ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ И МЕХАНИКИ
ВЫСШАЯ ШКОЛА ПРИКЛАДНОЙ МАТЕМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ФИЗИКИ

Отчет о прохождении преддипломной практики на тему: «Эксперементы с реализацией языка охраняемых команд Дейкстры»

Соломатина Макара Александровича, гр. 3630102/70201

Направление подготовки: 01.03.02 Прикладная математика и информатика.
Место прохождения практики: СПбПУ, ИПММ.
Сроки практики: с 10.05.2021 по 01.06.2021.
Руководитель практики от ФГАОУ ВО «СПбПУ»: Новиков Федор Александрович, профессор высшей школы прикладной математики и вычислительной доктор технических наук.
Консультант практики от ФГАОУ ВО «СПбПУ»: Новиков Федор Александрович, доктор технических наук, старший научный сотрудник.
Оценка: _____

Руководитель практики
от ФГАОУ ВО «СПбПУ» Ф.А. Новиков

Консультант практики
от ФГАОУ ВО «СПбПУ» Ф.А. Новиков

Обучающийся М.А. Соломатин

Дата: 01.06.2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В мире современного программирования все более актуальными становятся методы формальной верификации программ. Однако этот процесс является крайне трудоемким, и его стоимость настолько высока, что он применяется лишь в критических местах программного обеспечения, в котором цена ошибки слишком велика: ракетостроение, военная оборона, медицина и т.п. Поэтому, и по некоторым иным причинам, часто прибегают к тестированию программ, которое, как известно, доказывает корректность программ только в некоторых частных случаях. Подобная сложность обусловлена, по-видимому, незрелостью разработанного аппарата для описания программных моделей и соответствующей семантики программ. Э. Дейкстра в своей работе (цитата) рассматривает денотационный способ формализации семантики программ. Он предлагает теоретический язык программирования, доказывает несколько важных утверждений про его семантику и снабжает его множеством примеров. Однако практической реализации в виде интерпретатора языка, и вместе с ней – инструментария для анализа семантики программы, не существует или не было опубликовано. Подобная реализация позволила бы строить недетерминированные алгоритмы и проверять их корректность полуавтоматически, а в некоторых случаях и вовсе без участия программиста.

Формальную верификацию готовых программ, как и предварительное построение корректных программ может быть сделано и для других, распространненных языков программирования, таких как Си. Однако они обладают значительно более сложной семантикой операторов и сложность применения формальных методов значительно выше.

Объектом исследования работы является разработка интерпретатора недетерминированного языка, в основе которого лежит понятие охраняемой команды, а также статический анализ кода программы и вывод ее семантики. Неотъемлимой частью исследования является применение разработанного инструментария к реальным примерам, а также разработка алгоритмов методами, предложенным Э. Дейкстрой.

Предметом исследования работы является создание интерпретируемого недетерминированного языка охраняемых команд Дейкстры. Язык был предложен Э.Дейкстрой с тем, чтобы строить программы, являющиеся корректными по построению.

Автоматический синтез корректных программ не является предметом исследования работы, однако статический анализ программ, полуавтоматически выводящий их семантику, может служить полезным и удобным инструментом для формальной верификации.

Целью исследований является разработка синтаксиса языка охраняемых команд и интерпретатора для программ, построенных на этом языке.

Для достижения этой цели необходимо решить следующие задачи:

A. описать язык формально – задать синтаксис, определить основные операторы языка
B. разработать интерпретатор программ на языке охраняемых команд, исполняющий ее и выводящий результат
C. разработать программу-анализатор, позволяющей по исходному тексту программы выводить предусловие, если постусловие задано пользователем
D. исследовать приведенные в (цитата) примеры, сравненить методы прямого и обратного преобразования предикатов

Теоретической основой выпускной квалификационной работы послужили исследования Э. Дейкстры в своем труде «Дисциплина программирования», дополненные результатами, полученными С.С. Лавровым в книге «Математические основы, средства, теория». Практическая часть работы выполнялась на основании примеров, предложенных Э. Дейкстрой, и некоторых других алгоритмах.

ГЛАВА 1. ДЕНОТАЦИОННАЯ СЕМАНТИКА И ОХРАНЯЕМЫЕ КОМАНДЫ

В этой главе рассматривается денотационный способ задания семантики языка программирования в терминах преобразования предикатов, а также описывается семантика операторов языка охраняемых команд Дейкстры.

