МИНОБРНАКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Хакасский государственный университет им. Н. Ф. Катанова**»**

Колледж педагогического образования, информатики и права

ПЦК естественнонаучных дисциплин, математики и информатики

РЕФЕРАТ

на тему:

Теория и практика языков программирования

Автор реферата: _____ **К.А. Лубошникова**
(подпись) (инициалы, фамилия)

Специальность: 230115 - Программирование в компьютерных системах

Курс: II

Группа: И-21

Зачет/незачет: _____

Руководитель: _____ **О.П. Когумбаева**
(подпись, дата) (инициалы, фамилия)

г. Абакан, 2018г.

Содержание:

Введение

Интерес к формальным описаниям семантики языков программирования обусловлен несколькими причинами:

Во-первых, программисты должны точно знать, что именно делают операторы языка, хотя бы на уровне соответствия входа и выхода (результата действия) каждой конструкции.

Во-вторых, создатели компиляторов также должны совершенно одинаково понимать язык, для которого разрабатывается компилятор. К сожалению, чаще всего, и те, и другие руководствуются неточными и неполными англоязычными объяснениями смысла конструкций языка. Порой это приводит к получению компиляторов, дающих на одной и той же программе различные результаты, т.е. по сути, компилируются разные языки, «похожие на оригинал», представляющий с точки зрения семантики «неопознанный объект».

В-третьих, если формальное описание семантики допускает эффективную реализацию (вычислимую интерпретацию), то оно может быть воспринято системой, автоматизирующей получение компиляторов, т.е. компилятор (или интерпретатор) может быть получен непосредственно из формальной спецификации языка. Эта идея тесно связана с «мечтой» (в некоторых случаях, сбывшейся) об автоматизации программирования, когда программа может быть получена автоматически по формальному описанию задачи. Такой подход позволяет создавать компиляторы, правильность которых следует из математического представления смысла конструкций и их преобразований или может быть доказана математически. Следует отметить, что именно эта цель – получение компиляторов по формальным, «семантически-ориентированным» спецификациям языков программирования, дала толчок к разработке различных методов описания семантики, особенно интенсивно развивавшихся с 80-х годов прошлого столетия.

В-четвертых, процесс формализации языка позволяет выявить в нем скрытые противоречия и прямые ошибки, проанализировать его конструкции для устранения избыточности и неполноты для представления задач той предметной области, для которой язык предназначался.

В-пятых, формальное представление семантики позволит дать методы преобразований и анализа программ для различных целей. Без формального определения смысла языковых конструкций трудно представить работу таких «отраслей» в теории и практике программирования как верификация, синтез, оптимизация программ, изучение (и обеспечение) их трансформаций при переходе к различным новым вычислительным платформам. Формальные (или формализованные) описания языков называют часто спецификацией языка (или отдельно, спецификацией синтаксиса, статической, динамической семантики, спецификацией перевода и т.д.).

1. Классификация языков программирования

1.1. Языки программирования - искусственные языки. Они отличаются от естественных человеческих языков малым количеством слов, значение которых понятно транслятору (эти слова называются ключевыми), и довольно жесткими требованиями по форме записи операторов (совокупность этих требований образует грамматику и синтаксис языка программирования). Нарушения формы записи приводят к тому, что транслятор не может правильно выполнить перевод и выдает сообщение об ошибке.

1.2.Уровни языков программирования:

1. Языки низкого уровня

2. Языки высокого уровня

Язык программирования низкого уровня - это язык программирования, созданный для использования со специальным типом процессора и учитывающий его особенности. Он близок к машинному коду и позволяет непосредственно реализовать некоторые команды процессора. Они мало похожи на привычный человеку язык. Большие программы на таких языках пишутся редко. Но программы работают быстро, занимая маленький объем и допуская минимальное количество ошибок. Чем ниже и ближе к машинному уровень языка, тем меньше и конкретнее задачи, которые ставятся перед каждой командой.
Для каждого типа процессоров самым низким уровнем является язык ассемблера, который позволяет представить машинный код не в виде чисел, а в виде условных обозначений, называемых мнемониками. У каждого типа процессора свой язык ассемблера; его можно рассматривать одновременно и как особую форму записи машинных команд, и как язык программирования самого низкого уровня.
Достоинством языков низкого уровня является то, что с их помощью создают самые эффективные программы (краткие и быстрые). Недостаток таких языков в том, что их трудно изучать из-за необходимости понимать устройство процессора и в том, что программа, созданная на таком язык, неприменима для процессоров других типов.

