Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Московский государственный технический университет имени Н.Э. Баумана» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)

КАФЕДРА Теоретическая информатика и компьютерные технологии (ИУ-9)

ОТЧЕТ ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ НА ТЕМУ “Вычислитель и верификатор типов функций для языка РЕФАЛ-5”

Студент Иванов Георгий Михайлович

Группа ИУ9-82

Тип практики преддипломная

Студент _____
подпись, дата *фамилия, и.о.*

Руководитель практики _____
подпись, дата *фамилия, и.о.*

Оценка _____

2019 г.

Оглавление

ВВЕДЕНИЕ

Функциональное программирование — это парадигма программирования, в которой вычисление понимается как вычисление значений функций в математическом понимании. Функция является базовым элементом, которая используется для расчётов вычислений. Даже переменные заменяются функциями. В функциональном программировании переменные — это просто синонимы (alias) для выражений, для того чтобы нам не пришлось писать всё в одну строку. Они неизменяемы. Существует много функциональных языков, и большинство из них делают одни схожие вещи по-разному: LISP (Clojure), Haskell, Scala, R.

РЕФАЛ (РЕкурсивных Функций АЛгоритмический язык) — это один из старейших функциональных языков программирования, который ориентирован на символьные преобразования: обработку символьных строк (например, алгебраические выкладки), перевод с одного языка (искусственного или естественного) на другой. РЕФАЛ был впервые реализован в 1966 году в России Валентином Турчиным, где широко используется и поныне, который соединяет в себе математическую простоту с практической ориентацией на написание больших и сложных программ. В 1970-1990 были реализованы несколько различных диалектов этого языка. В МГТУ имени Н.Э. Баумана на кафедре ИУ9 компилятор этого языка используется в качестве учебного полигона для курсовых и дипломных работ. [1]

Основа РЕФАЛа состоит в сопоставлении с образцом, подстановке и рекурсивном вызове, поэтому легко реализовать символьную обработку. Программа на РЕФАЛе вдвое-втрое короче по объему, чем аналогичная программа на языке LISP, и гораздо более читаема. В базисном Рефале выражаются через вспомогательные функции конструкции вида: if, while, for, switch.

Синтаксис РЕФАЛа не позволяет принимать функциям более одного аргумента — объектное выражение. Если в функцию передать множество аргументов, то тогда из них будет формироваться одно единое объектное выражение, которое в теле функции будет разбираться, соответствующее входным аргументам. Исходя из этого в РЕФАЛ могут возникать ошибки с типом аргументов — это передача в функцию данных, не входящих в область определения функции — в таких случаях происходит аварийный останов программы.

Актуальность работы заключается в определении области определения функции (множество объектных выражений, на которых функция не падает) и множество значений (все значения функции на области определения).

ПОСТАНОВКА ЗАДАЧИ

Целью выпускной квалификационной работы является разработка верификатора типов в РЕФАЛ, который будет проверять корректность вызовов функций в программе, т.е. аргумент в любом вызове функции должен быть совместим с форматом аргумента. При этом пользователь может псевдокомментариями подсказывать верификатору форматы некоторых функций в сложных или неоднозначных случаях, поэтому такие подсказки должны проверяться на непротиворечивость. Благодаря верификатору производится проверка корректности программ, а также повышение местности и ко-местности функций, что полезно при векторном [2] и векторно-списковом [3] представлении данных. Форматы встроенных функций должны быть вшиты в верификатор. Сам верификатор будет представлять собой консольное приложение, которое на вход будет принимать программы, написанные на РЕФАЛ-5, а также, опционально, файл(ы) специализации функций.

Входной язык верификатора будет является классический РЕФАЛ-5. Почему не Простой Рефал? В Простом Рефале есть вложенные функции, которые серьёзно усложнят анализ. Почему не Рефал-6, Рефал-7? В них есть неуспехи, которые тоже усложнят анализ. Почему не Рефал+? В нём уже есть явная статическая типизация — типы всех функций должны быть явно описаны пользователем. Далее будет понятно, что мы хотим неявную статическую типизацию для РЕФАЛа-5.

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Системные требования.

Для запуска программы необходимо следующее ПО:

- Любая ОС (GNU/Linux, MacOS, Windows)
- Python 3.6 и выше
- Библиотеку pytest (для запуска тестов)

Установка и запуск

Данную программу можно установить двумя способами:

1. Из репозитория с Github
2. Используя официальный установщик пакетов (pip)

Рассмотрим более подробнее каждый из способов.

