Опорные конспекты лекций Ф СО ПГУ 7.18.2/06

Министерство образования и науки Республики Казахстан

Павлодарский государственный университет
им. С. Торайгырова

Кафедра _Информатики и информационных систем_

ОПОРНЫЕ КОНСПЕКТЫ ЛЕКЦИЙ

Дисциплины «Логическая культура программиста»

для студентов специальности 5В070300 – Информационные системы

Павлодар

Лист утверждения к опорному конспекту

Форма
Ф СО ПГУ 7.18.1/05

УТВЕРЖДАЮ
Декан ФФМиИТ

_____Н.А.Испулов
«__»_____2012 г.

Составители: доцент, к.п.н. Джарасова Г.С.,
преподаватель Рахимбаева Б.А.

Кафедра Информатика и информационные системы

ОПОРНЫЙ КОНСПЕКТ ЛЕКЦИЙ

по дисциплине «Логическая культура программиста»
для студентов специальности 050703 – Информационные системы

Рекомендована на заседании кафедры от «15»___11___2012 г.
Протокол № 4.

Заведующий кафедрой _____ Н.Н.Оспанова
(подпись)

Одобрена методическим советом факультета ФМиИТ
«16»____11____2012 г. Протокол №_4_

Председатель МС_____ А.Б.Искакова
(подпись)

Кегль 14, буквы строчные, кроме первой прописной

Тема 1 Введение

В настоящее время перед всеми участниками образовательного процесса стоит проблема повышения качества образования, его адаптации к складывающимся жизненным реалиям (экономическим, социальным, культурным, демографическим и т.д.). Увеличение информационных потоков, сопряженное с умениями и навыками оперативного анализа информации, развитие наукоемких технологий и производств, возросшая конкурентность в условиях рыночной экономики – все это обусловливает и предъявляет довольно высокие требования к уровню подготовки будущих IT-кадров. Приведение обучения в соответствие с требованиями современного общества возможно лишь при условии подготовки будущих IT-кадров к умению адаптироваться к любым меняющимся условиям профессиональной деятельности.

Основные задачи профессиональной подготовки бакалавров информатики: – обеспечение функционирования объектов профессиональной деятельности, участие в совершенствовании и разработке экономически выгодных, эргономичных компьютерных технологий, автоматизация процессов в различных отраслях. В процессе изучения вопроса подготовки компетентных бакалавров информатики в [1] была предложена четырехуровневая модель обучения программированию в высшей школе (Таблица 1):

Таблица 1 – Четырехуровневая модель обучения программированию в высшей школе

Уровень 1	Метауровень	Получение теории	Верхние уровни абстракции
Уровень 2	Целевой	Обучение доказательству	
Уровень 3	Содержательный	Доказательство правильности программ	Нижние уровни абстракции
Уровень 4	Процессуальный	Индукция/дедукция	

Понятие «компетентность» (лат competentia, от competo - совместно добиваюсь, достигаю, соответствую, подхожу) в словарях определяется как «обладание знаниями, позволяющими судить о чём-либо» [27], «осведомлённость, правомочность», «авторитетность, полноправность» [28]. Д.А.Власов [29] проводя дефиниционный анализ, определяет компетенцию как производное понятие от компетентности, т.е. как понятие, обозначающее сферу приложения знаний, умений и навыков человека, а компетентность – как семантически первичную категорию, представляющую их интериоризированную (присвоенную в личностный опыт) совокупность, систему, некий знаниевый «багаж» человека.

Выше приведенная модель позволяет определить норму компетентности IT-кадров, которая соответствует также экономической модели подготовки специалистов согласно нормам ЮНЕСКО по компетентности учителей в использовании ИКТ [30]: техническая модель углубление знаний создание знаний (Таблица 2).

Таблица 2 – Сопоставление четырехуровневой и экономической моделей обучения

Четырехуровневая модель обучения программированию в вузе		Экономическая модель подготовки специалистов в вузе
Уровень 1	Метауровень	Создание знаний
Уровень 2	Целевой	
Уровень 3	Содержательный	Углубление знаний
Уровень 4	Процессуальный	Техническая грамотность

Компетентность представляется в ряде педагогико-психологических исследованиях [31, 32, 33] как совокупность трёх аспектов:

- смыслового, включающего адекватность осмысления ситуации в общем культурном контексте, т.е. в контексте имеющихся культурных образцов понимания, отношения, оценки;

- проблемно-практического, обеспечивающего адекватность распознавания ситуации, адекватную постановку и эффективное выполнение целей, задач и норм в данной обстановке;

- коммуникативного, фокусирующего внимание на адекватном общении в ситуациях культурного контекста и по поводу таких ситуаций с учётом соответствующих культурных образцов общения и взаимодействия.

Человек имеет общекультурную компетентность, если он компетентен (в трёх указанных выше аспектах) в ситуациях, выходящих за пределы его профессиональной сферы. В профессиональной компетентности главная роль отводится проблемно-практическому аспекту, а в общекультурной -смысловому и коммуникативному. В целом важны все три аспекта, так как профессиональное образование направлено на подготовку общекультурной и компетентной во многих областях личности [31]. Профессиональная компетентность является производным компонентом общекультурной компетентности любого человека [29].

В работе Власова Д.М. [29] в качестве основных сфер профессиональной компетентности определены следующие: *потребностно–мотивационная* (совокупность ценностных ориентаций, социальных установок, потребностей, интересов, составляющих основу мотивов), *операционно–техническая* (совокупность общих и специальных знаний, умений и навыков, профессионально важных качеств), *самосознания* (оценка человеком своего знания, поведения, нравственного облика и интересов, идеалов и методов поведения, целостной оценки самого себя как чувствующего и мыслящего существа и деятеля).

В рамках и с позиций вышеприведенных подходов к определению компетентности построим модель подготовки будущих информатиков. Как было сказано выше, документом, определяющим в современных условиях профессиональные требования к подготовке выпускников вузов по различным специальностям, является Государственный общеобязательный стандарт образования. При определении требований к уровню образованности выпускников специальности 050602-Информатика, квалифицированы следующие компетенции: социально-этическая, экономическая и организационно-управленческая, профессиональная [5]. К профессиональной компетенции будущих информатиков предъявляются следующие требования:

- овладение теоретическими знаниями и практическими навыками, соответствующими основной образовательной программе подготовки;

- наличие опыта работы на различных типах компьютерных систем, умение применять алгоритмические языки, использовать приближенные методы и стандартное программное обеспечение, пакеты прикладных программ и баз данных, средства машинной графики, экспертных систем и баз знаний для решения прикладных задач и задач теоретического характера;

- знание современных средств вычислительной техники, телекоммуникаций и средств связи;

- знание перспектив и тенденций развития информационных технологий;

- способность к совершенствованию своей профессиональной деятельности в области информатики;

- изучение специальной литературы и другой научно-технической информации, достижений отечественной и зарубежной науки и техники в области своей профессиональной деятельности.

Исходя из требований к профессиональной компетенции и на основе материалов Computing Curricula (http://se.math.spbu.ru/main.html, http://www.computer.org/) определим конкретные качества, которыми должны обладать будущие информатики. Эти качества включают в себя:

Системный взгляд на дисциплину. Цели обучения, связанные с конкретными модулями знаний, имеют тенденцию фокусироваться на отдельных концепциях и темах, что впоследствии может привести к фрагментарному усвоению дисциплины. Обучающиеся должны развить в себе высокоуровневое понимание систем в целом. Это восприятие должно преодолевать детали отдельных реализаций

различных компонент и давать общее понимание структуры компьютерных систем и процессов их создания и анализа.

 Понимание связи теории и практики. Фундаментальный аспект информатики - это равновесие между теорией и практикой, их тесная связь друг с другом. Выпускники должны четко понимать не только теоретическую часть материала, но и осознавать роль стимулирующего влияния теории на развитие практических приложений.

 Твердое владение основными методами информатики. В процессе обучения студенты сталкиваются со многими общими методами, такими как синтез, анализ, абстракция, рекурсия, эволюционные изменения, индукция и дедукция. Выпускники должны осознавать широту применения этих методов в области информатики и не сводить их применимость только к тому материалу, в рамках которого они и были представлены.

 Опыт участия в большом проекте. Для того чтобы выпускники умели грамотно применять полученные знания, они обязательно должны принять участие хотя бы в одном реальном проекте. Такого рода опыт обучает студентов практически использовать приобретенные навыки и заставляет их интегрировать материал, изученный в различных учебных курсах.

 Адаптируемость. Одной из основных характеристик информатики на протяжении всей ее относительно небольшой истории является очень быстрый темп изменений. Поэтому выпускники должны обладать глубокими фундаментальными знаниями, помогающими им вырабатывать новые необходимые навыки по мере того, как эволюционирует область.

Ускоренное развитие компьютерной техники практически не учитывается при разработке стандартов образования, типовых программ дисциплин, определяющих основное содержание программы подготовки будущих информатиков. Современная образовательная программа подготовки будущих информатиков должна фокусироваться на правильном моделировании подготовки студентов, т.е. на проектировании и построении таких методических систем обучения, которые поощряют студентов самостоятельно приобретать новые знания, умения и навыки.

Таким образом, подготовка будущих информатиков, направленная на формирование профессиональной компетенции, означает обучение посредством формирования у них потенциальных способностей самостоятельного получения знаний, их применения и углубления.

Так как в профессиональной компетентности главная роль отводится проблемно-практическому аспекту, то при подготовке будущих информатиков особое внимание необходимо уделять углублению знаний, содержательно-мотивационному обучению, ориентированному на овладение основными методами информатики. Все это составляют мотивационно-генетические предпосылки формирования логической культуры будущих информатиков.

Тема 2 Структура подготовки будущих программистов

Формирование логической культуры будущих информатиков предполагает обучение студента основным методам информатики путем углубления знаний по курсам информатики, раскрывая их содержательную часть относительно логических методов, необходимых для дальнейшего применения в профессиональной сфере деятельности.

Концепция логической культуры интересует нас, прежде всего, как одна из важнейших компонент определяющих уровень профессиональной состоятельности и конкурентоспособности будущего информатиков в условиях грядущей информационной цивилизации. В этом аспекте понятия логической и математической культуры представляются как максимально совместимые, хотя и не равнозначные. Кроме того, применительно к профессиональной деятельности будущего информатика это понятие приобретает определенную специфику.

Выявление содержания понятия «логическая культура» неизбежно приводят таким понятиям, как мышление, язык, логика, логическое исчисление.

Орудием мышления являются естественные языки, а также абстрактные и конкретные образные знаковые системы. Элементы этих систем используются для проявления (т.е. своеобразной «материализации») оперативных и операционных функций

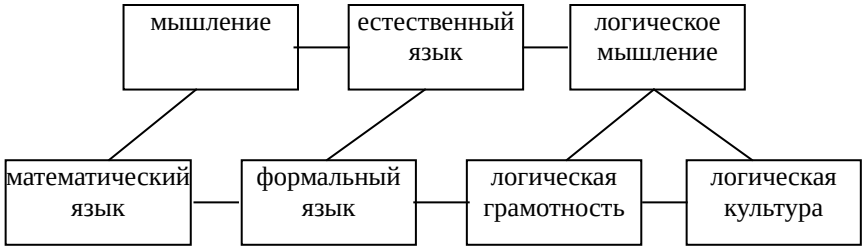


Рисунок 1- Структурная схема взаимосвязи системы понятий, сопряженных с понятием «логическая культура»

мышления и фиксации их средствами формализованных символических языков.

Постижение оперативных особенностей мышления, выявление правил и знаков, которым подчиняется процесс мыслительной деятельности и тех форм, в которых она осуществляется, занимает целую эпоху в развитии самопознания человека. В эту эпоху, на основе постижения связей мышления с языком был осуществлен переход от интуитивно-содержательных представлений об объектах, явлениях и процессах окружающей действительности к их абстрактному восприятию и формальному описанию. В рамках логических исчислений, построенных в это период, были выявлены основные схемы, принципы и рецепты разработки формальных символических языков, отражающих и «материализующих» опыт человеческого мышления. В формальных языках логических исчислений соединилась логика мышления и математика.

По сути своего значения «логика» заключается в том, что она учит, как правильно по форме (структуре) построить рассуждение, чтобы при условии верного применения формально-логических законов, прийти к истинному выводу из истинных посылок, расширяющему наши знания. Соблюдение требований логики – непременное условие последовательного, непротиворечивого, обоснованного мышления [48].

