Пример ВКР «SPbPU-student-thesis-template». Версия от 8 июня 2021 г.

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт прикладной математики и механики

Работа допущена к защите
Руководитель образовательной программы «Прикладная математика и информат
_____ К.Н. Козлов
«_____» _____ 2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
РАБОТА БАКАЛАВРА
ЭКСПЕРЕМЕНТЫ С РЕАЛИЗАЦИЕЙ ЯЗЫКА ОХРАНЯЕМЫХ КОМАНД ДЕЙКСТРЫ

по направлению подготовки 01.03.02 Прикладная математика и информатика
Направленность (профиль) 01.03.02_02 Системное программирование

Выполнил
студент гр. 3630102/70201

Руководитель
профессор высшей школы прикладной математики и вычислительной физики,
доктор технических наук, старший научный сотрудник[1]

Консультант[2]
доктор технических наук, старший научный сотрудник

Консультант
по нормоконтролю[3]

Санкт-Петербург
2021

[1]Должность указывают сокращенно, учёную степень и звание — при наличии, а подразделения — аббревиатурами. «СПбПУ» и аббревиатуры институтов не добавляют.

[2]Оформляется по решению руководителя ОП или подразделения. Только 1 категория: «Консультант». В исключительных случаях можно указать «Научный консультант» (должен иметь степень). Без печати и заверения подписи.

[3]Обязателен, из числа ППС по решению руководителя ОП или подразделения. Должность и степень не указываются. Сведения помещаются в последнюю строчку по порядку. Рецензенты не указываются.

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт прикладной математики и механики

УТВЕРЖДАЮ
Руководитель образовательной программы
«Прикладная математика и информатика»
_____ К.Н. Козлов
«_____» _____ 2021г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы студенту

Соломатину Макару Александровичу гр. 3630102/70201

1. Тема работы: Эксперементы с реализацией языка охраняемых команд Дейкстры.
2. Срок сдачи студентом законченной работы: июнь 2021.
3. Исходные данные по работе:
   - Синтаксическое описания языка охраняемых команд
   - Денотационная семантика языка охраняемых команд
   - Теоретические примеры программ

   Инструментальные средства:
   - Язык программирования Python
   - Генератор лексических и синтаксических анализаторов ANTLR4
   - Библиотеки python-antlr4 и sympy

   Ключевые источники литературы:
   - Дейкстра Э. Дисциплина программирования, 1978.
   - Лавров С.С. Программирование. Математические основы, средства, теория.
   - Карпов Ю.Г. Model Checking. Верификация параллельных и распределенных программных систем, 2009.
4. Содержание работы (перечень подлежащих разработке вопросов):
   4.1. Постановка задачи
   4.2. Обзор литературы по теме ВКР

4.3. Исследование программных продуктов
4.4. Разработка интерпретатора и анализатора
4.5. Эксперементы с программами на реализованном языке
4.6. Выводы

5. Дата выдачи задания: 20.12.2020.

Руководитель ВКР _____Ф.А. Новиков

Задание принял к исполнению 20.12.2020

Студент _____М.А. Соломатин

РЕФЕРАТ

На 28 с., 0 рисунков, 1 таблицу, 2 приложения

КЛЮЧЕВЫЕ СЛОВА: РАЗРАБОТКА ИНТЕРПРЕТАТОРА, ЯЗЫК ОХРАНЯЕМЫХ КОМАНД, ФОРМАЛЬНАЯ ВЕРИФИКАЦИЯ ПРОГРАММ, ПОШАГОВОЕ УТОЧНЕНИЕ.[4]

Тема выпускной квалификационной работы: «Эксперементы с реализацией языка охраняемых команд Дейкстры»[5].

В данной работе изложена сущность подхода к созданию динамического информационного портала на основе использования открытых технологий Apache, MySQL и PHP. Даны общие понятия и классификация IT-систем такого класса. Проведен анализ систем-прототипов. Изучена технология создания указанного класса информационных систем. Разработана конкретная программная реализация динамического информационного портала на примере портала выбранной тематики...[6]

В данной работе изложена сущность подхода к созданию динамического информационного портала на основе использования открытых технологий Apache, MySQL и PHP. Даны общие понятия и классификация IT-систем такого класса. Проведен анализ систем-прототипов. Изучена технология создания указанного класса информационных систем. Разработана конкретная программная реализация динамического информационного портала на примере портала выбранной тематики...