Github — самый популярный веб-сервис для хостинга IT-проектов, основанный на системе контроля версий. Выполним команду для загрузки проекта на наш компьютер (листинг №1):

Листинг 1. Клонирование репозитория

```
MacBook-Pro-Georgij:~ geoiva$ git clone -b dev
https://github.com/runnerpeople/Refal5.git
```

Далее необходимо запустить «установщик» (листинг №2):

Листинг 2. Запуск «установщика»

```
MacBook-Pro-Georgij:~ geoiva$ python3 setup.py install
```

В Python имеется очень удобный инструмент — система скриптов установки. Документация на данный момент слишком размыта, так как имеется большой набор утилит (setuptools, distutils, distribute), выполняющие одни и те же функции. Файл дистрибьюции (setup.py) описывает метаданные и python кода проекта. Хорошим тоном является вынесение данного файла в корневую директорию проекта, который также будет мета-данные пакета и

вспомогательные файлы (лицензию, документацию и т.д.), а сам модуль программы будет являться поддиректорией проекта.

Листинг 3. Файл дистрибьюции (setup.py)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from setuptools import find_packages, setup, Command

...

setup(
    name=NAME,
    version=VERSION,
    description=DESCRIPTION,
    long_description=long_description,
    long_description_content_type='text/markdown',
    author=AUTHOR,
    author_email=EMAIL,
    python_requires=REQUIRES_PYTHON,
    url=URL,
    packages=find_packages(),
    include_package_data=True,

    install_requires=REQUIRED,
    extras_require=EXTRAS,
    license='MIT',
    ...
    cmdclass={
        'upload': UploadCommand,
        'test': TestCommand
    },
    entry_points={
        'console_scripts': ['refalcheck=refal.refalcheck:main'],
    },
    tests_require=["pytest"],
    test_suite='tests'
)
```

Файл setup.py (листинг №3) обязательно содержит важную функцию *setup*, которая и выполняет установку пакета. Параметрами этой функции могут быть название модуля, версии программы, описание, данные об авторе, URL проекта, необходимые модули, необходимая версия Python, лицензия, а также собственные команды для setup.py (в листинге №1 описаны дополнительные команды *upload*, которая делает выгрузку модуля в PyPi, и *test*, которая запуск тесты) и собственные «точки вызова» программы (создание скриптов на уровне /usr/local/bin) и т.д. Существует множество

систем наименования версий программы в python, но обычно рекомендуется использовать PEP386. Чаще всего обозначение версии состоит из номера мажорного, минорного релизов, разделенных точками. В последней части версии разрешается использовать буквы латинского алфавита (для бета-версии используется “b”). Также для установки модуля, содержащего не только python-файлы, необходимо создать файл MANIFEST.in и указать в нём все файлы, которые будут необходимы при использовании нашей программы.

Если же данный способ является несколько сложным, то можно использовать установщик пакетов pip. Данный проект был опубликован на сайте PyPi.org, при помощи файла дистрибьюции (setup.py). Для скачивания проекта в этом случае понадобится (листинг №4):

Листинг 4. Скачивание модуля из PyPi

```
MacBook-Pro-Georgij:~ geoiva$ pip3 install refalchecker
```

В любом случае установки мы получим консольное приложение (refalcheck). На вход подается имя файла *.ref, который содержит программу, написанную на РЕФАЛ-5, и файлы *.type, содержащие форматы используемых функций. Например (листинг 5):

Листинг 5. Пример запуск программы

```
MacBook-Pro-Georgij:~ geoiva$ refalcheck R05-Parser.ref LibraryEx.type R05-
Lexer.type
```

В файле constants.py используются константы, конфигурируемые при запуске верификатора (например, для вывода отладочной информации).

Тестирование

Для проверки работоспособности верификатора необходимо можно запустить пакетное тестирование (листинг №6):

Листинг 6. Пакетное тестирование.

```
MacBook-Pro-Georgij:~ geoiva$ python3 setup.py test
```

При успешной работе вывод должен быть таким (листинг №7):

Листинг 7. Вывод работы пакетного тестирования.