В соответствии с вышеизложенным логическая культура включает:

- определенную совокупность знаний о средствах мыслительной деятельности, ее формах и законах;

- умение использовать эти знания в практике мышления — оперировать понятиями, правильно производить те или иные логические операции с ними, строить умозаключения, доказывать их или опровергать;

- навыки анализа мыслей — как своих собственных, так и чужих, с тем чтобы вырабатывать наиболее рациональные способы рассуждения, предотвращать логические ошибки, а если они допущены, находить и устранять их [49, 50];

- логическую грамотность, как систему умений и навыков правильного оперирования с синтаксическими конфигурациями формальных языков, посредством которых реализуются описательные функции этих языков и которые образуют символическую составляющую современного математического языка.

Структурная схема взаимосвязи системы понятий, сопряженных с понятием «логическая культура» может быть представлена следующим образом (рисунок 6):

Исходя из мотивационных и математических предпосылок формирования логической культуры будущих информатиков мы можем сформулировать основные составляющие логической культуры будущих информатиков:

1. Знания, умения и навыки в области методологии познания:

- формирование осознанных представлений о методах математического познания и способностей их применения для решения задач прикладного характера на основе овладения системой средств научно-мировоззренческого и идейно-методологического потенциалов, свойственных логико-алгебраическим дисциплинам;

- актуализация представлений об основополагающей роли методологии логико-алгебраических наук в научном познании.

2. Знания и умения по логическому исчислению

- овладение технологиями построения синтаксических составляющих формальных языков и их семантик;

- выявление и использование выразительных возможностей формальных языков;

- формирование навыков и умения выявления алгоритмических свойств формальных языков как прообразов свойств эффективности языков программирования.

3. Умения применять метод «от абстракции»

- формализация понятий «Доказательство» и «Алгоритм»;

- формализация понятия «Определимость»;

- формализация понятия «Эффективная вычислимость».

4. Навыки по применению индуктивных и дедуктивных методов в области программирования

- построение языка;

- построение семантик;

- построение класса частично-рекурсивных функций в форме логического исчисления;

- использование формальных языков для описания структурных свойств алгебраических систем.

Тем самым, при подготовке будущих информатиков необходимо уделять особое внимание обучению применения, определенных выше, составляющих логической культуры во всей предметной области информатики (Рисунок 6). Это обязывает педагогов вуза, при проведении текущих занятий, раскрывать рассматриваемые учебные темы в контексте междисциплинарных связей между математикой и информатикой.

Поскольку изучение информатики и математики в подготовке будущих информатиков подчинено, прежде всего, прикладным целям, то важно правильно расставить акценты в понимании сущности изучаемого объекта, которые помогают студенту усваивать связи между структурными элементами учебного материала при изучении различных предметов и способствуют формированию профессиональной деятельности.

Тема 3 Математические предпосылки основных интеллектуальных средств

Структуру формирования логической культуры будущих информатиков определим, применяя математические методы к выявлению оптимального порядка введения системы дидактических единиц при изучении конкретных дисциплин (или отдельных разделов этих дисциплин).

Развивая технологический подход к построению математических моделей образования, предложенный Абрамовым А.В.[73], мы смогли построить и проанализировать некоторые из возможных математических моделей множеств дидактических единиц, что позволило применить математические методы к определению очередности их введения.

Являясь основой данной дисциплины, множество дидактических единиц определяет ее содержание. В связи с этим порядок их введения играет существенную роль при изучении этой дисциплины.

Если $M=\{x_1; x_2; ...; x_n\}$ - некоторое множество дидактических единиц, то при практическом выявлении связей и зависимостей между ними используется отношение P:

$(\forall x, y \in M)(xPy \Leftrightarrow$ изучение дидактической единицы y опирается (в той или иной степени) на дидактическую единицу x).

Естественной (и обозримой) моделью P-связей и отношений между дидактическими единицами множества M является ориентированный граф $G(P)$. Множеством вершин этого графа являются элементы множества M. Ориентированные ребра определяют некоторые упорядоченные пары $\langle x_i; x_j \rangle$ элементов $x_i; x_j \in M$. При этом:

$\langle x_i; x_j \rangle$ - ребро графа $G(P) \Leftrightarrow x_i P x_j$, $(i; j \in \{1;2;...;n\})$.

Так как невозможно изучать дидактическую единицу x, опираясь на эту же самую дидактическую единицу, то отношение P иррекфлексивно, т.е.

$(\forall x \in M)(\neg xPx)$, что гарантирует отсутствие у графа $G(P)$ петель (контуров длины 1). Но в этом графе могут быть контуры длины $l \geq 2$, что обусловлено спецификой отношения P. В частности среди дидактических единиц множества M могут существовать логически несравнимые единицы – понятия, объемы

которых имеют непустую общую часть. В связи с возможным наличием таких контуров модель $G(P)$ не обеспечивает наличия предпосылок получения порочного круга в системе определения понятий (как дидактических единиц из M).

В целях обеспечения логической безупречности, эти контуры необходимо выявить и преобразовать их в незамкнутые цепи (путем удаления некоторых ребер) и только после этого приступить к выявлению порядка введения дидактических единиц из совокупности M. С этой целью используется аппарат теории матриц. А именно, по графу $G(P)$ строится квадратная матрица $A(P) = \|a_{ij}\|$ $(i; j = 1;2;...;n)$ порядка n, где

$$a_{ij} = \begin{cases} 1, & \text{если } (x_i P x_j) \\ 0, & \text{если } \neg(x_i P x_j) \end{cases}$$

, для любых $x_i; x_j \in M$ $(i; j = 1;2;...;n)$.

Приведем некоторые свойства матрицы *A*:

а) Если $(A(P))^m = \|b_{ij}\|$, то $b_{ij} \in N$ и b_{ij} равно числу путей длины m из x_i в x_j в графе $G(P)$ (при этом следует иметь в виду, что путь из x_i в x_j длины m может содержать в себе, в качестве подпути, контуры), $i; j = 1;2;...;n$.

б) Пусть $(A(P))^m = \|b_{ij}\|$; $(A(P))^{m+1} = \|c_{ij}\|$. Если $b_{ij} \neq 0$, а $c_{ij} = 0$, то максимально возможная длина пути из вершины x_i в вершину x_j в графе $G(P)$ равна m (отсюда вытекает, в частности, что все пути соединяющие вершины x_i и x_j являются простыми).

в) Если матрица A не имеет нулевых столбцов, то граф $G(P)$ содержит контуры длины $l \geq 2$.

Из этих свойств вытекает следующий признак отсутствия контуров в графе $G(P)$:

Граф $G(P)$ не имеет контуров тогда и только тогда, когда $(A(P))^n$ - нулевая матрица (n - число дидактических единиц в множестве M).

Возведение матрицы $A(P)$ в n-ую степень (даже при больших значениях n), при условии использования компьютера, представляет несложную задачу.

После, возможно неоднократного применения процедуры выявления и удаления контуров, мы перейдем к графу $G^*(P)$ и его матрице $A^*(P)$, для которой будет выполнятся отмеченный выше признак:

$(A^*(P))^m = \|0\|_{i;j}$, $(i; j = 1;2;...;n)$,

для некоторого m такого, что $1 \leq m \leq n$.

Анализ матрицы $A^*(P)$ и позволит выявить последовательность изучения дидактических единиц множества M.

Процесс выявления порядка основывается на следующем свойстве матрицы $A^*(P)$:

- если j-ый столбец матрицы $A^*(P)$ является нулевым, то дидактическая единица x_j является минимальным элементом в M (относительно P-упорядочения множества M); другими словами: дидактическую единицу x_j нужно изучать в первую очередь.

Используя это свойство, очередность изучения дидактических единиц из M будет определяться по шагам:

Шаг 1 Пусть $j_1^{(1)}; j_2^{(1)};...; j_{k_1}^{(1)}$ - номера нулевых столбцов матрицы $A_1 = A^*(P)$. Тогда дидактические единицы

$x_{j_1^{(1)}}; x_{j_2^{(1)}};...; x_{j_{k_1}^{(1)}}$

изучаются (в любом порядке) в первую очередь.

Вычеркиваем в матрице A_1 строки и столбцы с номерами $j_1^{(1)}; j_2^{(1)};...; j_{k_1}^{(1)}$; полученную матрицу обозначим через A_2 и переходим к следующему шагу.

Шаг 2 Пусть $j_1^{(2)}; j_2^{(2)};...; j_{k_2}^{(2)}$ - номера (в исходной нумерации) нулевых столбцов матрицы A_2. Тогда дидактические единицы $x_{j_1^{(2)}}; x_{j_2^{(2)}};...; x_{j_{k_2}^{(2)}}$ должны изучаться (в произвольном порядке) во вторую очередь. Как и на предыдущем шаге, вычеркиваем в матрице

A_2 строки и столбцы с номерами $j_1^{(2)}; j_2^{(2)};...; j_{k_2}^{(2)}$; полученную матрицу обозначим через A_3 и переходим к следующему шагу и т.д.

Указанный пошаговый процесс будет осуществляться до тех пор пока не будут вычеркнуты последняя строка и последний столбец матрицы $A^*(P)$. Если это произойдет на шаге с номером S, то множество M дидактических единиц разобьется на S непересекающихся групп:

$x_{j_1^{(1)}}; x_{j_2^{(1)}};...; x_{j_{k_1}^{(1)}};$

$x_{j_1^{(2)}}; x_{j_2^{(2)}};...; x_{j_{k_2}^{(2)}};$

- - - - - - - - - - -

$x_{j_1^{(s)}}; x_{j_2^{(s)}};...; x_{j_{k_s}^{(s)}}.$

Группы дидактических единиц изучаются в полученном порядке. Дидактические единицы в рамках каждой из отдельной групп могут изучаться в произвольном порядке [74,75].

Тема 4 Алгебра высказываний

Основными объектами рассмотрения являются высказывания. Под высказыванием понимают повествовательное предложение, о котором можно сказать одно из двух: истинно оно или ложно.

Пусть есть множество высказываний, фраз, принимающих значение «истина» или «ложь». Примером могут быть фразы «сегодня холодно».

«Идёт дождь», «Асет Темиров учится в группе А-201», «Президент Казахстана поехал в Китай» и др. Будем называть их *элементарными* высказываниями и обозначать прописными буквами латинского алфавита. При этом отвлечёмся от конкретного смысла высказывания, оставим только его истинностное значение.

В исчислении высказываний не рассматриваются утверждения, имеющие значения, отличные от значений «истинно» и «ложно». Не рассматривается и трёхзначная логика, со значениями, скажем «Да», «Нет», «Не знаю». Ответ отличный от «Да» должен быть «Нет». Древние философы называли этот принцип законом исключения третьего.

Высказывание – это утверждение, которое может быть только истинно или ложно. Его принято обозначать символами Т (от True), или F (от False), или соответственно, 1 (для истинного значения) или 0 (для значения ложь).

Значение высказывания зависит от предметной области. Например, высказывание «число 15 – простое» будет истинным в восьмеричной и ложным в десятичной системе счисления. Поэтому весьма важно конкретизировать область на которой определено употребляемое высказывание.

Из элементарных высказываний строятся более сложные высказывания с помощью логических связок «НЕ», «И», «ИЛИ», «ТО ЖЕ, ЧТО» («ЭКВИВАЛЕНЦИЯ»), «ИЗ … СЛЕДУЕТ…», (« … ВЛЕЧЁТ…», «…ПОТОМУ, ЧТО…»). Эти связки называются *сентенциональными*. Связкилогики высказываний представляют функции истинности или функции алгебры логики.

В таб.1.1 представлены логические связки и их обозначения.

Табли

Название	Тип	Обозна-чение	Как читается	Други обозначе
Отрицание	Унарный	¬	не	$\bar{s}$, not,
Конъюнкция	Бинарный	∧	и	&, . , and
Дизъюнкция	Бинарный	∨	или	| , or, ил
Импликация	Бинарный	→	влечёт	⇒, ⊃
Эквивалент-ность	Бинарный	⇔	эквивалентно	↔, ≈

Определение 1. Отрицанием высказывания p называется высказывание ¬p (или -p), которое истинно только тогда, когда p ложно.

Пример. Высказывание «Неправда, что идёт снег» является отрицанием высказывания «идёт снег».

Определение 2. Конъюнкцией высказываний p и q называется высказывание, которое истинно только тогда, когда p и q истинны., т.е. $p = 1$ и $q = 1$.

Пример. Чтобы успешно сдать экзамен, нужно иметь при себе зачётку и правильно ответить на вопросы. Для успешной сдачи экзамена нужно выполнить оба условия. Если обозначить как p – «иметь зачётку» и q – «правильно ответить на вопросы», то условием сдачи будет конъюнкция высказываний p&q.