Языки программирования высокого уровня заметно проще в изучении и применении. Программы, написанные с их помощью, можно использовать на любой компьютерной платформе при условии, что для нее существует транслятор данного языка. Эти языки вообще никак не учитывают свойства конкретного процессора и не предоставляют прямых средств для обращения к нему. В некоторых случаях это ограничивает возможности программистов, но зато и оставляет меньше возможностей для совершения ошибок.

2. Атрибутные грамматики

Как специальный формализм для определения семантики атрибутные грамматики (АГ) были предложены Кнутом [73] в 1968 году и с тех пор получили большое теоретическое развитие и практическое приложение. В атрибутных грамматиках семантическая информация и ее вычисление локализуются в рамках продукций КС-грамматики. Такой подход удобен при реализации синтаксически управляемого перевода, привязанного к дереву синтаксического разбора транслируемой программы. Поэтому АГ широко используются в системах построения трансляторов как средство спецификации статической семантики и, возможно, перевода.

Атрибутной грамматикой называется четверка AG = (G, S, I, R), где

• G = (N, T, Z, P) - КС-грамматика;

• S- конечное множество синтезируемых атрибутов;

• I- конечное множество наследуемых атрибутов, $I \cap S = ;$

• R - конечное множество семантических правил.

Условия на значения (предикативные функции) имеют вид булевских выражений, заданных на множестве атрибутов A(X) правила грамматики. Единственными выводами, разрешенными в атрибутной грамматике, являются те, в которых все условия истинны. Ложное значение говорит о нарушении правил статической семантики языка. Например, для синтаксического правила оператора присваивания:

$$\text{присваивание} \rightarrow \text{переменная} := \text{выражение}$$

правило вычисления атрибута val (значение) для символа переменная:

$$\text{val}(\text{переменная}) = \text{val}(\text{выражение})$$

задает значение переменной как функцию от значения выражения в правой части оператора присваивания, предикат для атрибута type:

$$\text{type}(\text{переменная}) = \text{type}(\text{выражение})$$

выражает условие: «типы левой и правой частей оператора присваивания должны быть эквивалентны». Чтобы отличать отношение равенства от присваивания значения, в первом случае вместо знака «=» часто употребляют символ «←» передачи значения:

$$\text{val}(\text{переменная}) \leftarrow \text{val}(\text{выражение}).$$

2.1. Примеры атрибутных грамматик

Пример 1. Вычисление семантических значений.

Рассмотрим атрибутную грамматику AG= (G, S, I, R) двоичной системы записи чисел. Соответствующая контекстно-свободная грамматика G задает правильные двоичные числа, а семантика всякой двоичной записи определяется ее значением в десятичной системе счисления, вычисленным по двоичному разложению. Так, значением семантической функции для двоичного числа 1101.01 будет десятичное число 13.25.

Знак равенства в семантических правилах фактически означает присваивание: нужно вычислить значение выражения правой части равенства и присвоить это значение атрибуту левой части.

Семантические правила, по сути, реализуют алгоритм получения десятичного числа из двоичного, используя его разложение в полином по степеням двойки.

3. Денотационная семантика

Денотационная семантика – математически наиболее строгий широко известный и развиваемый метод определения функциональной семантики программ. Программа рассматривается как функциональный терм, подход предоставляет способ придания этому терму значения, которое и составляет смысл программы. Теоретической основой этого метода послужили работы Д. Скотта о непрерывных решетках, в которых предложен подход к математическому представлению «смысла» рекурсивных программ и рекурсивных определений типов.

Проблема и ранние попытки ее решения обозначены в работах Маккарти, отметившего, что для математической теории вычислений мало что дают как методы вычислений, так и традиционная теория рекурсивных функций. Причина - отсутствие единого понятия вычислимости над действительными числами; традиционная теория рекурсии, по его мнению, неоправданно «привязана» к натуральным числам и концентрируется на доказательствах неразрешимости.