ABSTRACT

28 pages, 0 figures, 1 tables, 2 appendices

[4]Всего **слов**: от 3 до 15. Всего **слов и словосочетаний**: от 3 до 5. Оформляются в именительном падеже множественного числа (или в единственном числе, если нет другой формы), оформленных по правилам русского языка. *Внимание! Размещение сноски после точки является примером как запрещено оформлять сноски.*

[5]Реферат **должен содержать**: предмет, тему, цель ВКР; метод или методологию проведения ВКР: результаты ВКР: область применения результатов ВКР; выводы.

[6]ОТ 1000 ДО 1500 печатных знаков (ГОСТ Р 7.0.99-2018 СИБИД) на русский или английский текст. Текст реферата повторён дважды на русском и английском языке для демонстрации подхода к нумерации страниц.

KEYWORDS: INTERPRETER DEVELOPMENT, GUARDED COMMAND LANGUAGE, PROGRAM FORMAL VERIFICATION, STEPWISE REFINEMENT.

The subject of the graduate qualification work is «Experiments with the implementation of the Dijkstra's guarded command language».

In the given work the essence of the approach to creation of a dynamic information portal on the basis of use of open technologies Apache, MySQL and PHP is stated. The general concepts and classification of IT-systems of such class are given. The analysis of systems-prototypes is lead. The technology of creation of the specified class of information systems is investigated. Concrete program realization of a dynamic information portal on an example of a portal of the chosen subjects is developed...

In the given work the essence of the approach to creation of a dynamic information portal on the basis of use of open technologies Apache, MySQL and PHP is stated. The general concepts and classification of IT-systems of such class are given. The analysis of systems-prototypes is lead. The technology of creation of the specified class of information systems is investigated. Concrete program realization of a dynamic information portal on an example of a portal of the chosen subjects is developed...

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, распространенных 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

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 – пе-

ременные 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.

1.5. Свойства преобразователя предикатов

Далее рассмотрены важные свойства преобразователя предикатов $wp(S,R)$.

1.5.1. Правило 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.

1.6. Выводы

Итак, описана денотационная семантика языка программирования с помощью сопоставления каждой программе преобразователя предикатов. Получение семантики программы позволяет проверить или доказать корректность составленной программы. Однако процесс построения денотационной семантики может быть крайне затруднительным, как будет видно из следующей главы.

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 и понятие охраняемой команды.

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 случайным 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:

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

В практическом смысле это означает, что для нахождения предусловия условного оператора необходимо вычислить предусловия всех охраняемых команд, и подставить их в формулу (2.13).

2.7. Оператор цикла

Оператор цикла определим как оператор, состоящий из одной или нескольких охраняемых команд, заключенных между лексемами $do \ldots od$:

$$\textbf{do } B_1 \to S_1 \mid B_2 \to S_2 \mid \ldots \mid B_n \to S_n \textbf{ od} \tag{2.14}$$

Семантика оператора цикла следующая:

A. Если ни один из предохранителей $B_1, B_2, \ldots, B_n$ не имеет истинного значения на текущем состоянии программы, то оператор цикла завершает свою работу

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 оператора цикла. Тогда случайным образом выбирается к исполнению оператор из множества $\{S_{i_1}, S_{i_2}, \ldots, S_{i_m}\}$. Затем заново вычисляются истинностные значения всех предохранителей на новом состоянии программы, полученном после выполнения одной из охраняемых команд, и выполняется переход к предыдущему шагу.

Соответствующий преобразователь предикатов определяется следующим образом выражается через преобразователь предикатов для условного оператора, имеющего те же охраняемые команды.