Определение 3. Дизъюнкцией высказываний p и q называется высказывание, которое ложно тогда и только тогда, когда оба высказывания ложны, т. е. p= 0 и q =0.

Примеры. (7 >3 или 4 ≠ 1) =1; (или sin2x имеет период 2π, или √-2 – рациональное число) = 0.

Определение 4. Импликацией высказываний p и q называется высказывание, которое ложно тогда и только тогда, когда p истинно, q ложно, т.е. p = 1 и q = 0 (из p следует q).

Пример. Вышеприведённый пример с успешной сдачей экзамена можно записать как p&q → r, где r – «успешно сдать экзамен».

Определение 5. Эквиваленцией высказываний p и q называется высказывание, которое истинно только и только тогда, когда значения высказываний p и q совпадают (p эквивалентно q).

Пример. «Граф является двудольным тогда и только тогда, когда он не содержит циклов нечётной длины». Если p – высказывание «иметь циклы нечетной длины», *q* – «граф двудольный», то начальная фраза примера запишется в виде $q \Leftrightarrow \neg p$.

Значения истинности для бинарных связок представлены в табл.1.2.

Таблица

p	q	$\bar{p}$	$p \vee q$	$p \,\&\, q$	$p \to q$	$P \Leftrightarrow q$
0	0	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	1	1	1

С помощью связок можно получать составные высказывания, которым соответствует *формула*, например, (A & B) → (-A v B). Такие высказывания называют *сложными*. Каждое сложное высказывание, как и элементарное принимает значение из множества {F, T}. В формулах используются скобки для определения порядка выполнения действий.

Пример. Пусть значения элементарных высказываний: p1 = 1, p2 = 0, p3 = 1 и имеется составное высказывание:

((¬p1 ∧ p2) → p3) ⇔ (¬p2 ∨ p3).

Найти значение сложного высказывания.

Решение:

((¬p1 ∧ p2) → p3) ⇔ (¬p2 ∨ ¬p3)

0 0 1 1 0

0 1 1

1

Ответ: Значение сложного высказывания – 1.

Итак, словарь исчисления высказываний даёт возможность строить сложные высказывания из простых или элементарных, соединяя последние связками. Получаем формулы, которые являются объектами языка. Аналогия с естественными языками очевидна: фраза – это составное высказывание, построенное по определённым правилам.

Совокупность правил построения выглядит так:

• *Базис*. Всякое высказывание является формулой.

• *Индукционный шаг*. Если A и B формулы, то ¬A, (A ∨ B), (A ∧ B), (A → B), (A ⇔B) – формулы.

• *Ограничение*. Формула однозначно получается с помощью правил, описанных в базисе и индукционном шаге.

Множество всех формул *счётное* (можно установить взаимно однозначное соответствие между ним и множеством натуральных чисел), *разрешимое* (можно достоверно выяснить, является ли данное высказывание формулой или нет).

Важное значение в теории выбора и принятия решений имеет понятие бинарного отношения, позволяющее формализовать операции попарного сравнения альтернатив.

Для любых двух множеств X и Y (различных или нет) существует единственное множество, состоящее из всех упорядоченных пар (x,y), где $x \in X$, $y \in Y$. Оно обозначается символами $X * Y$ и называется декартовым произведением (или просто произведением) X и Y. При этом $X * X$ обозначается как X^2.

Бинарным отношением на множестве A называется произвольное подмножество R множества A^2. Имеем, следовательно, $R \subseteq A^2$, в том числе $A^2 \subseteq A^2$.

Существует наглядный способ задания бинарных отношений на конечных множествах. Изобразим элементы конечного множества A точками на плоскости. Если задано отношение $R \subseteq A^2$ и $(a_i, a_j) \in R$, где $a_i \in A$, $a_j \in A$, то проведем стрелку от a_i к a_j. Если $(a_i, a_i) \in R$, то у точки a_i нарисуем петлю-стрелку, выходящую из a_j и входящую в ту же точку. Получившаяся фигура называется ориентированным графом или просто графом, а сами точки - вершинами графа. Заметим, что вместо $(a_i, a_j) \in R$ можно писать $a_i R a_j$.

Дополнение $\overline{R}$ отношения R определяется следующим образом: $a_1 \overline{R} a_2$ означает, что $(a_1, a_2) \notin R$, т.е. неверно, что a_1 находится в отношении R с a_2 .

Бинарное отношение R на множестве A называется:

полным , если любые два элемента из A связаны отношением R;

рефлексивным, если aRa для всех $a \in A$;

симметричным , если $a_1 R a_2 \Rightarrow a_2 R a_1$ для всех $a_1, a_2 \in A$;

антирефлексивным, если $a_1 R a_2 \Rightarrow a_1 \neq a_2$;

антисимметричным, если для всех $a_1, a_2 \in A$ из $a_1 R a_2$, $a_2 R a_1$ следует $a_1 = a_2$;

асимметричным, если $a_1 R a_2 \Rightarrow a_2 \overline{R} a_1$ для всех $a_1, a_2 \in A$;

транзитивным, если для всех $a_1, a_2, a_3 \in A$: $a_1 R a_2$, $a_2 R a_3 \Rightarrow a_1 R a_3$;

эквивалентностью, если R транзитивно, рефлексивно и симметрично (если R - эквивалентность и $(a_1, a_2) \in R$, то этот факт будем обозначать также $a_1 \sim a_2$);

квазипорядком , если R транзитивно и рефлексивно;

строгим порядком, если оно антирефлексивно и транзитивно;

порядком, если R транзитивно и антисимметрично.

В случае, если одновременно $a_1 \overline{R} a_2$ и $a_2 \overline{R} a_1$, альтернативы a_1 и a_2 называются несравнимыми по бинарному отношению R. Если не существует несравнимых альтернатив, то отношение R называется линейным (не путать со свойством полноты).

Упражнения.

а). Докажите, что если отношение R симметрично и транзитивно, то оно и рефлексивно (следовательно, при определении эквивалентности можно было ограничиться только двумя требованиями);

б). Докажите, что из асимметричности следует антирефлексивность;

в). Подумайте, как приведенные свойства бинарных отношений могут быть представлены (проиллюстрированы) на языке ориентированных графов.

Содержательные примеры рефлексивных отношений: “быть похожим на”, “быть не старше”. С другой стороны, отношения типа “быть братом”, “быть старше” не являются рефлексивными. В графе, изображающем рефлексивное отношение, каждая вершина имеет петлю.

Примерами симметричных отношений служат “быть похожим на”, “быть одинаковым с “, “быть родственником”. В соответствующем графе вместе с каждой стрелкой, идущей из вершины a_i в вершину a_j , существует и противоположно

направленная стрелка (следовательно, симметричное отношение естественнее изображать неориентированным графом).

Свойство транзитивности также легко интерпретируется на графе. Именно, если точки a_i и a_j соединены путем, проходимым по направлению стрелок, то существует стрелка, непосредственно идущая из вершины a_i в вершину a_j. Примером транзитивного отношения может служить отношение “больше” на множестве вещественных чисел. Ясно также, что граф антирефлексивного (а значит и асимметричного) отношения не может иметь петель.

Примерами строгого порядка могут служить отношение “ $<$ “ для вещественных чисел и отношение строгого включения “ $\subset$ “ для множеств.

Большинство из приведенных свойств бинарных отношений прямо или косвенно будет использовано в дальнейшем изложении.

Упорядоченной парой $\langle x, y \rangle$ называется совокупность, состоящая из двух элементов x и y, взятых в определенном порядке: элемент x считается в паре первым, а элемент y — вторым. Две упорядоченные пары $\langle x_1, y_1 \rangle$ и $\langle x_2, y_2 \rangle$ называются **равными** тогда и только тогда, когда $x_1 = x_2$ и $y_1 = y_2$.

Прямым (декартовым) произведением множеств X и Y называется множество всех упорядоченных пар $\langle x, y \rangle$ таких, что $x \in X$ и $y \in Y$. Прямое произведение обозначается $X \times Y$, а в случае $Y = X$ — просто X^2, т.е. $X \times X = X^2$.

Аналогично определяются упорядоченные тройки, четверки и т.д., а также прямые произведения трех, четырех и т.д. множеств. Например, прямым произведением множеств $\mathbb{R}$ действительных чисел называется множество всех упорядоченных наборов $\langle x_1, x_2, \ldots, x_n \rangle$ из n действительных чисел $x_1, x_2, \ldots, x_n$.

Бинарным отношением р на множестве $X \times Y$ называется подмножество ρ этого множества упорядоченных пар $\langle x, y \rangle,\ x \in X,\ y \in Y$. Если пара $\langle x, y \rangle$ принадлежит отношению ρ, то пишут $\langle x, y \rangle \in \rho$ или $x \rho y$. Если $Y = X$, то отношение ρ, т.е. подмножество множества X^2, называют **бинарным отношением на множестве** X.

Бинарное отношение ρ на множестве X называется:

— рефлексивным, если $x \rho x$ для любого $x \in X$;

— симметричным, если для любых $x, y \in X$ из $x \rho y$ следует, что $y \rho x$;

— транзитивным, если для любых $x, y, z \in X$ из $x \rho y$ и $y \rho z$ следует, что $x \rho z$.

Рефлексивное, симметричное и транзитивное отношение на множестве X называется **отношением эквивалентности на множестве** X и обозначается символом $\sim$.

Примеры. Даны бинарные отношения:

а) отношение $=$ ($x = y$ — "x равен y") на множестве действительных чисел;

б) отношение $<$ ($x < y$ — "x меньше y") на множестве действительных чисел;

в) отношение $\leqslant$ ($x \leqslant y$ — "x не больше y") на множестве действительных чисел;

г) отношение $\mathbf{B}$ ($x \mathbf{B} y$ — "x брат y") на множестве людей;

д) отношение $\sim$ ($M \sim N$ — "многоугольник M подобен многоугольнику N") на множестве правильных многоугольников;

е) отношение $m = n \pmod p$ на множестве целых чисел: "число m сравнимо с числом n по модулю p", т.е. остатки от деления чисел m и n на натуральное число p равны.

Установить, являются ли заданные отношения рефлексивными, симметричными, транзитивными, отношениями эквивалентности.

Решение:

а) Так как $x = x$для любого действительного числа x, то отношение $=$рефлексивное. Поскольку из $x = y$следует, что $y = x$, то отношение симметричное. Так как из равенств $x = y$и $y = z$ следует, что $x = z$, то отношение транзитивное. Таким образом, отношение равенства является отношением эквивалентности.

б) Отношение "меньше" не является рефлексивным (неравенство $x < x$неверно) и симметричным (из $x < y$не следует $y < x$но является транзитивным (так как из неравенств $x < y$и $y < z$следует $x < z$). Это отношение не является отношением эквивалентности.

в) Отношение "не больше" является рефлексивным (неравенство $x \leqslant x$справедливо для любых действительных чисел) и транзитивным (из неравенств $x \leqslant y$и $y \leqslant z$следует $x \leqslant z$), но не является симметричным (например, из $1 \leqslant 2$не следует, что $2 \leqslant 1$). Это отношение не является отношением эквивалентности.

г) Отношение "братства" не является рефлексивным (любой человек не является братом для самого себя), симметричным (утверждение, если xбрат y $(x\mathbf{B}y)$, то yбрат x $(x\mathbf{B}y)$неверно, поскольку yможет оказаться сестрой для x), транзитивным (например, если для трех людей x, y, zимеем $x\mathbf{B}y$и $y\mathbf{B}z$, то отсюда не следует, что $x\mathbf{B}z$, поскольку zможет оказаться сестрой для x). Это отношение не является отношением эквивалентности.

д) Каждый многоугольник подобен самому себе $M \sim M$. Поэтому отношение подобия рефлексивное. Из подобия многоугольников $M \sim N$следует, что $N \sim M$, значит отношение симметричное. Так как из подобия многоугольников $M \sim N$и $N \sim K$следует, что $M \sim K$, то отношение транзитивное. Таким образом, отношение подобия многоугольников является отношением эквивалентности.

е) Сравнение $m = n \pmod p$равносильно условию: разность $m - n$делится на p(без остатка). Так как число нуль $m - m = 0$делится без остатка на любое натуральное число p, то $m = m \pmod p$, значит отношение рефлексивное. Если $m - n$ делится на p, то и $n - m$делится на p, следовательно, отношение симметричное. Наконец, если числа $m - n$и $n - k$делятся на число p, то и их сумма $(m - n) + (n - k) = m - k$делится на p, т.е. из $m = n \pmod p$и $n = k \pmod p$следует, что $m = k \pmod p$. Поэтому отношение транзитивное. Таким образом, отношение сравнения целых чисел по модулю pявляется отношением эквивалентности.