```
MacBook-Pro-Georgij:~ geoiva$
============================================================================
========== test session starts
============================================================================
==========

platform darwin -- Python 3.7.0, pytest-4.5.0, py-1.8.0, pluggy-0.11.0
rootdir: /Users/geoiva/Desktop/Учеба/Учеба (8 сем)/Диплом
collected 29 item

        tests/refal_test.py .
[100%]

============================================================================
======== 29 passed in 14.32 seconds
============================================================================
=========
```

Использование

Как говорилось ранее, есть разные режимы работы верификатора. Для подробной отладочной информации (списки токенов и др.), необходимо указать в файле constants.py значение переменной *DEBUG_MODE* в `True`. Во время работы верификатора происходит лексический, синтаксический и семантический анализ программы на РЕФАЛ-5. Оговорим необходимые правила для корректной работы верификатора:

- Все имена функций после < или встроенные, или описанные в текущем файле, или объявленные как $EXTERN.
- Для каждой переменной в результатном выражении должно быть её вхождение в образцовом выражении в предшествующей части предложения. Т.е. при чтении предложения образцовые части пополняют область видимости новыми именами переменных (включая тип, s.Var и t.Var — две разные переменные), в результатных частях могут быть переменные только из области видимости.

- Функция в исходном тексте должна быть определена один раз — или как функция, или как $EXTERN.
- В тексте спецификации типов семантический анализатор должен только проверять, что имена функций не повторяются — нет двух разных определений типов для одной функции. Больше ничего проверять не нужно.
- В паре файлов Рефал-5 и спецификация типов семантический анализ должен проверять, что все функции, для которых заданы спецификации типов, определены в исходном файле.

При несоблюдении одного из них верификатор предупредит об ошибке и дальнейшая работа верификатора прекратится. Если же на этапе вычисления типов верификатор возникает конфликтная ситуация (невозможно вычислить тип), то верификатор продолжает работать (по алгоритму вывода типов). Все сообщения о невозможности вычисления типа функций будут выведены на экран в конце работы программы верификатора.

Для. определения типа функции для верификатора спецификация функции будет иметь вид:

$$func\_name\ format\_in = format\_out$$

где *format_in* и *format_out* – объектные «жесткие» выражения, а *func_name* – название функции.

Рассмотрим пример работы программы. Например, дан файл (листинг №8) и файл спецификации типов (листинг №9):

Листинг №8. Пример программы на РЕФАЛ-5

```
*$FROM LibraryEx
$EXTERN MapAccum;

$ENTRY generator_GenSentence {
  ((e.Pattern) (e.Result)) =
    ()
    ('  do {')
    <SkipIndentAccum
      <MapAccum
        generator_GenCommand
        ('    ' /* отступ */)
        <CompileSentence (e.Pattern) (e.Result)>
      >
    >
    ('  } while (0);');

  ReturnRecognitionImpossible =
    ()
    ('  r05_recognition_impossible();');
}

SkipIndentAccum {
  ('    ') e.Generated = e.Generated;
}

CompileSentence {
  (e.Pattern) (e.Result) = e.Pattern e.Result;
}
```

Листинг №9. Пример файла спецификации типов

```
SaveFile (e.FileName) e.Lines = ;
MapAccum t.Func t.Acc e.Terms = t.Acc e.Res;
Map t.Func e.Terms = e.Res;
```

То после работы верификатора вывод будет таким:

```
/usr/local/bin/python3.7 "/Users/geoiva/Desktop/Учеба/Учеба (8
сем)/Диплом/refal/refalcheck.py" "/Users/geoiva/Desktop/Учеба/Учеба (8
сем)/Диплом/tests/files/test18.ref" "/Users/geoiva/Desktop/Учеба/Учеба (8
сем)/Диплом/tests/files/test18.type"
Ok. Program satisfy grammar
Ok. Program-Type satisfy grammar
Ok. Program-Type satisfy grammar
/* result program refalcheck */

generator_GenSentence t = () ('  ' s s s s e) e
*SkipIndentAccum ('    ') e = e
*CompileSentence (e) (e) = e
```

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.

1. Алгоритмический язык рекурсивных функций (РЕФАЛ) В.Ф. Турчин, ИПМ АН СССР, 1968.
2. Представление объектных выражений массивами при реализации языка Рефал. С.М.Абрамов, С.А.Романенко. М.:ИПМ им.М.В.Келдыша АН СССР, 1988, препринт N 186.
3. Реализация Рефал-компилятора. Представление данных и язык сборки. Скоробогатов С. Ю. Москва, 2008.
4. Язык программирования Рефал Плюс *Курс лекций. Учебное пособие для студентов университета города Переславля.* Р. Гурин, С. Романенко, Университет города Переславля» им. А. К. Айламазяна: 2006.
5. Исходный код приложения refalchecker. URL: https://github.com/runnerpeople/Refal5