В качестве базиса программирования необходимо понятие вычислимости над самыми различными типами данных. Введенный Маккарти базис исходных функций и предикатов позволил выразить действие программы через рекурсивные функции над этим базисом, однако определение семантики самих рекурсивных схем было операционным. Этот формализм был использован как основа для языка Лисп.

3.1 Обоснование подхода

Изначально построения Скотта были ориентированы на определение семантики лямбда-исчисления (λ-исчисления), в результате была получена модель для этого функционального языка. λ-исчисление часто используется для формального представления языков программирования и теоретических выкладок по поводу их функциональной семантики. Рассмотрим некоторые ключевые понятия из этой области. Лямбда-исчисление – это бестиповая теория, рассматривающая функции как правила, а не как графики (множества пар аргумент – значение). Понятие функции как закона или правила подразумевает переход от аргумента к значению как процесс, закодированный некоторым определением. Такое рассмотрение функций подчеркивает их вычислительные аспекты.

Функции как правила рассматриваются в полной общности: можно считать, что функции заданы определениями на, например, русском языке и применяются

к аргументам, также описанным по-русски. Можно рассматривать функции, заданные программами для машин и применяемые к другим таким программам.

В обоих случаях мы имеем бестиповую структуру, где объекты изучения являются одновременно и функциями, и аргументами. Это – отправная точка бестипового λисчисления. В частности, функция может применяться к самой себе. При обычном понимании функции в математике это невозможно.

3.2. Элементы теории Скотта: Скотт, принимая во внимание неполноту λ-исчисления (не все истинные функциональные равенства можно получить, используя редукционный вывод), утверждал, что можно определить смысл любого λвыражения, вне зависимости от того, имеет ли оно нормальную форму (т.е. редуцируется к некоторой окончательной форме за конечное число применений редукций), придать одинаковый смысл выражениям, не сводимым один к другому редукцией.

Потребность в денотационной семантике, которая выражала бы функциональный смысл программ, проявила проблему, которая в чистом виде возникла и в λ-исчислении. Бестиповый характер этой теории делал неясным вопрос о построении моделей для нее. Желательно было иметь множество X, в которое вкладывалось бы его функциональное пространство $X \to X$; этого, однако, невозможно достичь по мощностным соображениям.

Эту трудность преодолел Скотт (в 1969 г.), который построил модели λ-исчисления, «урезав» $X \to X$ до множества непрерывных (в надлежащей топологии) функций на X. Только после этого стало понятно, как строить денотационную семантику языков программирования. Это произошло благодаря тому, что методом Скотта можно также придать смысл двум важным конструкциям языков программирования – рекурсии (наименьшие неподвижные точки) и типам данных (целые числа, списки, массивы и т.д.). Главная цель разработки – получение удобной теории функций.

3.3.Оценки денотационного подхода

Денотационный подход позволяет глубоко понять математическую структуру той области объектов, на которую ориентирован алгоритмический язык. Семантика Скотта ориентирована на языки типа λисчисления, поэтому для изучения свойств языка, доказательств правильности программ и их преобразований семантика наименьшей неподвижной точки оказывается адекватной. Для многих других задач более естественным может быть использование некоей канонической неподвижной точки.

4. Логический подход к определению статической семантики

4.1. Статическая семантика

Статическая семантика языка программирования L выражает контекстно-зависимые свойства его конструкций и представляется при трансляции совокупностью всех необходимых для контекстного анализа и перевода (или интерпретации) программ $\ell \in L$ отношений и атрибутов, которые могут быть вычислены до выполнения программ. Соответственно обычной методике трансляции эта совокупность разделяется на части, отвечающие различным этапам обработки ℓ: построению синтаксической (или абстрактной синтаксической), контекстной и семантической структур программы. К (абстрактной) синтаксической структуре программы ℓ относятся отношения и атрибуты, характеризующие лексику и контекстно-свободное строение ℓ и содержащие минимальный объем информации, достаточный для проведения контекстного анализа и преобразования ℓ. В качестве такой структуры выступает обычно неполное, частично атрибутированное (лексическими атрибутами) дерево разбора ℓ или его образ в виде графа, дуги которого представляют транзитивные замыкания отношений подчинения и соседства на дереве вывода. В контекстную структуру включаются отношения и атрибуты, определяемые на основе контекстной зависимости объектов программы и фиксирующие информационную связь конструкций программы. Они необходимы для последующего контроля контекстных условий и синтеза семантического терма, используемого для определения динамической семантики, перевода или преобразования программы. В рассматриваемом подходе абстрактная синтаксическая структура, контекстные связи и контекстные условия задаются логическими средствами: множеством аксиом (формул, все переменные которых вязаны кванторами) в исчислении предикатов первого порядка с равенством.