Пусть

$$wp(S_{do}, R) = \textbf{do}\ B_1 \to S_1 \mid B_2 \to S_2 \mid \ldots \mid B_n \to S_n\ \textbf{od} \tag{2.15}$$

$$wp(S_{if}, R) = \textbf{if}\ B_1 \to S_1 \mid B_2 \to S_2 \mid \ldots \mid B_n \to S_n\ \textbf{if} \tag{2.16}$$

Дадим рекурсивное определение предикатов $H_0(R),\ k = 1, 2 \ldots$:

$$H_0(R) = R \wedge \neg(\exists j : 1 \leqslant j \leqslant n : B_j) \tag{2.17}$$

Для любого $k > 0$:

$$H_k(R) = wp(S_{if}, H_{k-1}(R)) \vee H_0(R) \tag{2.18}$$

Тогда преобразователь предикатов для S_{do} определяется как

$$wp(S_{do}, R) = (\exists k : k \geqslant 0 : H_k(R)) \tag{2.19}$$

Предикат $H_k(R)$ ялвяется слабейшим предусловием того, что оператор цикла S_{do} завершится не более чем за k выборок.

Формула (2.17) означает предусловие, в котором должна находиться программа в случае, если все предохранители ложны. В этом случае не производится вычисление ни одной охраняемой команды.

Формула (2.18) при $k > 0$ имеет два дизъюнктивных слагаемых: либо все предохранители оказались ложны, и это произошло меньше чем за k выборок, либо есть истинные предохранители, и тогда выполняются действия, семантически эквивалентные однократному оператору S_{if}. После этой выборки программа должна оказаться в таком состоянии, чтобы далее потребовалось не более чем $k - 1$ выборка охраняемой команды для завершения цикла в состоянии, удовлетворющем R.

Формальное определение семантики оператора цикла не имеет большого практического смысла, так как заблаговременно определить количество итераций цикла k представляется невозможным.

2.7.1. Инвариант цикла

Если удается найти предикат, не изменяющий свою истинность после выполнения оператора S_{if}, и при этом истинный до начала исполнения оператора

S_{do}, более того достаточно сильный, чтобы из него следовало постусловие цикла, то преобразователь предикатов для оператора цикла можно определить гораздо проще, чем в формуле (2.19). Данные рассуждения следуют из *основной теоремы для оператора цикла* (цитата).

Теорема. Пусть оператор цикла S_{do} и предикат P таковы, что предикат

$$(P \wedge (\exists i : 1 \leqslant i \leqslant n : B_i)) \Rightarrow wp(S_{if}, P) \tag{2.20}$$

справедлив для всех состояний. Тогда для оператора цикла S_{do} предикат

$$(P \wedge wp(S_{do}, T)) \Rightarrow wp(S_{do}, P \wedge \neg(i : 1 \leqslant i \leqslant n : B_j)) \tag{2.21}$$

где T – тавтология, также справедлив для всех состояний.

Предикат $w(S_{do}, T)$ характеризует начальные состояний, для которых оператор цикла завершает свою работу. Основная теорема для оператора цикла позволяет считать предикат $P \wedge wp(S_{do}, T)$ предусловием цикла, если:

A. Удается доказать, что цикл завершается. Другими словами, удается найти предикат $wp(S_{do}, T)$
B. Предикат $P \wedge \neg(i : 1 \leqslant i \leqslant n : B_j)$ истинен для всех состояний

2.8. Макроподстановки

Многие алгоритмы удобно строить разбивая исходную задачу на подзадачи, а их, в свою очередь на еще более маленькие подзадачи и так далее. Такой метод проектирования алгоритмов и программного обеспечения в целом носит название *метода пошагового уточнения*. Для того, чтобы использовать этот метод, удобным инструментом внутри языка являются макросы.

В рассматриваемом языке охраняемых команд введено понятие макро-функций – макросов с параметрами. Такие макросы могут быть объявлены и определены отдельно от программы, а затем встроены в место ее вызова. Макросы являются синтаксическими и работают с абстрактным синтаксисом, в отличие, например, от макросов языка Си. Вызов макроса с параметрами является отдельным оператором языка.

