint egy *ArithmeticExpression*, amelyben az értékadás bal oldalán szereplő kifejezést tároljuk. Ez állhat egyetlen változóból, vagy bármilyen aritmetikai kifejezésből (negálás, összeadás, kivonás, szorzás, osztás).

##### BoolExpression

A logikai kifejezéseket írhatjuk le vele. Ezek a résznyelvben az elágazások feltételében szerepelnek. *Statement*-re a *BoolExpr* konstruktorral tudjuk hozni.

Állhat logikai konstansokból(igaz, hamis), valamint ezek és változók valamilyen logikai kifejezéséből(negálás, és, vagy, egyenlő, nem egyenlő, kisebb, nagyobb).

Egy P4 specifikus függvény, a *.isValid()* is ide tartozik.

#### Függvények

## Felhasználói felület

1. Cikk linkje [↑](#endnote-ref-1)
2. A könyvtár dokumentációjának linkje

   <https://hackage.haskell.org/package/parsec-3.1.14.0/docs/Text-ParserCombinators-Parsec.html> [↑](#endnote-ref-2)
3. Egyszerű imperatív nyelv elemzése

   <https://wiki.haskell.org/Parsing_a_simple_imperative_language> [↑](#endnote-ref-3)