Говорят, что **множество Xразбито на классы**, если оно представлено тем или иным способом в виде объединения своих попарно непересекающихся подмножеств. Например, множество всех студентов вуза разбивается на учебные группы (а множество учеников школы разбивается на классы). Любое разбиение множества Xна классы определяет на Xотношение: $x \sim y$— "x находится в том же классе, что и y". Покажем, что это отношение, обозначенное символом $\sim$, действительно является отношением эквивалентности. В самом деле, оно рефлексивное: $x \sim x$, симметричное: $x \sim y \Rightarrow y \sim x$(если xнаходится в том же классе, что и y, то и y находится в том же классе, что и x), транзитивное (из $x \sim y$и $y \sim z$ следует, что все три элемента x, y, zпринадлежат одному классу, тогда и $x \sim z$). Следовательно, рассматриваемое отношение является отношением эквивалентности

Справедливо и обратное утверждение. Любое отношение эквивалентности $\sim$, заданное на множестве X, позволяет разбить это множество на непустые классы.

Классом эквивалентности, порожденным элементом x, называется подмножество K_xмножества X, состоящее из тех элементов $y \in X$, для которых $x \sim y$ Любой класс K_x— непустое

множество, так как, в силу рефлексивности $x \sim x$, он содержит по крайней мере один элемент x.

Таким образом, отношение эквивалентности на множестве X определяет разбиение множества Xна непустые классы эквивалентности относительно этого отношения. Каждый класс эквивалентности однозначно определяется любым своим элементом. Совокупность классов эквивалентности называется **фактор-множеством** множества X.

Например, отношение подобия (см. пункт "д" примера В.2) разбивает множество правильных многоугольников на классы эквивалентности: множество правильных треугольников, множество квадратов и т.д. Отношение сравнения целых чисел по модулю p(см. пункт "е" примера В.2) разбивает множество целых чисел на p классов эквивалентности, поскольку при делении на pколичество различных остатков $(0; 1; \dots; p-1)$равно p.

Тема 5 Алгебра предикатов

Два частично упорядоченных множества называются изоморфными, если между ними существует изоморфизм, то есть взаимно однозначное соответствие, сохраняющее порядок. (Естественно, что в этом случае они равномощны как множества.) Можно сказать так: биекция $f\colon A \to B$ называется изоморфизмом частично упорядоченных множеств A и B, если

$$a_1 \leqslant a_2 \;\Leftrightarrow\; f(a_1) \leqslant f(a_2)$$

для любых элементов $a_1, a_2 \in A$ (слева знак $\leqslant$обозначает порядок в множестве A, справа - в множестве B).

Очевидно, что отношение изоморфности рефлексивно (каждое множество изоморфно самому себе), симметрично (если X изоморфно Y, то и наоборот) и транзитивно (два множества, изоморфные третьему, изоморфны между собой). Таким образом, все частично упорядоченные множества разбиваются на классы изоморфных, которые называют порядковыми типами. (Правда, как и с мощностями, тут необходима осторожность - изоморфных множеств слишком много, и потому говорить о порядковых типах как множествах нельзя.)

Теорема 3. Конечные линейно упорядоченные множества из одинакового числа элементов изоморфны.

Доказательство. Конечное линейно упорядоченное множество всегда имеет наименьший элемент (возьмем любой элемент; если он не наименьший, возьмем меньший, если и он не наименьший, еще меньший - и так далее; получим убывающую последовательность $x > y > z > \dots$, которая рано или поздно должна оборваться). Присвоим наименьшему элементу номер 1. Из оставшихся снова выберем наименьший элемент и присвоим ему номер 2и так далее. Легко понять, что порядок между элементами соответствует порядку между номерами, то есть что наше множество изоморфно множеству $\{1, 2, \dots, n\}$.

Докажите, что множество всех целых положительных делителей числа 30с отношением " быть делителем" в качестве отношения порядка изоморфно множеству всех подмножеств множества $\{a, b, c\}$, упорядоченному по включению.

Будем рассматривать финитные последовательности натуральных чисел, то есть последовательности, у которых все члены, кроме конечного числа, равны 0. На множестве таких последовательностей введем покомпонентный порядок: $(a_0, a_1, \dots) \leqslant (b_0, b_1, \dots)$, если $a_i \leqslant b_i$ при всех i. Докажите, что это множество изоморфно множеству всех положительных целых чисел с отношением " быть делителем" в качестве порядка.

Взаимно однозначное отображение частично упорядоченного множества A в себя, являющееся изоморфизмом, называют автоморфизмом частично упорядоченного множества A. Тождественное отображение всегда является автоморфизмом, но для некоторых множеств существуют и другие автоморфизмы. Например, отображение прибавления единицы ($x \mapsto x+1$) является автоморфизмом частично упорядоченного множества $\mathbb{Z}$ целых чисел (с естественным порядком). Для множества натуральных чисел та же формула не дает автоморфизма (нет взаимной однозначности).

Вот несколько примеров равномощных, но не изоморфных линейно упорядоченных множеств (в силу теоремы 12 они должны быть бесконечными).

Отрезок $[0,1]$ (с обычным отношением порядка) не изоморфен множеству $\mathbb{R}$, так как у первого есть наибольший элемент, а у второго нет. (При изоморфизме наибольший элемент, естественно, должен соответствовать наибольшему.)

Множество $\mathbb{Z}$ (целые числа с обычным порядком) не изоморфно множеству $\mathbb{Q}$ (рациональные числа). В самом деле, пусть $\alpha\colon \mathbb{Z} \to \mathbb{Q}$ является изоморфизмом. Возьмем два соседних целых числа, скажем, 2 и 3. При изоморфизме α им должны соответствовать какие - то два рациональных числа $\alpha(2)$ и $\alpha(3)$, причем $\alpha(2) < \alpha(3)$, так как $2 < 3$. Но тогда рациональным числам между $\alpha(2)$ и $\alpha(3)$ должны соответствовать целые числа между 2 и 3, которых нет.

Более сложный пример - множества $\mathbb{Z}$ и $\mathbb{Z}+\mathbb{Z}$. Возьмем в $\mathbb{Z}+\mathbb{Z}$ две копии нуля (из той и другой компоненты); мы обозначали их 0 и $\overline{0}$. При этом $0 < \overline{0}$. При изоморфизме им должны соответствовать два целых числа a и b, для которых $a < b$. Тогда всем элементам между 0 и $\overline{0}$ (их бесконечно много: 1, 2, 3, $\ldots$, $-\overline{3}$, $-\overline{2}$, $-\overline{1}$) должны соответствовать числа между a и b - но их лишь конечное число.

Этот пример принципиально отличается от предыдущих тем, что здесь разницу между свойствами множеств нельзя записать формулой. Как говорят, упорядоченные множества $\mathbb{Z}$ и $\mathbb{Z}+\mathbb{Z}$ "элементарно эквивалентны".

Докажите, что линейно упорядоченные множества $\mathbb{Z}\times\mathbb{N}$ и $\mathbb{Z}\times\mathbb{Z}$ (с описанным выше порядком) не изоморфны.

Будут ли изоморфны линейно упорядоченные множества $\mathbb{N}\times\mathbb{Z}$ и $\mathbb{Z}\times\mathbb{Z}$?

Будут ли изоморфны линейно упорядоченные множества $\mathbb{Q}\times\mathbb{Z}$ и $\mathbb{Q}\times\mathbb{N}$?

Отображение $x \mapsto \sqrt{2}x$ осуществляет изоморфизм между интервалами $(0,1)$ и $(0,\sqrt{2})$. Но уже не так просто построить изоморфизм между множествами рациональных точек этих интервалов (то есть между $\mathbb{Q}\cap(0,1)$ и $\mathbb{Q}\cap(0,\sqrt{2})$), поскольку умножение на $\sqrt{2}$ переводит рациональные числа в иррациональные. Тем не менее изоморфизм построить можно. Для этого надо взять возрастающие последовательности рациональных чисел $0 < x_1 < x_2 < \ldots$ и $0 < y_1 < y_2 < \cdots$, сходящиеся соответственно к 1 и $\sqrt{2}$ и построить кусочно - линейную функцию f, которая переводит x_i в y_i и линейна на каждом из отрезков $[x_i, x_{i+1}]$ (1). Легко понять, что она будет искомым изоморфизмом.

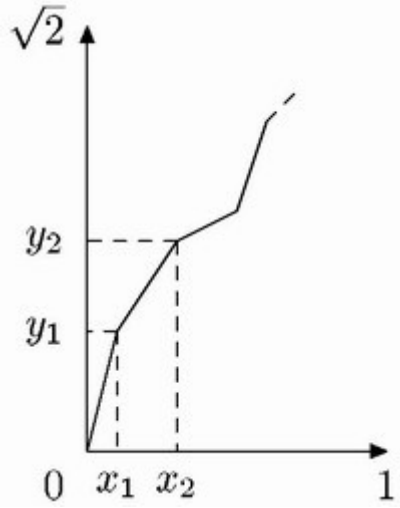


Рис. 1. Ломаная осуществляет изоморфизм

Покажите, что множество рациональных чисел интервала $(0,1)$ и множество $\mathbb{Q}$ изоморфны. (Указание: здесь тоже можно построить ломаную; впрочем, у этой задачи есть и другое решение,

которое начинается с того, что функция $x \mapsto 1/x$ переводит рациональные числа в рациональные.)

Более сложная конструкция требуется в следующей задаче (видимо, ничего проще, чем сослаться на общую теорему 13, тут не придумаешь).

Докажите, что множество двоично - рациональных чисел интервала $(0,1)$ изоморфно множеству $\mathbb{Q}$. (Число считается двоично - рациональным, если оно имеет вид $m/2^n$, где m - целое число, а n - натуральное.)

Два элемента x, y линейно упорядоченного множества называют соседними, если $x < y$ и не существует элемента между ними, то есть такого z, что $x < z < y$. Линейно упорядоченное множество называют плотным, если в нем нет соседних элементов (то есть между любыми двумя есть третий).

Теорема 5. Любые два счетных плотных линейно упорядоченных множества без наибольшего и наименьшего элементов изоморфны.

Доказательство. Пусть X и Y - данные нам множества. Требуемый изоморфизм между ними строится по шагам. После n шагов у нас есть два n - элементных подмножества $X_n \subset X$ и $Y_n \subset Y$, элементы которых мы будем называть " охваченными", и взаимно однозначное соответствие между ними, сохраняющее порядок. На очередном шаге мы берем какой - то неохваченный элемент одного из множеств (скажем, множества X) и сравниваем его со всеми охваченными элементами X. Он может оказаться либо меньше всех, либо больше, либо попасть между какими - то двумя. В каждом из случаев мы можем найти неохваченный элемент в Y, находящийся в том же положении (больше всех, между первым и вторым охваченным сверху, между вторым и третьим охваченным сверху и т.п.). При этом мы пользуемся тем, что в Y нет наименьшего элемента, нет наибольшего и нет соседних элементов, - в зависимости от того, какой из трех случаев имеет место. После этого мы добавляем выбранные элементы к X_n и Y_n, считая их соответствующими друг другу.

Чтобы в пределе получить изоморфизм между множествами X и Y, мы должны позаботиться о том, чтобы все элементы обоих множеств были рано или поздно охвачены. Это можно сделать так: поскольку каждое из множеств счетно, пронумеруем его элементы и будем выбирать неохваченный элемент с наименьшим номером (на нечетных шагах - из X, на четных - из Y). Это соображение завершает доказательство.

Теорема 6. Всякое счетное линейно упорядоченное множество изоморфно некоторому подмножеству множества $\mathbb{Q}$.

Доказательство. Заметим сразу же, что вместо множества $\mathbb{Q}$ можно было взять любое плотное счетное всюду плотное множество без первого и последнего элементов, так как они все изоморфны.

Доказательство этого утверждения происходит так же, как и в теореме 13 - с той разницей, что новые необработанные элементы берутся только с одной стороны (из данного нам множества), а пары к ним подбираются в множестве рациональных чисел.

Гомоморфизмы

Определение 1. Пусть Σ_1 и Σ_2 - алфавиты. Если отображение $h\colon \Sigma_1^* \to \Sigma_2^*$ удовлетворяет условию $h(x \cdot y) = h(x) \cdot h(y)$ для всех слов $x \in \Sigma_1^*$ и $y \in \Sigma_1^*$, то отображение h называется гомоморфизмом (морфизмом).