1.1. Семантика языка программирования

Построение семантики языка программирования – формальное описание смысла, придаваемого его элементам – выражениям, предложениям, операторам. На настоящий момент известны следующие способы определения семантики языка программирования:

A. Операционная семантика
B. Аксиоматическая, или деривационная семантика
C. Денотационная семантика

В основе денотационной семантики является сопоставления синтаксическим единицам языка математических объектов – множеств и отоношений. Денотационная семантика оператора описывает связь между двумя состояниями:

– *начальным* – состоянием программы перед выполнением оператора
– *конечным* – состоянием программы после выполнения оператора

1.2. Множество состояний

Любая программа, предназначенная для выполнения на вычислительном устройстве, оперирует с определенным множеством ячеек памяти – записывает в них и считывает из них. Назовем это множество ячеек памяти, или как принято называть переменных, состоянием программы. Нетрудно заметить, что множеством всех возможных состояний программы является прямое произведение областей значений каждой переменной. Программа может изменять состояние программы как изменением значений имеющихся переменных, так и добавлением новых переменных. Экспоненциальный рост числа состояний программы с количеством переменных, также известный как проклятие размерности, дает основание считать множество возможных состояний практически несчетным.

Для описания подмножеств множества состояний используется исчисление предикатов первого порядка, в которых формулы содержат свободные термы – переменные состояния. Таким образом каждому предикату соответствует некоторое множество состояний, в интерпретации которых формула истинна.

1.3. Преобразование предикатов

Преобразователь предикатов – это функциональное отношение на множестве предикатов. В рамках денотационной семантики, каждой программе S языка охраняемых команд сопоставляется преобразователь предикатов p, который по предикату R определяет предикат $p(S, R)$.

1.4. Предусловие и постусловие

Рассмотрим некоторую программу S языка охраняемых команд. Предикаты R и $p(S, R)$ называются *постусловием* и *предусловием* программы S соответственно, если программа S, запущенная в состоянии, удовлетворяющем $p(S,R)$, обязательно завершится и после своего выполнения останется в состоянии, удовлетворяющем предикату R.

Предикат $wp(S, R)$ называется *слабейшим предусловием*, если он характеризует множество *всех* состояний, запуск программы S из которых обязательно приведет к завершению программы и оставит ее в состоянии, удовлетворяющем предикату R.

Предикат $wlp(S, R)$ называется *слабейшим свободным предусловием*, если он характеризует множество *всех* состояний, запуск программы S из которых, если он приведет к завершению программы, оставит ее в состоянии, удовлетворяющем предикату R.

1.4.1. Закон исключенного чуда

Имеет место следующее свойство преобразователя предикатов, называемое *правилом исключенного чуда*: для любой программы S

$$wp(S, F) = F \tag{1.1}$$

где предикат F – противоречие.

Действительно, если $wp(S, F) \neq F$, то найдется такое начальное состояние, удовлетворяющие $wp(S, F)$, такое, что программа после своего выполнения останется в состоянии, которой должно удовлетворять F, т.е. F должен быть истинен в интерпретации конечного состояния программы. Однако такого не может быть, поскольку предикат F ложен в любой интерпретации.

1.5. Выводы

Текст выводов по главе 1.

Кроме названия параграфа «выводы» можно использовать (единообразно по всем главам) следующие подходы к именованию последних разделов с результатами по главам:

- «выводы по главе N», где N — номер соответствующей главы;
- «резюме»;
- «резюме по главе N», где N — номер соответствующей главы.

Параграф с изложением выводов по главе *является обязательным*.

ГЛАВА 2. ОПИСАНИЕ ЯЗЫКА

В этой главе изложено синтаксическое описание языка охраняемых команд наряду с его семантикой. Для этого поэтапно вводятся используемые в нем операторы, понятие охраняемой команды, а в приложении находятся полное синтаксическое описание языка с помощью расширенной формы Бэкуса-Наура (РБНФ) и синтаксических диаграмм Вирта.

2.1. Оператор «skip»

Оператор *skip* ничего не делает и оставляет программму в том же состоянии, какое было и перед ее выполнением. Поэтому

$$wp(skip, R) = R \tag{2.1}$$

2.2. Оператор «reject»

Оператор *reject* не может завершиться ни в каком начальном состоянии. Поэтому

$$wp(reject, R) = False \tag{2.2}$$

2.3. Оператор присваивания

Оператор присваивания связывает значение некоторого выражения E с переменной x:

$$x := E \tag{2.3}$$

Пусть R – постусловие, а предикат $R_{E \to x}$ получен из R подстановкой выражения E вместо всех вхождений переменной x. Тогда

$$wp(x := E, R) = R_{E \to x} \tag{2.4}$$

2.3.1. Множественное присваивание

Вполне естественным расширением оператора присваивания является оператор множественного присваивания:

$$x_1, x_2 \ldots, x_n := E_1, E_2 \ldots, E_n \tag{2.5}$$

Он вычисляет значения выражений $E_i,\ i = 1{,}2 \ldots, n$ и связывает соответственно переменные $x_1, x_2 \ldots, x_n$ с их значениями. Тогда

$$wp(x_1, x_2 \ldots, x_n := E_1, E_2 \ldots, E_n, R) = R_{E_1 \to x_1, E_2 \to x_2 \ldots, E_3 \to x_3} \tag{2.6}$$