Например макрос, складывающий два числа x и y объявляется следующим образом:

```
add(x,y) :=
    z := x + y
```

Оператор вызова макроса:

```
x := 2; add(x, y)
```

2.9. Задание постусловий и инвариантов

Поскольку исследуется вывод предусловия исходя из заданного пользователем постусловие, необходима возможность его указывать либо вне программного текста, либо внутри него. Было решено встроить постусловие и инварианты цикла в сам программный текст. Таким образом, вместе с операторами программы, которые будут исполнены при интерпретации, содержится и спецификация алгоритма.

Постусловие объявляется как логическое выражение, содержащее переменные программы. В этом случае программа принимает вид:

```
S1; S2; ...; Sn {<постусловие>}
```

Каждый оператор цикла содержит свой инвариант, который также является логическим выражением:

```
{<инвариант>} do ... od
```

2.10. Выводы

Итак, описаны операторы языка охраняемых команд и определена их семантика путем сопоставления им некоторого преобразователя предикатов – функционального отношения на множестве предикатов. В приложении ??? приведено описание конкретного синтаксиса языка в расширенной форме Бэкуса-Наура и иллюстрированы синтаксичекие диаграммы Вирта.

ГЛАВА 3. РАЗРАБОТКА ИНТЕРПРЕТАТОРА И АНАЛИЗАТОРА

3.1. Генерация лексического и синтаксического анализатора

Для разработки интерпретатора языка охраняемых команд дейкстры, необходимо разработать:

A. лексический анализатор, разбивающий исходный набор символов программы на лексемы
B. синтаксический анализатор, разбивающий полученный набор лексем на синтаксические единицы и строящий дерево разбора программы
C. обход дерева разбора и выполнение необходимых семантических операций в его узлах

Существует два подхода к разработке лексического и синтаксического анализатора: создание их вручную, или автоматическая их генерация. При разработке интерпретатора языка охраняемых команд был выбран второй подход, а в качестве инструмента для генерации анализаторов был использован ANTLR4.

Генератор ANTLR4 использует грамматику языка, заданную в специальном формате, похожем на РБНФ. Грамматика ANTLR4 изложена в приложении П.

3.2. Интерпретация программы

Интерпретация программы выполняется путем обхода ее дерева разбора. Корневой узел этого дерева представляет из себя всю программу. Его дочерними узлами являются операторы, упорядоченные в порядке их нахождения в исходной программе.

Например, для программы $x := 2; y := 3; z := x + y$, дочерними узлами корневого узла являются три оператора присваивания. Посещения узлов операторов имеют специфичные для них семантические операции. Далее изложены все семантические операции, выполняемые для каждого типа оператора.

3.2.1. Оператор присваивания

Оператор присваивания содержит два важных дочерних узла: имя переменной, стоящее слева от знака присваивания, и выражение, стоящее справа. Для вычисления выражений используется стек выражений, на который кладутся результаты выполнения арифметических или логических операций, а снимаются с него перед выполнением арифметических или логических операций. Выражение справа от знака присваивания вычисляется, и его значение оказывается на вершине стека. Связь между именем переменной и ее значением сохраняется в отдельном словаре.

3.2.2. Условный оператор

Условный оператор содержит один или несколько узлов, соответствующих охраняемым командам. Узел охраняемой команды, в свою очередь, содержит выражение-предохранитель, и список операторов, охраняемых этим предохранителем.

При посещении узла условного оператора, поэтапно выполняются следующие шаги:

A. Вычисляются логические значения предохранителей всех охраняемых команд, содержащихся в условном операторе.
B. Если среди них не находится ни одного истинного значения, программа завершает свою работу, поскольку в этом случае условный оператор не завершается успешно.
C. Среди охраняемых команд, чей предохранитель истинен, случайным образом выбирается одна из них. Случайность выбора получается путем генерирования псевдослучайного целого числа.
D. Вызывается процедура посещения узла списка операторов, соответствующего выбранной охраняемой команде.

Таким образом, интерпретация недетерминированна с точностью до генерации псевдослучайного числа, однако в случаях, когда охраняемая команда всего одна, или их несколько и они попарно исключают друг друга, условный оператор остается детерминированным.