4.2. Обоснование подхода

Использование подхода предполагает выполнение двух этапов. На первом этапе составляется описание исследуемого объекта в предлагаемом логическом языке. В описание включаются: перечень элементов, их атрибутов, отношений между ними, правила, индуктивно определяющие эти отношения и атрибуты (зависимости между элементами) и правила, задающие ограничения на зависимости – условия «правильности» функционирования, аварийные ситуации и т.п.

В целом, правила должны полностью определять требуемые свойства объекта и/или его поведение. Если необходима поддержка корректности трансформаций исследуемой системы, то аналогичным образом могут быть описаны и трансформационные преобразования.

На втором этапе подключаются интерпретаторы спецификаций, цель которых – построение логической модели объекта и проверка выполнимости требуемых свойств. В процессе интерпретации диагностируются кроме того логические ошибки - противоречивость и неполнота представления объекта или проекта системы.

На практике более адекватен многосортный логический язык, но для простоты изложения далее используется односортная сигнатура. Формально, $\Sigma = (R, F, C)$, где R - множество имен предикатов (отношений), F - функций (атрибутов) и C - констант, причем C не пусто и все константы различны. Отношение равенства считается встроенным (аксиомы равенства автоматически добавляются к T D). Константы кодируют элементы объекта (например, конструкции программ или технические узлы, устройства, модули и т.п.).

Свойство индуктивной вычислимости фактически обеспечивает существование фундированного порядка на модели, при котором термы условия определения могут быть вычислены прежде определяемых значений и, таким образом, интерпретирующее отображение i может быть индуктивно вычислено по системе определений.

4.3. Языковые средства

Описание - спецификацию S = (Σ, T) исследуемого объекта составляют сигнатура Σ и теория T = TD ∪ TR - множество аксиом, определяющих объект (TD) и ограничивающих его состояние или поведение (TR). На практике более адекватен многосортный логический язык, но для простоты изложения далее используется односортная сигнатура.

Формально, Σ = (R, F, C), где R - множество имен предикатов (отношений), F - функций (атрибутов) и C - констант, причем C не пусто и все константы различны. Отношение равенства считается встроенным (аксиомы равенства автоматически добавляются к T D). Константы кодируют элементы объекта (например, конструкции программ или технические узлы, устройства, модули и т.п.).

4.4. Обозначения

Σt - множество замкнутых термов сигнатуры Σ, VФ - множество переменных формулы Ф; для произвольного множества A: A* - совокупность конечных списков над A, - элемент A, a a - вектор таких элементов; для всякой алгебраической системы M = (A, i) A обозначает носитель, i - интерпретирующее отображение на A.

4.5. Диагностика ошибок.

Ошибки, обнаруживаемые при интерпретации, делятся на три класса: 1) неопределенные функции, 2) переопределение функций или отношений (противоречие), 3) невыполнимость ограничений. Первые два относятся к корректности спецификации (Σ,TD), третий - к неверным поведению или структуре моделируемого комплекса. Вообще говоря, такое разделение условно, т. к. определения атрибутов и отношений в аксиомах TD сами по себе являются некоторым ограничением связей и взаимодействий.

Это дает возможность вовремя исправить ошибки проекта. Если же моделируется уже готовый, работающий объект, то получение ошибки говорит о том, что заданное на вход множество фактов T 0 либо недостаточно для воссоздания полной картины моделируемой ситуации, либо противоречиво, либо ведет к аварии. Для контекстного анализа это означает, что ошибки в программе возникают уже на уровне локального контекста.

5. Трансформации и модификации программ

Преобразования моделируемого объекта сводятся к изменениям его логической модели (и, соответственно, спецификаций) которые можно представить комбинацией следующих базовых трансформаций.