Замечание 2. Можно доказать, что если h - гомоморфизм, то $h(\varepsilon) = \varepsilon$.

Замечание 3. Пусть $\Sigma_1 = \{a, b\}$ и $\Sigma_2 = \{c\}$. Тогда отображение $h\colon \Sigma_1^* \to \Sigma_2^*$, заданное равенством $h(w) = c^{2|w|}$, является гомоморфизмом.

Замечание 4. Каждый гомоморфизм однозначно определяется своими значениями на однобуквенных словах.

Замечание 5. Если $h\colon \Sigma_1^* \to \Sigma_2^*$ - гомоморфизм и $L \subseteq \Sigma_1^*$, то через h(L) обозначается язык $\{h(w) \mid w \in L\}$.

Пример. Пусть $\Sigma = \{a, b\}$ и гомоморфизм $h\colon \Sigma^* \to \Sigma^*$ задан равенствами h(a) = abba и $h(b) = \varepsilon$. Тогда

$$h(\{baa, bb\}) = \{abbaabba, \varepsilon\}.$$

Определение 2. Если $h\colon \Sigma_1^* \to \Sigma_2^*$ - гомоморфизм и $L \subseteq \Sigma_2^*$, то через $h^{-1}(L)$ обозначается язык $\{w \in \Sigma_1^* \mid h(w) \in L\}$.

Пример. Рассмотрим алфавит $\Sigma = \{a, b\}$. Пусть гомоморфизм $h\colon \Sigma^* \to \Sigma^*$ задан равенствами h(a) = ab и h(b) = abb. Тогда

$$h^{-1}(\{\varepsilon, abbb, abbab, ababab\}) = \{\varepsilon, ba, aaa\}.$$

Определение. Порождающей грамматикой (грамматикой типа 0, generative grammar, rewrite grammar) называется четверка $G = \langle N, \Sigma, P, S\rangle$, где N и Σ- конечные алфавиты, $N \cap \Sigma = \varnothing$, $P \subset (N \cup \Sigma)^+ \times (N \cup \Sigma)^*$, P конечно и $S \in N$. Здесь Σ- основной алфавит (терминальный алфавит), его элементы называются терминальными символами или терминалами (terminal), N - вспомогательный алфавит (нетерминальный алфавит), его элементы называются нетерминальными символами, нетерминалами, вспомогательными символами или переменными (nonterminal, variable), S - начальный символ (аксиома, start symbol). Пары $(\alpha, \beta) \in P$ называются правилами подстановки, просто правилами или продукциями (rewriting rule, production) и записываются в виде $\alpha \to \beta$.

Пример. Пусть даны множества N = {S}, $\Sigma = \{a, b, c\}$, $P = \{S \to acSbcS,\ cS \to \varepsilon\}$. Тогда $\langle N, \Sigma, P, S\rangle$ является порождающей грамматикой.

Замечание. Будем обозначать элементы множества Σ строчными буквами из начала латинского алфавита, а элементы множества N - заглавными латинскими буквами. Обычно в примерах мы будем задавать грамматику в виде списка правил, подразумевая, что алфавит N составляют все заглавные буквы, встречающиеся в правилах, а алфавит Σ- все строчные буквы, встречающиеся в правилах. При этом правила порождающей грамматики записывают в таком порядке, что левая часть первого правила есть начальный символ S.

Замечание Для обозначения n правил с одинаковыми левыми частями $\alpha \to \beta_1, \ldots, \alpha \to \beta_n$ часто используют сокращенную запись $\alpha \to \beta_1 \mid \ldots \mid \beta_n$.

Определение. Пусть дана грамматика G. Пишем $\phi \underset{G}{\Rightarrow} \psi$, если $\phi = \eta\alpha\theta$, $\psi = \eta\beta\theta$ и $(\alpha \to \beta) \in P$ для некоторых слов $\alpha,\ \beta,\ \eta,\ \theta$ в алфавите $N \cup \Sigma$. Если $\phi \underset{G}{\Rightarrow} \psi$, то говорят, что слово ψ непосредственно выводимо из слова ϕ.

Замечание. Когда из контекста ясно, о какой грамматике идет речь, вместо $\underset{G}{\Rightarrow}$ можно писать просто $\Rightarrow$.

Пример. Пусть

$$G = \langle \{S\}, \{a, b, c\}, \{S \to acSbcS,\ cS \to \varepsilon\}, S\rangle.$$

Тогда $cSacS \underset{G}{\Rightarrow} cSa$.

Определение. Если $\omega_0 \underset{G}{\Rightarrow} \omega_1 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} \omega_n$, где $n \geqslant 0$, то пишем $\omega_0 \overset{*}{\underset{G}{\Rightarrow}} \omega_n$ и говорим, что слово ω_n выводимо из слова ω_0. Другими словами, бинарное отношение $\overset{*}{\underset{G}{\Rightarrow}}$ является рефлексивным, транзитивным замыканием бинарного отношения $\underset{G}{\Rightarrow}$, определенного на множестве $(N \cup \Sigma)^*$.

При этом последовательность слов $\omega_0, \omega_1, \ldots, \omega_n$ называется выводом (derivation) слова ω_n из слова ω_0 в грамматике G. Число n называется длиной (количеством шагов) этого вывода.

Замечание . В частности, для всякого слова $\omega \in (N \cup \Sigma)^*$ имеет место $\omega y t * \underset{G}{\Rightarrow} \omega$ (так как возможен вывод длины 0).

Пример. Пусть $G = \langle \{S\}, \{a, b\}, \{S \to aSa,\ S \to b\}, S \rangle$. Тогда $aSa \overset{*}{\underset{G}{\Rightarrow}} aaaaSaaaa$. Длина этого вывода - 3.

Определение. Язык, порождаемый грамматикой G, - это множество $L(G) \rightleftharpoons \{\omega \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} \omega\}$. Будем также говорить, что грамматика G порождает (generates) язык L(G).

Замечание Существенно, что в определение порождающей грамматики включены два алфавита - Σ и N. Это позволило нам в определении 1.4.11 "отсеять" часть слов, получаемых из начального символа. А именно, отбрасывается каждое слово, содержащее хотя бы один символ, не принадлежащий алфавиту Σ.

Пример 1.4.13. Если $G = \langle \{S\}, \{a, b\}, \{S \to aSa,\ S \to bb\}, S \rangle$, то $L(G) = \{a^n bba^n \mid n \geqslant 0\}$.

Тема 6 Гомоморфизмы и изоморфизмы

Основной задачей изучения совокупностей алгебраических операций и отношений, в современной её постановке, является задача изучения их абстрактных свойств, то есть природа элементов множеств, на которых задаются эти операции и отношения играет несущественную роль.

В соответствии с этим, алгебраические системы, как органические единства основных множеств, совокупностей основных операций и отношений, изучаются с точностью до изоморфизма, что позволяет выявлять и рассматривать свойства операций и отношений алгебраических систем и их классов в «чистом» виде.

Прежде, чем перейти к определению понятия изоморфизма алгебраических систем, отметим, что если *Ф* — отображение M основного множества системы *M* в основное множество системы *N* и $\tau: X$ означивание множества переменных *X* в *M*, то *Ф* - означивание *X* в *N*, где *Ф* - композиция отображения и *Ф*.

Напомним основные определения. Пусть $M = \langle M; {}^{\varphi}\sigma \rangle$ и $N = \langle N$ - алгебраические системы сигнатуры и *Ф* - биективное отображение основного множества *M* системы *M* на основное множество *N* системы *N*.

Отображение *Ф* называется **изоморфным отображением** или **изоморфизмом** системы *M* в систему *N*, если выполняются следующие условия:

1) $\Phi(F_i^n)$

2)M(=^;^{y}ftM;^M;-;^W)«

3) $c_к[m] = c_к[моф]$

для любых *F'"(x_l;x₂;...;x_{ri})<=F;* P'"(x,;x,;...;x„.) е *P; c_к е C, (/; j; к е N)* и любого означивания *m: X -> M.*

Условия 1)-3) сформулированы в вышеприведенном виде, в связи с тем, что такая форма их записи даёт базис индукции в доказательстве теоремы об абстрактности свойств алгебраических систем, выразимых ми тыке исчисления предикатов сигнатуры *а*. Обычно эти условия

формулируются следующим образом:

Г) *ФГГПа^а,;...;^))* = ^'"(Ф**(а,)**;Ф**(а₂)**;...;Ф(^));

) М |= $^{9}P'''$ **(а,;а₂;...;а_{я/}) <=> Nf=** $^{V}P'''(Ф(a_1);Ф(a_1);...;Ф(a_{m/}У)$

для любых $a_x;a_г;...;a_{m1}$ *е М;*

У) Ф(%)=Ч

для любых *F'"*(x,;x₂;...;x_{nj})е*F*; P''^{J} *(x₁;x₂;...;x_n)*е*P ; c_к е C, (i;j;keN).*

Нетрудно видеть, что отношение изоморфизма, как бинарное отношение на совокупности всех алгебраических систем сигнатуры *а* является рефлексивным, симметричным и транзитивным, то есть отношением эквивалентности. В соответствии с этим, все алгебраические системы данной сигнатуры *а* разбиваются* на классы изоморфных, между собой, систем. Получающиеся в результате этого разбиения классы, называются классами изоморфизма.

Если в определении изоморфизма не требовать биективности отображения Ф то есть считать, что Ф - произвольное отображение *М* в *N* и в условии 2) вместо эквивалентности <=> записать импликацию => , то получим определение гомоморфного отображения системы *М* в систему *N* или гомоморфизма изМвЖ

Из приведенного определения следует, что всякий изоморфизм является гомоморфизмом. В связи с этим, предоставляется возможность создания следующей проблемной ситуации: при каких условиях гомоморфное отображение *ф-.М^N* является изоморфизмом? На пути разрешения этой проблемы последовательно устанавливается, что:

- отображение Ф должно быть сюрьективным;
- соответствие Ф"[1] *ж Мв N* должно быть отображением, то есть с учетом первого утверждения, отображение Ф должно быть биективным;
- отображение Φ^{-1} должно быть гомоморфизмом.

Исходя из этих положений, получаем следующую характеризацию отношения изоморфизма: биективное отображение *Ф:М-^-Ы* основного множества *М* алгебраической системы *М = (hi; [v]<j]* на основное множество *N* алгебраической системы *N = (N; [v]<J}*) является изоморфизмом *М* на *N* тогда и только тогда, когда Ф и Ф"[1] являются гомоморфизмами системы *М* на N и системы 7V на М соответственно. В связи с тем, что условие 2) определения гомоморфизма:

$$M\backslash = {}^{,p}P_J^{mj}(a_l;a_2;...;a_{mj}) => N\wedge Tp\ (\Phi(a_x y,\Phi(a_2);...;\Phi(a_m,))$$

любых $a_1;a_2',...',a_m$ е М носит импликативный характер, общее понятие гомоморфизма может конкретизироваться следующим образом:

- гомоморфизм *Ф: М -> N* называется сильным гомоморфизмом, если для любого предикатного символа *Рр* е *Р*, для любых элементов $a_l;a_2;...;a_{nij}$ е М выполняется условие:

$$N\backslash = \wedge(\Phi(«,);\Phi(«_2);...;\Phi(«_{Иу})) => M(= *Pp\{a[;a_2;...;<,)$$

для некоторых ef;aj ;...;«£ бЛ/ таких, что) = ;

Ф(a_2) = Ф(я$_2$) ; ...; *Ф{$a_{Шj}$}* = <Р(а^), то есть из истинности в *N* предиката [41] *Рр* на образах следует истинность соответствующего предиката *'Рр* в *М* на некоторых прообразах; '

- гомоморфизм *Ф-.М^N* называется строгим гомоморфизмом, если для любого предикатного символа *Рр* е *Р*, для любых элементов о,;a_2;...;а е *М*выполняется условие:

$$M\backslash = *Pp\{\Phi\{_{a1}У,\Phi\{a_г);...;\Phi\{\%))\wedge M\wedge\ 'Pp\ (a>;a_2;..-a_m),$$

то есть из истинности в ./V предиката [¥]*Рр* на образах следует истинность в *М* соответствующего предиката [9]*Рр* на любых прообразах.

Из этих определений, с очевидностью, вытекает, что всякий строгий гомоморфизм является сильным.

Дифференцированному уяснению всех этих понятий способствует самостоятельное построение простейших примеров, показывающих, что существуют:

- биективные гомоморфизмы, то есть гомоморфизмы биективно отображающие основное множество первой системы на основное множество второй системы, не являющиеся изомоморфизмами;
- гомоморфизмы, не являющиеся ни строгими, ни сильными;
- гомоморфизмы, являющиеся сильными, но не строгими.