3.2.3. Оператор цикла

Оператор цикла не отличается по своей синтаксической структуре от условного оператора, за исключением лексем do и od, обрамляющих охраняемые команды. При его посещении, поэтапно выполняются следующие шаги:

A. Вычисляются логические значения предохранителей всех охраняемых команд, содержащихся в условном операторе.
B. Если среди них не находится ни одного истинного значения, посещение узла завершается.
C. Среди охраняемых команд, чей предохранитель истинен, случайным образом выбирается одна из них. Случайность выбора получается путем генерирования псевдослучайного целого числа.
D. Вызывается процедура посещения узла списка операторов, соответствующего выбранной охраняемой команде.

E. Выполняется переход к шагу 1.

3.2.4. Оператор вызова макроса

Узел оператор вызова макроса имеет следующие важные дочерние узлы: имя макроса и список аргументов, передаваемых макросу. При посещении этого узла:

A. Выполняется поиск списка операторов, соответствующего этому имени макроса. Такое соответствие сохраняется в словаре при объявлении макросов.
B. Создается копия этого списка операторов и заменяются все вхождения формальных параметров макроса, указанных в его объявлении, на фактически переданные аргументы.
C. Вызывается процедура посещения узла полученного списка операторов.

3.3. Вывод предусловия

Для вывода предусловия используется тот же лексический и синтаксический анализатор, что и для интерпретации программы. Однако семантические операции в узлах дерева разбора совершенно другие.

Вся программа состоит из списка операторов и заданного в конце программы желаемого постусловия. Это постусловие необходимо задавать именно для обратного вывода, но совершенно не обязательно для интерпретации. Если интерпретатор встретит постусловие в конце программы, он проверит истинность этого постусловия на полученном конечном состоянии программы.

Для обратного вывода необходимо посещать узлы операторов не в прямом порядке, а в обратном. Таким образом, постусловие "протаскивается"от конца программы к началу, видоизменяясь при встрече с каждым оператором. Далее описаны те семантические операции, которые выполняются при посещении каждого типа оператора.

3.3.1. Оператор присваивания

3.3.2. Условный оператор

3.3.3. Оператор цикла

3.3.4. Оператор вызова макроса

3.3.5. Упрощение предикатов

Изложенный способ трансформации предикатов, хоть и являяется удобным для машинной автоматизации, совершенно не пригоден для чтения результирующего предусловия человеком. Дело в том, что с каждым условным оператором или оператором цикла предикат достаточно сильно раздувается – это видно из формулы ???. Поэтому необходимо сокращать получающиеся предикаты, пользуясь правилами вывода исчисления предикатов.

3.4. Функции в предикатах

В постусловии программы, как и в инвариантах циклов, могут присутствовать вызовы функции с некоторым именем как одна из альтернатив выражения. Например, следующая программа содержит функцию add внутри постусловия:

```
z := x + y
{z == add(x, y)}
```

Очевидным образом будет выведено предусловие

$$x + y == add(x, y) \tag{3.1}$$

Это равенство должно быть истинным для любых переменных состояния x и y, однако поскольку заранее о свойствах функции add ничего не известно, сократить предусловие в $True$ не удается. Данный механизм является мощным инструментом при формальной проверке корректности алгоритма, поскольку таким образом может быть введена произвольная функция с неизвестными свойствами и указано постусловие, ее содержащие, а после вывода предусловия его необходимо рассмотреть, опираясь на известные свойства введенной функции. Общезначимость полученного предусловия не обязательна, как в примере ???, может быть получено некоторое ограничение на множество начальных состояний.

3.5. Выводы

Спроектирован и реализован язык охраняемых команд, содержащий основные конструкции структурного программирования и при этом не является строго детерминированным. Для него реализован вспомогательный инструмент, который позволяет в полуавтоматическом режиме строить семантику разработанной программы. Однако значительная часть разработки алгоритма остается на плечах его создателя, потому что параллельно с написанием кода алгоритма необходимо:

A. Указать постусловие, отражающее желаемый результат работы программы
B. Для каждого цикла программы вычислить его инвариант
C. Доказать, пользуясь свойствами введенных в предикаты функций, что из инвариантов циклов следует их постусловие, а также упростить полученное предусловие аналогичным образом.