Предикат $R_{E_1 \to x_1, E_2 \to x_2 \ldots, E_3 \to x_3}$ получается из предиката R подстановкой выражений $E_1, E_2 \ldots, E_3$ вместо всех вхождений переменных $x_1, x_2 \ldots, x_n$ соответственно.

2.3.2. Литералы и типы выражений

В языке охраняемых команд, по крайней мере в упрощенном его варианте, есть выражений двух видов: численные и логические. Вычисление численного выражения производит целое или вещественное число, а вычисление логического выражения производит True или False.

В языке есть следующие литералы:

A. Целые числа, текстовое представление в исходном коде которых описывается регулярным выражением $[1-9][0-9]*$
B. Вещественные числа, текстовое представление в исходном коде которых описывается регулярным выражением $[1-9][0-9]*'.'[0-9]+$
C. Логические значения, текстовое представление в исходном коде которых – *True* или *False*

2.4. Композиция операторов

До настоящего момента рассматривались только однооператорные программы. Рассмотрим возможность составлять программы из нескольких операторов. Композиция операторов – программа вида

$$S1; S2$$

При композиция операторов обладает следующей семантикой:

- Выполняется оператор S1
- Если выполнение оператора S1 завершилось, тогда выполняется оператор S2 с начальным состоянием, равным конечному состоянию оператора S1

Сопоставим композиции операторов ее преобразователь предикатов. Пусть R – постусловие композиции операторов $S_1; S_2$. Это значит, что выполнение оператора S_2 должно оставить программу в состоянии, удовлетворющем предикату R. Тогда предусловием оператора S_2 будет являться $wp(S_2,R)$. Это то состояние, в котором должна находиться программа после выполнения оператора S_1, значит будет являться также и постусловием оператора S_1. Соответствующее предусловие оператора S_1 есть $wp(S_1, wp(S_2, R)))$. Итак, получили преобразователь предикатов для композиции операторов:

$$wp(S_1; S_2, R) = wp(S_1, wp(S_2, R)) \tag{2.7}$$

В практическом смысле это означает, что для получения предусловия композиции двух операторов необходимо последовательно получить предусловия каждого оператора начиная с последнего; этот процесс называется *обратным выводом*.

Несложным образом, определяя трехместную композицию операторов $S_1; S_2; S_3$ как $S_1; (S_2; S_3)$ получим следующий преобразователь предикатов для нее

$$wp(S_1; S_2; S_3, R) = wp(S_1, wp(S_2, wp(S_3, R))) \tag{2.8}$$

Аналогичным образом строится преобразователь предикатов для n-местной композиции $S_1; S_2; \ldots; S_n$.

Заметим, что композиция операторов является ассоциативной:

– $wp(S_1; (S_2; S_3), R) = wp(S_1, wp(S_2; S_3, R)) = wp(S_1, wp(S_2, wp(S_3, R)))$
– $wp((S_1; S_2); S_3, R) = wp(S_1; S_2, wp(S_3, R)) = wp(S_1, wp(S_2, wp(S_3, R)))$

2.5. Охраняемая команда

Понятие охраняемой команды является центральным в языке охраняемых команд. Охраняемой командой называется конструкция вида

$$B \to S \tag{2.9}$$

где B – логическое выражение, называемое *предохранителем*, а S – оператор, называемый командой.

Оператор S может быть выполнен только в том случае, если выражение–предохранитель B имеет истинное значение на текущем состоянии прогарммы, т.е. на состоянии, в котором программа приступила к выполнению охраняемой команды.

Сама охраняемая команда не является оператором языка, но служит составной часть для структурных операторов – *ветвления* и *цикла*.

2.6. Условный оператор

Условный оператор определим как оператор, состоящий из одной или нескольких охраняемых команд, заключенных в лексемы $if \ldots fi$

$$\textbf{if } B_1 \to S_1 \mid B_2 \to S_2 \mid \ldots \mid B_n \to S_n \textbf{ if} \tag{2.10}$$

Условный оператор обладает следующей семантикой:

A. Если ни один из предохранителей $B_1, B_2, \ldots, B_n$ не имеет истинное значение на текущем состоянии программы, то выполнение условного оператора не приводит к завершению программы. В этом случае он эквивалентен оператору *reject*.