С целью демонстрации технологических подходов к построению подобных примеров дадим пример сильного гомоморфизма, не являющегося строгим. Для построения подобных примеров обычно достаточно рассмотрения конечных моделей конечных предикатных сигнатур.

Пусть <г = (/>,'); М = <Ц;a_2;a_3};^[1]) и *N = Щ^Щ* - модели сигнатуры *а*, при этом на множестве *М* = {а,;a_2;a_3} предикат [9]*Р}* определен по правилам:

"$P_{1'}$(o_1)=и; ^'(«$_2$) = л ;

а на множестве *Ы = {Ъ^Ъ$_2$}* - соответствующий предикат [W]*Р* по правилам:

»7?(*у) = и; ^V(*$_2$) = к .

Отображение *Ф : М ->N* определим следующим образом: ***Ф(а$_1$) = Ь$_1$, Ф(а$_2$) = Ъ$_1$ 1 *(<»,) = V***

Непосредственная проверка показывает, что Φ - сильный гомоморфизм M на N.

Тем не менее, строгим он не является, так как для образа , на котором предикат $^{¥}P\{$ принимает истинное значение, имеется прообраз a_2, на котором соответствующий предикат принимает ложное значение.

Докажем теперь теорему, показывающую, что структурные свойства алгебраических систем сигнатуры a, допускающие запись посредством формул языка исчисления предикатов этой сигнатуры сохраняются при изоморфизме, то есть являются одними и теми же у всех систем любого класса изоморфизма.

Эта теорема формулируется следующим образом: пусть Φ - изоморфизм алгебраической системы **Af,** $=\{M^{\wedge}$, $*(T)$ на алгебраическую систему M_2 =(M$_2$; 'о); A - $A(x^{\wedge};x_2;...;x_n)$ - произвольная формула сигнатуры сг и г - любое означивание множества предметных переменных X в М,. Тогда

$$M^{\wedge}A[m] < ^{\wedge}M_2 \setminus = A[m \circ \phi]. \qquad (1)$$

Проведем доказательство этого утверждения обобщенно индуктивным методом.

а) Если формула A является минимальным элементом частично упорядоченного множества <1$_{ст}$;ч), то:

а-1) $Л$ = (?, (х,;х$_2$;х„) = t_2 (х,;x_2;x_n)) для некоторых термов =^(х$_i$;х$_2$;...;х„) и t_2=$t_2(x_x;x_2;...;x_n)$ из $\Gamma_{ст}$ или

а.2) $A = Pp(t_1;t_2;...;t_{mj})$ для некоторого предикатного символа Pp εP и некоторых термов $t_x;t_2;...;t_m$ <$вT_a$.

В случае а.1) для доказательства эквивалентности (1) воспользуемся равенством Ф(? [г]) = t [m о Φ], доказательство которого приводится в параграфе 5.16 (смотри равенство (3)). Это равенство имеет место для любого гомоморфизма, а, следовательно, и для любого изоморфизма системы M_x в систему M_2,' для любого терма t е T_a и любого означивания $г$: X -> M_x. В содержательном плане это равенство утверждает, что образ значения терма t в модели Afj при означивании m при гомоморфизме Φ равен значению этого терма в модели M_2 при означивании $г$ о $ф$. Таким образом, в рассматриваемой ситуации будем иметь:

$$0(t_x\ [!\blacksquare]) = /,\ [m \circ \phi] - \qquad (2)$$

$$^{\wedge}M='Лг»\Phi]. \qquad (3)$$

Пусть теперь M_x **Н** Л [г], то есть значения и $t_2[r]$ термов ^ и f$_2$, соответственно, в модели M_x равны. Так как Φ - отображение множества M_x в множество M_2 $и$ $ф$]; Г$_2$[г]бМ,, то из равенства AW$^{=}$^W следует, **что**

$$\Phi(/,[г])=\Phi(/_2[г]). \qquad (4)$$

Из (4), с учетом равенств (2) и (3), далее, получаем t_x [г°Ф] = г$_2$ [г о $ф$]$_г$ то есть, что $M_2\setminus=A[m°\Phi]$.

В случае а.2) эквивалентность (1) представляет собой условие 2) определения изоморфизма.

б) Если формула A не является минимальным элементом частично упорядоченного множества $(L_a; <)$, то:

6.1) $A = B_X \& B_2;$

6.2) $A = B_X \vee B_2;$

6.3) A = **В,** ~> $B_2;$

6.4) $A = -\setminus B;$

6.5) Л = (Зх)D(x) ;

6.6) A = (Vx)C(x),

то есть формулы J3; = **В**.(х,;х$_2$;...;х„); **В** = 2?(х,;х$_2$;...;х„); $D(x)$ = D(x;x$_1$;x$_2$;...;х„) (/' = 1;2) строго предшествуют формуле Л, в смысле порядкового отношения < .

Таким образом, в дальнейших рассуждениях можно предполагать, что для формул B_x; B_2; B и $\boldsymbol{D}(x)$ эквивалентность (1) уже доказана.

Для примера докажем эквивалентность (1) в случаях 6.1) и 6.5) синтаксического строения формулы A, оставляя случаи 6.2) - 6.4) и 6.6) студентам для самостоятельного рассмотрения.

6.1) Пусть M_x $(= A[m]$ имеет место. Тогда, в соответствии с положениями истинностной семантики, отсюда следует, $что$ $M_x\setminus = B_x[m]$ и M_x |= B_2 [г], то есть с учетом предположений относительно формул B_x и B_2, что

$$M_2 \setminus = B_x[m°\Phi] \text{ и } M_2 \setminus -B_2\ [m°\Phi]. \qquad (5)$$

Из (5) вновь по правилам истинностной семантики получаем, что M_2 **И** $B_x\,[m\ o\ \Phi]\ \&\ B_2\,[m o \phi]$, то есть $M_2 \setminus = A[m \circ \Phi]$.

6.5) Пусть $M_X \setminus = A\ [m]$, то есть М, |= (Эх) $'^{p}D(x;T(x_1);T(x_2);\ldots;T(x_{\ldots}))$ ■ В соответствии с определением истинностного значения формулы $A = (3x)D(x)$ в истинностной семантике, отсюда получаем, что существует означивание m_x: X -» M_x такое, что

г,/$X \setminus \{x\} = m$ и М, |= $''0\{r_x\{x)\setminus r_x\{x_x);\ldots;\phi_{\ldots}))$. (6)

Из (6), с учетом предположения относительно формулы $D(x)$ получаем

M_2 **1=** "Д((г, оФХх);(г, $o\phi)(x_x);\ldots;(m_x \circ \Phi)(x_{\ldots}))$, то есть

M_2 |= $*D((r_x$ оф)(х);(гоФХх$_1$);...;(тоф)(х$_я$))

или по правилам истинностной семантики: M_2 | =(Эх) $*B(x;(m o \phi)(x,y,..-(m\circ\Phi)(x_{\ldots}))$, то есть M_2 (= A $[m \circ \Phi]$.

Фундаментальная значимость этой теоремы для математики заключается в том, что она является одним из веских аргументов, обеспечивших принятие концепции изучения алгебраических систем с точностью до изоморфизма.

Теорема о гомоморфизмах для алгебраических систем формулируется следующим образом: если Φ: **М, -»** M_2 - гомоморфизм системы **Л/,** на систему M_2, то отображение Φ/P_ϕ является гомоморфизмом фактор-системы M_x/P_ϕ на систему M_2, при этом, если Φ - сильный гомоморфизм, то гомоморфизм Φ/P_ϕ будет изомофизмом.

Для доказательства первого утверждения теоремы достаточно доказать, что для отображения Φ/P_ϕ выполняются условия 1) - 3) определения гомоморфизма.

1) Пусть $F'''\ e F$ и $a_x;a_2;\ldots;a_{ni}$ еМ, (/' б $N)$, тогда

$\Phi/P\Phi$ $й'РГ\ i\hat{}a_2;\ldots;a_{ni}\ )]_P J$= <Р(«7^ (я,;^;...;^)) =

$\hat{}P;'(\Phi(a_1 У_> \Phi(a_2);\ldots;\Phi(cr_{\ldots 1})) =$

$= \hat{};'(\Phi/P_\phi([a\hat{};\Phi/P_\phi(\{a_2]_P\hat{};\ldots-\Phi/P_\phi([a_n X))$ ■

Равенство (1) этой цепочки обусловлено определением интерпретации *г)* при построении фактор-системы **Л/,** $/P_\phi$; равенство (2) - правилом (9) определения отображения Φ/P_ϕ ; равенство (3) - условием 1) определения гомоморфизма Φ ; равенство (4) - правилом (9).

Равенство первого и последнего звена этой цепочки равенств даёт выполнение требуемого условия.

2) Пусть eP и $[S]_a = ([S]_a;{}^9 a)$ $M = \{M;$ **V>** $Sc z M$ и

М, $1P_\Phi$ |= $^n P?\{[a\hat{}[a_2\setminus_p$—$Aa_{mj}\setminus)$ (Ю)

Из (10) по определению интерпретации *г/* следует существование элементов $b_l;b_2;\ldots;b_{nij}$ **еЛ/,** таких, что Ь, $e[a_{\{}]_{P\phi}$ (i = 1;2;...;»»у) и М, (= $^v Pp\phi_x;Ь_2;\ldots;Ь_m$) . Так как Φ - гомоморфизм M_x на M_2, то

м$_2$ и (Ф(^;Ф(6$_2$);...;Ф(6)) (И)

из (11) с учетом равенства (9) получаем

$M_г$ И $*\phi$ (ft],.);<Р/Р, $\{[Ь_2\setminus);\ldots;\Phi/P_\Phi$ ([А$_{Я/}$)). (12)

Изтого,что 6, е [а,]$_{Рф}$ следует, что $Щ\ ] = [a,,]_{P\phi}$ $(i = 1;2;\ldots;/Wy)$.

Отсюда с учетом (12) следует:

$м_2$ и $''Р?\ ({}^{\phi}/Ъ(KU(fak);-;ф/\hat{}(К,\setminus))\bullet$ (12')

Доказав, что импликация (10) => (12') имеет место, получаем, что второе условие определения гомоморфизма для отображения Φ/P_ϕ

также выполняется.

3) Пусть $c_к$ е С $(k e N)$. Тогда $\Phi/P_\phi\ Cc_к) =$ $\Phi/P_\phi(Гc_к]_{P\phi}) = <Р(\%) = *c_к$,

что доказывает третье условие определения гомоморфизма для отображения $\Phi/Р\phi$ ■

Пусть теперь гомоморфизм Φ: M_x —> M_2 является сильным. Докажем, что в этом случае, гомоморфизм Φ/P_ϕ будет изоморфизмом.

Для этого осталось показать, что отображение Φ/P_ϕ биективно и что импликация условия 2) определения гомоморфизма - обратима.

Действительно, рассуждая методом от противного, предположим, чт0 K]pф * **[a2]Рф** > н0 тем не менее> $^{\phi}1^{P}\Phi$ **(1**<*\|p_ϕ) = $^{\phi}/^{p}\phi$**({a2]**p_ϕ **)** • В соответствии с правилом (9), определения соответствия Φ/P_ϕ , отсюда следует, что $\Phi(a_x) = \Phi(a_2)$, то есть $a_x P_\phi a_2$, что влечёт совпадение классов **[ax]Рф** и **[a2]Рф** , противоречащее условию **[ax]Рф *[a2]Рф-**

Для доказательства обратимости импликации в условии 2) определения гомоморфизма (смотри параграф 5.14) предположим, что

$M_г$ И *P**?(Ф1**P_ϕ{{аДгф)ДФ1**P_ϕ**(1a**$_г$**\);..-Ф1РЛ%\))** (13)

для предикатного символа $Pp\,e\,P$ $(j$ е $N)$ и элементов

^{e}M-

С учетом (9) из (13) получаем

M_2 И ·"Pp(Ф(a_1 У>Ф(a_2);...;Ф(^)) (14)

По условию, Φ - сильный гомоморфизм. В связи с этим, исходя из (14) находим элементы е/ц такие, что $a_i P_0 b_i$ $(i = l;2;...;m_j)$ и $M_X\backslash=^{V}P^{TM}\{b_x;b_2;...;b_{mi}\}$. Отсюда, по правилам построения фактор-системы M_x/P_ϕ следует, что

м,**/**p_ϕ и **"p;>mpAWi"№**$_{mj}$**}pJ** (is)

И, наконец, из (15), в связи с равенствами $[a_к]_{Рф} = [Ь_к]_{Рф}$ $(к$ — 1;2;...; m Л , получаем

M_X/P_Φ **(= '^(h]**$_v$**K]**$_f$**J;-;k,]**$_4$**)** (Щ

Доказав импликацию (13) => (16), мы доказали тем самым, требуемую обратимость импликации из второго условия определения гомоморфизма, что завершает доказательство теоремы.