ГЛАВА 4. ЭКСПЕРИМЕНТЫ С ЯЗЫКОМ ОХРАНЯЕМЫХ КОМАНД

Для эксперементального вывода корректных программ, а также обратного вывода семантики программы по ее исходному тексту, использованы примеры, приведенные Э. Дейкстрой в ??? и С.С. Лавровым в ???.

4.1. Факториал числа

Следующий алгоритм вычисляет факториал числа N:

```
n, f := N, 1;

{factorial(n) * f == factorial(N) & n >= 1}
do n > 1 -> f, n := f * n, n - 1 od

{f == factorial(N)}
```

Программа состоит из инициализации переменных n и f и цикла, умножающего f на числа от 2 до N. Инвариант цикла

$$P = (n! \cdot f == N! \wedge n \geqslant 1) \tag{4.1}$$

обеспечивает истинность постусловия по окончании этого цикла. Действительно, после окончания работы цикла состояние программы удовлетворяет предикату $P \wedge n <= 1$. Подставляя P, получаем

$$n! \cdot f == N! \wedge n \geqslant 1 \wedge n \leqslant 1 \vdash n! \cdot f == N! \wedge n == 1 \vdash f == N! \tag{4.2}$$

Выведенное предусловие:

$$N \geqslant 1 \tag{4.3}$$

Для функции $factorial$, введенной в постусловии, необходимо показать, что

$$n == 1 \wedge factorial(N) == f \cdot factorial(n) \vdash f == factorial(N) \tag{4.4}$$

Тогда денотационная семантика программы будет построена, и корректность программы будет формально доказана.

4.2. Максимум двух чисел

Следующий алгоритм вычисляет максимум из двух чисел a и b:

```
if a > b -> m := a
|  b >= a -> m := b
fi

{(m >= a) & (m >= b) & (m == a || m == b)}
```

4.3. Алгоритм Евклида

4.4. 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 СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

DOI Digital Object Identifier.

WoS Web of Science.

ВКР Выпускная квалификационная работа.

ТГ-объект Текстово-графический объект.

СЛОВАРЬ ТЕРМИНОВ

TeX — язык вёрстки текста и издательская система, разработанные Дональдом Кнутом.

LaTeX — язык вёрстки текста и издательская система, разработанные Лэсли Лампортом как надстройка над TeX.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Приложение 1

Краткие инструкции по настройке издательской системы LaTeX

В SPbPU-BCI-template *автоматически выставляются необходимые настройки и в исходном тексте шаблона приведены примеры оформления текстово-графических объектов*, поэтому авторам достаточно заполнить имеющийся шаблон текстом главы (статьи), не вдаваясь в детали оформления, описанные далее. Возможный «быстрый старт» оформления главы (статьи) под Windows следующий[П1.1]:

A. Установка полной версии MikTeX [**latex-miktex**]. В процессе установки лучше выставить параметр доустановки пакетов «на лету».
B. Установка TexStudio [**latex-texstudio**].
C. Запуск TexStudio и компиляция `my_chapter.tex` с помощью команды «Build&View» (например, с помощью двойной зелёной стрелки в верхней панели). *Иногда, для достижения нужного результата необходимо несколько раз скомпилировать документ.*
D. В случае, если не отобразилась библиография, можно
   - воспользоваться командой Tools → Commands → Biber, затем запустив Build&View;
   - настроить автоматическое включение библиографии в настройках Options → Configure TexStudio → Build → Build&View (оставить по умолчанию, если сборка происходит слишком долго): `txs:///pdflatex | txs:///biber | txs:///pdflatex | txs:///pdflatex | txs:///view-pdf`.

В случае возникновения ошибок, попробуйте скомпилировать документ до последних действий или внимательно ознакомьтесь с описанием проблемы в log-файле. Бывает полезным переход (по подсказке TexStudio) в нужную строку в pdf-файле или запрос с текстом ошибке в поисковиках. Наиболее вероятной проблемой при первой компиляции может быть отсутствие какого-либо установленного пакета LaTeX.