B. Пусть предохранители $B_{i_1}, B_{i_2}, \ldots, B_{i_m}$, где $\{i_1, i_2, \ldots, i_m\}$ – размещение множества $\{1, 2, \ldots, n\}$, имеют истинное значение на начальном состоянии условного оператора. Тогда случаным образом выбирается к исполнению оператор из множества $\{S_{i_1}, S_{i_2}, \ldots, S_{i_m}\}$.

Недетерминированности выбора оператора для исполнения не возникает в случае, когда все предохранители попарно исключают друг друга. Детерминированный вариант условного оператора выбирает первую команду, чей предохранитель имеет истинное значение.

Построим преобразователь предикатов, соответствующей условному оператору. Пусть задано постусловие R условного оператора

$$S_{if} = \textbf{if } B_1 \to S_1 \mid B_2 \to S_2 \mid \ldots \mid B_n \to S_n \textbf{ if} \tag{2.11}$$

Для завершения выполнения оператора необходимо, чтобы хотя бы один предохранитель был истинен на начальном состоянии, откуда приходим к следующему предикату:

$$B_1 \vee B_2 \vee \ldots \vee B_n \tag{2.12}$$

Помимо этого, каждый оператор, чей предохранитель истинен, должен иметь конечное состояние, удовлетворющее R. Поэтому получаем следующее предусловие для условного оператора:

$$wp(S_{if}, R) = (B_1 \vee B_2 \vee \ldots \vee B_n) \wedge (\forall i : B_i \Rightarrow wp(S_i, R)) \tag{2.13}$$

2.7. Оператор цикла

2.8. Макроподстановки

2.9. Выводы

Текст заключения ко второй главе. Пример ссылок [**Article**; **Book**; **Booklet**; **Conference**; **Inbook**; **Incollection**; **Manual**; **Mastersthesis**; **Misc**; **Phdthesis**; **Proceedings**; **Techreport**; **Unpublished**; **badiou:briefings**], а также ссылок с указанием страниц, на котором отображены те или иные текстово-графические объекты [**Naidenova2017**] или в виде мультицитаты на несколько источников [**Naidenova2017**; **Ganter1999**]. Часть библиографических записей носит иллюстративный характер и не имеет отношения к реальной литературе.

Короткое имя каждого библиографического источника содержится в специальном файле `my_biblio.bib`, расположенном в папке `my_folder`. Там же

находятся исходные данные, которые с помощью программы `Biber` и стилевого файла `Biblatex-GOST` [**ctan-biblatex-gost**] приведены в списке использованных источников согласно ГОСТ 7.0.5-2008. Многообразные реальные примеры исходных библиографических данных можно посмотреть по ссылке [**ctan-biblatex-gost-examples**].

Как правило, ВКР должна состоять из четырех глав. Оставшиеся главы можно создать по образцу первых двух и подключить с помощью команды `\input` к исходному коду ВКР. Далее в приложении **??** приведены краткие инструкции запуска исходного кода ВКР [**latex-miktex**; **latex-texstudio**].

В приложении **??** приведено подключение некоторых текстово-графических объектов. Они оформляются по приведенным ранее правилам. В качестве номера структурного элемента вместо номера главы используется «П» с номером главы. Текстово-графические объекты из приложений не учитываются в реферате.

ГЛАВА 3. РАЗРАБОТКА ИНТЕРПРЕТАТОРА И АНАЛИЗАТОРА

Хорошим стилем является наличие введения к главе. Во введении может быть описана цель написания главы, а также приведена краткая структура главы.

3.1. Название параграфа

3.2. Название параграфа

3.3. Выводы

Текст выводов по главе 3.

ГЛАВА 4. ЭКСПЕРИМЕНТЫ С ЯЗЫКОМ ОХРАНЯЕМЫХ КОМАНД

4.1. Выводы

Текст выводов по главе 4.

ЗАКЛЮЧЕНИЕ

Заключение (2 – 5 страниц) обязательно содержит выводы по теме работы, *конкретные предложения и рекомендации* по исследуемым вопросам. Количество общих выводов должно вытекать из количества задач, сформулированных во введении выпускной квалификационной работы.

Предложения и рекомендации должны быть органически увязаны с выводами и направлены на улучшение функционирования исследуемого объекта. При разработке предложений и рекомендаций обращается внимание на их обоснованность, реальность и практическую приемлемость.

Заключение не должно содержать новой информации, положений, выводов и т. д., которые до этого не рассматривались в выпускной квалификационной работе. Рекомендуется писать заключение в виде тезисов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