Гомоморфизм Φ/P_ϕ называется **каноническим.**

Изоморфное отображение Φ системы M_x на подсистему системы M_2 называется изоморфным вложением М, в M_2.

Из теоремы о гомоморфизмах, в качестве простого следствия, получаем следующее утверждение: если Φ гомоморфизм системы М, в систему M_2, то для Φ существует единственное представление в виде композиции

$\Phi = {}_{£p}\circ\Phi/P_\phi o 1$,

где

£Р - естественный гомоморфизм M_x на M_x/P_ϕ ; Φ/P_ϕ - канонический гомоморфизм M_x/P_ϕ на $\Phi(M_X)$; I - изоморфное вложение $\Phi(M_X)$ в M_2.

Тема 7 Алгоритмы построения синтаксической составляющей формализованных языков

Основой глубокого понимания знаний в области программирования является формальный аппарат, основанный на четких правилах оперирования с ними, т.е. в общем случае - логическое исчисление. Логическое исчисление определяется такими составляющими: алфавит, множество формул, множество аксиом, совокупность правил вывода.

Логическое исчисление строится на базе некоторого формализованного языка. Построение формализованного языка осуществляется следующим образом (по А.Черчу):

1) выписываются единые неделимые исходные символы, которые будут употребляться в языке (алфавит);

2) конечная линейная последовательность исходных символов образуют формулу (множество формул);

3) из числа всех формул по определенным правилам выделяются правильно построенные формулы, из которых некоторые являются аксиомами (множество аксиом);

4) устанавливаются правила вывода, по которым из соответствующих правильно построенных формул как из посылок непосредственно следует как заключение некоторая правильно построенная формула (совокупность правил вывода).

Построенный формализованный язык должен отвечать требованиям эффективности:

- должен существовать метод, позволяющий эффективно определить, является ли любой данный символ одним из исходных символов (алфавита) или нет;

- должен существовать метод, позволяющий эффективно определить, является ли любая данная формула (построенное из символов алфавита) правильно построенной или нет;
- должен существовать метод, позволяющий эффективно определить, является ли любая данная формула (построенное из символов алфавита) одной из аксиом или нет;
- должен существовать метод, позволяющий эффективно определить, выводится ли любая данная правильно построенная формула, как заключение, из некоторых данных правильно построенных формул, как из посылок, или нет.

Правила пользования формализованным языком должны быть изложены на языке, хорошо понятном человеку (такой язык называется метаязыком).

Формализованный язык является основой языков программирования. Формально, язык программирования – это открытое для пополнения множество текстов, записанных с помощью некоторого набора символов (алфавита языка). Структура любого языка (естественного, формализованного, программирования) предполагает выделение синтаксической и семантической составляющих.

Синтаксис формализованного языка состоит из системы правил построения выражений этого языка и проверки того, являются ли эти выражения правильно построенными формулами, аксиомами, теоремами, выводами или доказательствами.

Синтаксис языка программирования – это совокупность требований, которым должна удовлетворять любая осмысленная программа (правила образования текстов). Синтаксический разбор программы осуществляется путем указания из каких составных частей и каким способом могут быть построены программы, определить, какие именно последовательности символов считаются программами данного языка программирования.

В процессе ознакомления с методологией построения формализованного языка, изучения его синтаксических особенностей и выразительных возможностей студенты познают специфику индуктивных определений и их роль в обеспечении алгоритмической природы определяемых объектов.

В частности, индуктивный анализ синтаксического строения слов (последовательностей слов) в базовых исчислениях (исчисление высказываний и исчисление предикатов) дает:

алгоритм определения, по данному слову, является ли это слово формулой;

алгоритм выписывания всех подформул данной формулы;

алгоритм выяснения, по конечной последовательности формул, является ли эта последовательность выводом.

Одной из важнейших задач при изучении этих и им подобных алгоритмов синтаксического характера является выявление основных механизмов, обеспечивающих наличие разрешающих процедур. Именно на этом пути происходит осознание алгоритмической природы синтаксических конструкций.

С интуитивной точки зрения под алгоритмом для некоторого (потенциально бесконечного) класса однотипных задач понимается точное предписание, состоящее из конечного числа определенных инструкций, исполнение которых приводит к решению любой задачи из этого класса.

Изучение алгебры знакомит студентов с широким спектром примеров классических алгоритмов: алгоритмом деления с остатком, алгоритмом Евклида, алгоритмом Эратосфена, алгоритмом разложения натурального числа $n \neq 1$ в произведение простых чисел, алгоритмом Гаусса решения систем линейных уравнений, алгоритмом Штурма отделения действительных корней многочленов с действительными коэффициентами, алгоритмом Горнера деления многочлена на двучлен $x - x_0$, алгоритмом параллельного вычисления ядра и базиса линейного оператора и т.д., в которых, в наиболее яркой форме воплотились основные черты, характерные для интуитивных представлений об алгоритмах, как вычислительных процедурах чисто механического характера :

алгоритм задается как набор инструкций конечных размеров;

имеется вычислитель (обычно человек), который умеет обращаться с инструкциями и производить вычисления;

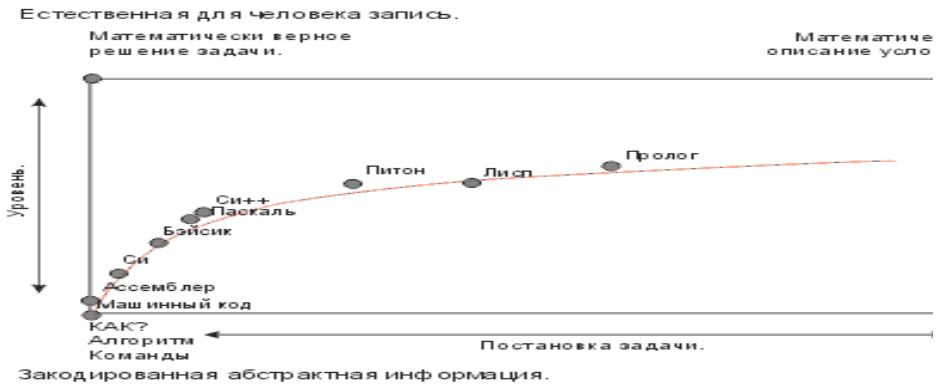
алгоритм применяется к решению однотипных задач из потенциально бесконечного класса;

вычислитель взаимодействует с набором инструкций так, что вычисление осуществляется по шагам в дискретном времени;

в начальный момент времени задается исходная система величин, а в каждый последующий момент новая (промежуточная) система величин однозначно определяется результатами, полученными в предшествующие моменты времени на предшествующих шагах;

закон получения последующих величин из предшествующих является простым и понятным и не требует обращения к другим методам или устройствам.

Кроме этого, следует отметить: в связи с тем, что алгоритм описывается конечным набором инструкций конечного размера, а число шагов вычисления не ограничено, то, как правило, в алгоритмах дается общее описание одного или нескольких шагов, которые в процессе исполнения алгоритма могут многократно (циклически) повторяться. Для организации пошагового процесса вычисления, алгоритм должен также содержать информацию о том, как начинается этот процесс, как осуществляется переход от одного шага работы к другому и что считать результатом исполнения алгоритма.



Языки программирования относятся к группе формальных языков, для которых в отличие от естественных языков однозначно определены синтаксис и семантика. Описание синтаксиса языка включает определение алфавита и правил построения различных конструкций языка из символов алфавита и более простых конструкций. Для этого обычно используют форму Бэку-са-Наура (БНФ) или синтаксические диаграммы. Описание конструкции в БНФ состоит из символов алфавита языка, названий более простых конструкций и двух специальных знаков:

«::=» - читается как «может быть заменено на», «|» - читается как «или».

При этом символы алфавита языка, которые часто называют терминальными символами или терминалами, записывают в неизменном виде. Названия конструкций языка (нетерминальные символы или нетерминалы), определяемых через некоторые другие символы, при записи заключают в угловые скобки («< », « >»). Например, правила построения конструкции <Целое>, записанные в БНФ, могут выглядеть следующим образом:

<Целое> ::= <Знак> <Целое без знака> | <Целое без знака>

<Целое без знака> ::= <Целое без знака> <Цифра> | <Цифра>

<Цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Знак> ::= + | -

Для отображения того, что конструкция <Целое без знака> может включать неограниченное количество цифр, использовано правило с левосторонней рекурсией. Многократное применение этого правила позволяет построить целое число с любым количеством цифр.

Синтаксические диаграммы отображают правила построения конструкций в более наглядной форме. На такой диаграмме символы алфавита изображают блоками в овальных рамках, названия конструкций - в прямоугольных, а правила построения конструкций - в виде линий со стрелками на концах. При этом, если линия входит в блок, то в описываемую конструкцию должен входить соответствующий символ. Разветвление линии означает, что при построении конструкции возможны варианты.

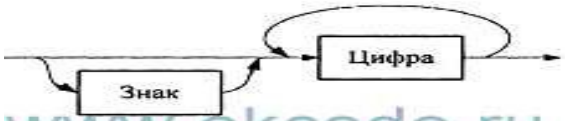


Рис. 2.1. Синтаксическая диаграмма конструкции <Целое>

На рис. 2.1 представлена синтаксическая диаграмма, иллюстрирующая первые два правила описания конструкции <Целое>. Из диаграммы видно, что целое число может быть записано со знаком или без и включать произвольное количество цифр. Для описания синтаксических конструкций своего языка Н. Вирт использовал именно синтаксические диаграммы, поэтому в тех случаях, когда словесное описание синтаксиса конструкции длинно и нечетко, мы будем использовать синтаксические диаграммы.

Из символов алфавита в соответствии с правилами синтаксиса строят различные конструкции. Простейшей из них является конструкция <Идентификатор>. Эта конструкция используется во многих более сложных конструкциях для обозначения имен программных объектов (полей данных, процедур, функций и т. п.). В Borland Pascal идентификатор представляет собой последовательность букв латинского алфавита (включая символ подчеркивания) и цифр, которая обязательно начинается с буквы, например: аааа, ЫЗ1, Parametral, _а и т. п. Синтаксическая диаграмма идентификатора приведена на рис. 2.2. Остальные конструкции будут рассмотрены в последующих разделах.

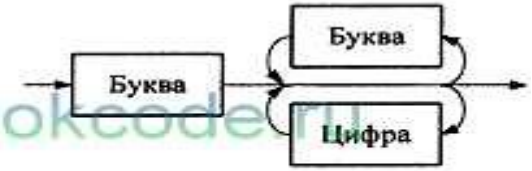


Рис. 2.2. Синтаксическая диаграмма <Идентификатор>

Тема 8 Технологий семантического анализа языков программирования

1. Понятие семантики.
2. Методы описания семантики.
3. Интерпретатор и компилятор.

I. Понятие семантики.

Семантика языка программирования – это правила, определяющие, какие операции и в какой последовательности должна исполнить машина, работая по произвольной заданной ей программе (правила истолкования текстов тем, кому они адресованы).

С теоретической точки зрения формальный язык это произвольное множество строк конечной длины (рис. 1).

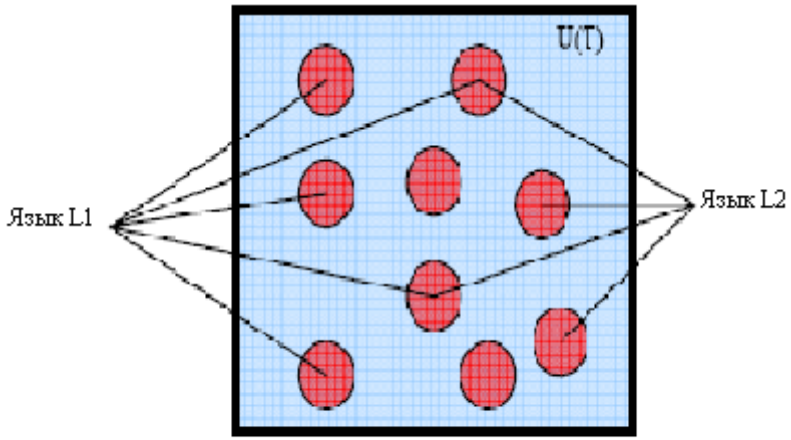


Рис. 1. Формальный язык.