В случае корректной работы настройки «установка на лету» все дополнительные пакеты будут скачиваться и устанавливаться в автоматическом режиме. Если доустановка пакетов осуществляется медленно (несколько пакетов за один запуск

[П1.1] Внимание! Пример оформления подстрочной ссылки (сноски).

компилятора), то можно попробовать установить их в ручном режиме следующим образом:

1. Запустите программу: меню → все программы → MikTeX → Maintenance (Admin) → MiKTeX Package Manager (Admin).
2. Пользуясь поиском, убедитесь, что нужный пакет присутствует, но не установлен (если пакет отсутствует воспользуйтесь сначала MiKTeX Update (Admin)).
3. Выделив строку с пакетом (возможно выбрать несколько или вообще все неустановленные пакеты), выполните установку Tools → Install или с помощью контекстного меню.
4. После завершения установки запустите программу MiKTeX Settings (Admin).
5. Обновите базу данных имен файлов Refresh FNDB.

Для проверки текста статьи на русском языке полезно также воспользоваться настройками Options → Configure TexStudio → Language Checking → Default Language. Если русский язык «ru_RU» не будет доступен в меню выбора, то необходимо вначале выполнить Import Dictionary, скачав из интернета любой русскоязычный словарь.

Далее приведены формулы (П1.2), (П1.1), рис.П1.2, рис.П1.1, табл.П1.2, табл.П1.1.

$$\pi \approx 3{,}141. \tag{П1.1}$$

Рис.П1.1. Вид на гидробашню СПбПУ [**spbpu-gallery**]

Таблица П1.1

Представление данных для сквозного примера по ВКР [**Peskov2004**]

G	m_1	m_2	m_3	m_4	K
g_1	0	1	1	0	1
g_2	1	2	0	1	1
g_3	0	1	0	1	1
g_4	1	2	1	0	2
g_5	1	1	0	1	2
g_6	1	1	1	2	2

П1.1. Параграф приложения

П1.1.1. Название подпараграфа

Название подпараграфа оформляется с помощью команды `\subsection{...}`. Использование подподпараграфов в основной части крайне не рекомендуется.

П1.1.1.1. Название подподпараграфа

$$\pi \approx 3{,}141. \tag{П1.2}$$

Рис.П1.2. Вид на гидробашню СПбПУ [**spbpu-gallery**]

Таблица П1.2

Представление данных для сквозного примера по ВКР [**Peskov2004**]

G	m_1	m_2	m_3	m_4	K
g_1	0	1	1	0	1
g_2	1	2	0	1	1
g_3	0	1	0	1	1
g_4	1	2	1	0	2
g_5	1	1	0	1	2
g_6	1	1	1	2	2

Приложение 2

Некоторые дополнительные примеры

В приложении[П2.1] приведены формулы (П2.2), (П2.1), рис.П2.2, рис.П2.1, табл.П2.2, табл.П2.1

$$\pi \approx 3{,}141. \qquad \text{(П2.1)}$$

Рис.П2.1. Вид на гидробашню СПбПУ [**spbpu-gallery**]

Таблица П2.1

Представление данных для сквозного примера по ВКР [**Peskov2004**]

G	m_1	m_2	m_3	m_4	K
g_1	0	1	1	0	1
g_2	1	2	0	1	1
g_3	0	1	0	1	1
g_4	1	2	1	0	2
g_5	1	1	0	1	2
g_6	1	1	1	2	2

[П2.1] Внимание! Пример оформления подстрочной ссылки (сноски).

П2.1. Подраздел приложения

$$\pi \approx 3{,}141. \qquad \text{(П2.2)}$$

Рис.П2.2. Вид на гидробашню СПбПУ [**spbpu-gallery**]

Таблица П2.2

Представление данных для сквозного примера по ВКР [**Peskov2004**]

G	m_1	m_2	m_3	m_4	K
g_1	0	1	1	0	1
g_2	1	2	0	1	1
g_3	0	1	0	1	1
g_4	1	2	1	0	2
g_5	1	1	0	1	2
g_6	1	1	1	2	2