5.1. Доопределение функций и отношений на исходном множестве (задачи анализа).

Пусть t 1, … , tn – последовательность трансформаций, реализующая преобразование объекта (программы, проекта, системы) из состояния **Ω**1 в состояние **Ω**n. В качестве логического представления трансформации ti рассматривается спецификация Si = (Σi,Ti), а сам процесс трансформации состоит в интерпретации S i – получении минимальной модели M i=(Ci,Ii), представляющей семантику S i. Условием корректности

156 всякой трансформации t i тогда является условие правильности спецификации Si : У1). Трансформация t i корректна тогда и только тогда, когда T i имеет модель M i. Это условие эквивалентно непротиворечивости (относительно равенства и использования отрицаний) и F-полноте спецификации Si. Условие У1 не ограничивает поведение последовательности трансформаций. Корректность всего цикла преобразований может потребовать выполнения дополнительных ограничений. Например, если для всякого i Σi = Σi+1, то каждая Ii+1 получена доопределением I i. Тогда имеем: У2) Последовательность t 1, … ,tn трансформаций корректна, если корректна каждая t i, и M 1 ⩽ M2⩽ … ⩽ Mn , где ⩽ - отношение гомоморфного вложения. Условие У2 является достаточно слабым для последовательности трансформаций. В частности, оно не требует корректности совокупной спецификации S=(Σ,T1 ∪ T2 ∪ … ∪ Tn). Поскольку допускаются формулы с отрицанием, и вывод с фактами, полученными на шаге i не повторяется для спецификаций S j, j<i, то минимальная модель M для S, хотя и может существовать, но не обязательно является объединением моделей Mi.

5.2. Добавление новых элементов, отношений и атрибутов (пошаговое уточнение, задачи проектирования и синтеза).

Если при преобразованиях t i → t i+1 сигнатура расширяется, т.е. Σi⊂Σi+1, то корректность последовательности трансформаций поддерживает следующее условие: У3) последовательность трансформаций t i → ti+1 корректна, если корректна каждая из них и модель M i является подсистемой Mi+1 в сигнатуре Σi ∪Σi+1 , причем Ii+1↾((Ri ∪Fi ∪Ci)×Ci*) = Ii. Символ ↾ обозначает операцию ограничения функции.

5.3. Сужение исходной логической структуры (удаление элементов, задачи оптимизации).

Это немонотонное в общем случае преобразование, тем не менее, в ряде задач может быть представлено в рамках рассматриваемого класса спецификаций.

6. Аксиоматическая семантика

Этот подход создан в процессе разработки способа доказательства правильности программ, основан на математической логике и развит в работах Флойда и Хоара. Логические выражения, называемые утверждениями или предикатами, используются в качестве пред- и постусловий выполнения каждого оператора программы. Оператор σ программы рассматривается как процедура, изменяющая значения каких-то переменных.

6.1. Использование и проблемы подхода

Аксиоматический подход развивается как средство доказательства и проверки свойств программ и в качестве входного описания для систем построения трансляторов аксиоматическая спецификация не используется.

Непротиворечивость аксиоматических определений может быть доказана посредством их интерпретации на областях математической модели, с использованием функций денотационной семантики. Такой подход рассмотрен для подмножества Паскаля. Для доказательства (довольно громоздкого) использованы структурная индукция, индукция по неподвижной точке, выполнена дополнительная более строгая аксиоматизация подстановки. Наиболее сложными являются доказательства непротиворечивости объявления и вызова процедур.

Статическая семантика не аксиоматизируется, соответствие типов параметров процедур оговаривается как неформализованное дополнение (комментарий) к правилам вывода. В формуле предлагается для отражения контекстных условий, связанных с типами переменных, ввести (формализовать) понятие «статической среды» с областями-типами и семантическими функциями проверки типов. В результате получена «гора» формул и уравнений, в которой определяемые контекстные условия окончательно затерялись.

Как уже упоминалось, построение правил вывода для операторов программ является сложной задачей, в связи с чем описание может содержать ошибки принципиального характера.

В работе Кларка рассмотрены возможности, характерные для алголоподобных языков программирования:

1. Процедуры как параметры процедур.
2. Статическая область действия идентификаторов, входящих в описание процедур.
3. Рекурсия.
4. Глобальные переменные.
5. Процедуры, определенные внутри программы и используемые в выражениях для фактических параметров процедур, вызываемых по наименованию.

Доказано, что совмещение всех этих пяти возможностей в одном языке делает невозможным построение полной хоаровской семантики. Устранение любой из этих пяти возможностей может дать полную хоаровскую систему. Известно, что функционально эквивалентные программы далеко не всегда доказуемо эквивалентны.

Например, можно построить тривиально зацикливающуюся программу и программу, зацикливающуюся тогда и только тогда, когда верна какая-либо истинная, но не доказуемая в теории множеств формула. Флойд-Хоаровская система правил полностью определяет выполнение программы только в том случае, если к ней добавляется бесконечное правило вывода (правило бесконечной индукции для циклов).

Если такое правило не предусмотрено, то возможно «более чем бесконечное выполнение программы, когда моделью множества моментов времени служит нестандартная модель натурального ряда. Вопросы полноты для алгоритмических логик очень тонки и трудны с математической точки зрения. Кларк доказал необходимость использования дополнительных понятий в аксиоматизациях хоаровского типа. Эта необходимость учтена введением переменных-призраков, отсутствующих в программе, но необходимых для ее верификации.

Более того, показано, что ни один однозначный преобразователь предикатов не допускает корректных преобразований без введения дополнительных переменных.

С другой стороны, это показывает необходимость чисто логических правил преобразования высказываний в Флойд-Хоаровской системе. Кларк связал аксиоматическую семантику со скоттовскими интерпретациями и показал, что посылки правил вывода полной системы обязаны быть наибольшими неподвижными точками преобразователей предикатов.

Библиографический список

1. Орлов, С. А. Теория и практика языков программирования/ С. А. Орлов. - СПб. [и др.] : Питер, 2014. - 688 с

2. Серебряков, В.А. Теория и реализация языков программирования/ В. А. Серебряков [и др.]. - 2-е изд., испр. и доп. - М. : МЗ- ПРЕСС, 2006. - 352 с.

3. Лавров, С. С. Универсальный язык программирования] / С. С. Лавров. - 3-е изд., испр. - М. : Наука. Гл. ред. физ.-мат. лит., 1972. - 184 с. : ил. - Б. ц.. [электронный источник] http://library.khsu.ru (дата обращения: 14.01.18)

4. Гилев, Сергей Евгеньевич. Теория языков программирования и проектирование компиляторов : утв. ред.-изд. советом СибГТУ : "Программное обеспечение вычислительной техники и автоматизированных систем / С. Е. Гилев, Д. Н. Кузьмин, Красноярск : СибГТУ, 2010. - 108 с. [электронный источник] http://library.khsu.ru (дата обращения: 14.01.18).

5. Гутер, Р. С. Практика программирования / Р. С. Гутер. - М. : Наука , 1965. - 211 с.. [электронный источник] http://library.khsu.ru (дата обращения: 14.01.18)

6. Сухарев, Михаил. Теория и практика программирования (Turbo Pascal)/ Михаил Сухарев ; под ред. М. В. Финкова. - СПб. : Наука и техника, 2003. - 576 с. [электронный источник] http://library.khsu.ru (дата обращения: 14.01.18).

7. Кетков, Юлий Лазаревич. Практика программирования: Visual Basic, C++ Builder, Delphi / Ю. Кетков, А. Кетков. - СПб. : БХВ- Петербург, 2002. - 449 с. [электронный источник] http://library.khsu.ru (дата обращения: 14.01.18).

8. Светозарова, Г. И. Практикум по программированию на алгоритмических языках / Г. И. Светозарова, Е. В. Сигитова, А. В. Козловский ; под ред. С. В. Емельянова. - М. : Наука. Гл. ред. физ.-мат. лит., 1980. - 320 с. : ил. - Б. ц.

9. Васюкова, Нина Дмитриевна. Практикум по основам программирования. Язык ПАСКАЛЬ / Н.Д. Васюкова, В.В. Тюляева. - М. : Высшая школа, 1991. - 160 с. (дата обращения: 14.01.18). [электронный источник] http://library.khsu.ru

10. Кнут, Дональд Э. Искусство программирования 2002 - . - (Классический труд). - 832 с.