Для описания семантики формального языка необходимо задать интерпретирующее отображение его строк в элементы некоторой модели, возможно в строки другого языка (рис. 2). В основе описания семантики формальных языков лежит принцип композиционности Г. Фреге, заключающийся в определении семантики целого через семантику частей, т.е. между семантическими правилами образования выражений языка устанавливается соответствие. Синтаксис формальных языков традиционно задается формальными грамматиками.

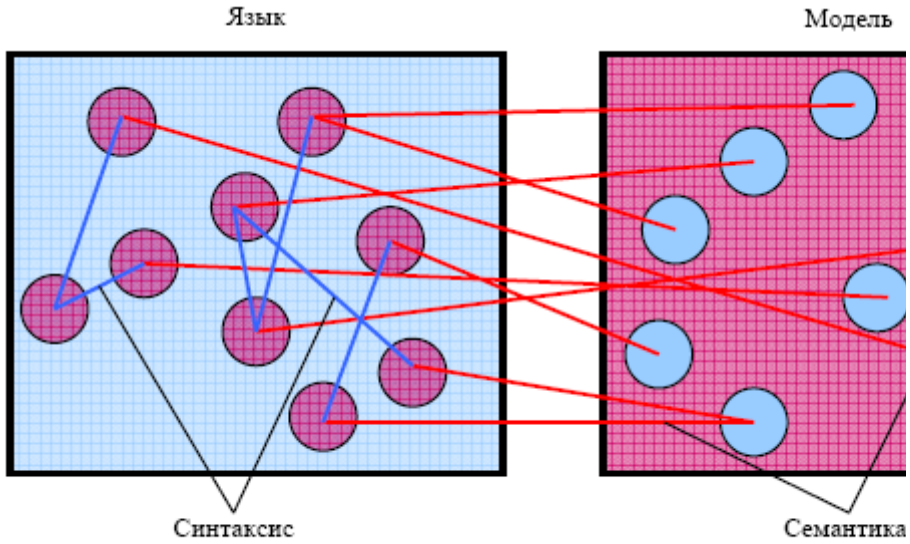


Рис. 2. Синтаксис и семантика

Задача формального определения семантики рассматривается теоретиками так же долго, как и задача определения синтаксиса, но до сих пор удовлетворительного решения не найдено. Для описания семантики формальных языков разработано множество различных моделей и методов (таблица 1).

Таблица 1 – Модели для описания семантик формальных языков.

Модели		
модели спецификаций, в которых описываются отношения между различными функциями языка и, если программа реализует эти отношения, то она корректна по отношению к спецификации	*аппликативные модели*, в которых непосредственно конструируется определение функции, которую вычисляет каждая программа, написанная на этом языке	*грамматические модели*, основанные на добавлении расширений к грамматике, определяющей этот.

Во всех перечисленных методах описание семантики осуществляется путем выделения некоторой типологии смыслов – категорий значений (семантических категорий, примитивов), через которые и посредством которых на некотором семантическом языке задается более сложный смысл.

II. Методы описания семантики.

При описании семантики языков программирования упоминаются три наиболее распространенных метода:

1. ***Операционный*** (смысл конструкций языка в терминах переходов абстрактной машины из одного состояния в другое), например, SECD-машина П. Лендина;
2. ***Аксиоматический*** (смысл конструкций языка в терминах множества формул, описывающих состояние объектов программы), например, аксиоматический метод Хоара и метод индуктивных утверждений Флойда;
3. ***Денотационные*** (смысл конструкций языка в терминах абстракции функций на состояниях программы), например, теория семантических доменов Д. Скотта.

2.1 Операционный метод описания семантики.

В 60-х годах, уже в эпоху высокоуровневых языков программирования, П. Лендином (Peter J. Landin) была разработана так называемая SECD-машина, а именно, математическая формализация для вычисления ламбда-выражений.
При этом SECD-машина была предназначена для редукции ламбда-выражений и использовала механизм передачи параметров при вызове функции по значению (call-by-value), в отличие от других типов передачи параметра (например, по имени или call-by-name).

Свое название SECD-машина получила от аббревиатур имен своих основных четырех элементов, а именно:

- Stack – стек, т.е. последовательность атомарных лямбда-выражений (переменных и констант), а также лямбда-абстракций;
- Environment – среда, т.е. хранилище значений, которые связываются с переменными в ходе вычислений;
- Control – управление, т.е. последовательность «инструкций», управляющих работой SECD-машины;
- Dump – дамп, т.е. хранилище состояния SECD-машины (обычно пуст либо содержит прежнее состояние).

Именно перечисленная четверка элементов в полной мере характеризует состояние SECD-машины в произвольный момент вычислений.

Своим названием *категориальная абстрактная машина* обязана тому обстоятельству, что множество ее возможных *состояний* принадлежит пространству так называемых *декартово замкнутых категорий* (или, сокращенно, *д.з.к.*). Формальная система с *декартово замкнутыми категориями* должна удовлетворять следующим условиям:

1. определена *функция тождества* или тождественное преобразование (имеющее в комбинаторной логике аналог в форме комбинатора тождества I);
2. определена *операция композиции* или построения сложной функции (имеющая в комбинаторной логике аналог в форме комбинатора тождества B);
3. определена операция образования упорядоченной пары объектов < . , . >;
4. определена операция взятия первого элемента из упорядоченной пары объектов;
5. определена операция взятия второго элемента из упорядоченной пары объектов;
6. определена операция преобразования терма из алгебраической формы в аппликативную;
7. определена операция *аппликации* или применения функции к аргументу.

Проанализируем смысл сформулированных условий для категорий. Условия существования тождественного преобразования и построения сложных функций являются основополагающими и необходимы для формирования категорий произвольного типа. Условия существования операции образования упорядоченной пары объектов, а также операций взятия первого и второго элементов из упорядоченной пары необходимы для построения декартовых категорий, т.е. категорий, оснащенных операцией прямого (или, иначе, декартова) произведения и соответствующих ему *проекций*. Условия существования операций *аппликации* и преобразования терма из алгебраической формы в *аппликативную* необходимы для формирования *декартово замкнутых категорий*.

2.2 Аксиоматический метод описания семантики.

Логика Хоара — формальная система с набором логических правил, предназначенных для доказательства корректности компьютерных программ. Была предложена в 1969 году английским учёным в области информатики и математической логики Хоаром, позже развита самим Хоаром и другими исследователями. Первоначальная идея была предложена в работе Флойда, который опубликовал похожую систему в применении к блок-схемам.

Таблица 2. Аксиоматический метод описания семантики.

Аксиоматический метод описания семантики		
Тройки Хоара	Частичная и полная корректность	Правила (см. табл. 3)
$\{P\}\ C\ \{Q\}$ где *P* и *Q* являются *утверждениями*, а *C* — *командой*. *P* называется *предусловием*, а *Q* — *постусловием*.	если *P* имеет место до выполнения *C*, то либо имеет место *Q*, либо *C* никогда не завершится.	

Основной характеристикой **логики Хоара** является **тройка Хоара**. **Тройка** описывает, как выполнение фрагмента кода изменяет состояние вычисления. Тройка Хоара имеет следующий вид:

$$\{P\}\ C\ \{Q\}$$

где *P* и *Q* являются *утверждениями*, а *C* — *командой*. *P* называется *предусловием*, а *Q* — *постусловием*. Если предусловие выполняется, команда делает верным постусловие. *Утверждения* являются формулами логики предикатов.

В логике Хоара есть аксиомы и правила вывода для всех конструкций простого императивного языка программирования. В дополнение к этим конструкциям, описанным в оригинальной работе Хоара, Хоаром и другими исследователями были разработаны правила и для остальных конструкций: одновременного выполнения, вызова процедуры, перехода и указателя.

2. Частичная и полная корректность

В стандартной логике Хоара может быть доказана только частичная корректность, так как завершение программы нужно доказывать отдельно. Таким образом, «интуитивное» понимание тройки Хоара можно выразить так: если *P* имеет место до выполнения *C*, то либо имеет место *Q*, либо *C* никогда не завершится. Действительно, если *C* не завершается, никакого «после» нет, поэтому *Q* может быть любым утверждением. Более того, мы можем выбрать *Q* со значением «ложь», чтобы показать, что *C* никогда не завершится.

Полная корректность также может быть доказана с использованием расширенной версии правила для оператора *While*.

Таблица 3.

Правила	
Аксиома пустого оператора	$$\{P\}\ \mathbf{skip}\ \{P\}$$
Аксиома оператора присваивания	$$\{P[E/x]\}\ x := E\ \{P\}$$
Правило композиции	$$\frac{\{P\}\ S\ \{Q\}\ ,\ \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}$$
Правило условного оператора	$$\frac{\{B \wedge P\}\ S\ \{Q\}\ ,\ \{\neg B \wedge P\}\ T\ \{Q\}}{\{P\}\ \mathbf{if}\ B\ \mathbf{then}\ S\ \mathbf{else}\ T\ \mathbf{endif}\ \{Q\}}$$
Правило вывода	$$\frac{P' \rightarrow\ P\ ,\ \{P\}\ S\ \{Q\}\ ,\ Q \rightarrow\ Q'}{\{P'\}\ S\ \{Q'\}}$$
Правило оператора цикла	$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{done}\ \{\neg B \wedge P\}}$$
Правило оператора цикла с полной корректностью	$$\frac{<\ \text{is well-founded},\ [P \wedge B \wedge t = z]\ S\ [P \wedge t <}{[P]\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{done}\ [\neg B \wedge P]}$$

Аксиома пустого оператора

Правило для пустого оператора утверждает, что оператор **skip** (пустой оператор) не меняет состояния программы, поэтому утверждение, верное до **skip**, остаётся верным после его выполнения.

$$\{P\}\ \mathbf{skip}\ \{P\}$$

Аксиома оператора присваивания

Аксиома оператора присваивания утверждает, что после присваивания, значение любого предиката относительно правой части присваивания не меняется с заменой правой на левую часть:

$$\{P[E/x]\}\ x := E\ \{P\}$$

Здесь $P[E/x]$ означает выражение *P* в котором все вхождения свободной переменной *x* заменены выражением *E*.

Смысл аксиомы присваивания заключается в том, что истинность $\{P[E/x]\}$ эквивалентна $\{P\}$ после выполнения присваивания. Таким образом, если $\{P[E/x]\}$ имело значение «истина» до присваивания, согласно аксиоме присваивания $\{P\}$ будет иметь значение «истина» после присваивания. И наоборот, если $\{P[E/x]\}$ было равно «ложь» до оператора присваивания, $\{P\}$ должно быть равно «ложь» после.

Примеры корректных троек:

- $\{x + 1 = 43\}\ y := x + 1\ \{y = 43\}$

$$\{x+1 \le N\}\ x := x+1\ \{x \le N\}$$

Аксиома присваивания в формулировке Хоара *не применима*, когда более одного идентификатора ссылаются на одно и то же значение. Например,

$$\{y = 3\}\ x := 2\ \{y = 3\}$$

является неверным утверждением, если *x* и *y* ссылаются на одну и ту же переменную, так как никакое предусловие не может обеспечить, чтобы *y* было равно 3 после того, как *x* присвоено 2.

Правило композиции

Правило композиции Хоара применяется к последовательному выполнению программ *S* и *T*, где *S* выполняется до *T*, что записывается как *S;T*.

$$\frac{\{P\}\ S\ \{Q\}\ ,\ \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}$$

Например, рассмотрим два экземпляра аксиомы присваивания:

$$\{x+1 = 43\}\ y := x+1\ \{y = 43\}$$

и

$$\{y = 43\}\ z := y\ \{z = 43\}$$

Согласно правилу композиции, мы получаем:

$$\{x+1 = 43\}\ y := x+1; z := y\ \{z = 43\}$$

2.3 Денотационный метод описания семантики.

Теория вычислений Д. Скотта была создана до появления большинства современных языков программирования в конце 60-х годов. Эта теория позволяет произвести адекватную (а именно, полную и непротиворечивую) формализацию семантики языков программирования.

Теория вычислений Д. Скотта основана на фундаментальном понятии домена, который будем понимать как некоторый аналог множества, адекватно формализующий рекурсивно (т.е. на основе самоприменения) определенные функции и множества.

Для построения теории вычислений необходимо, во-первых, *перечислить* так называемые стандартные, или, точнее, наиболее часто используемые в рамках данной формализации, домены.

После перечисления стандартных доменов необходимо *определить* так называемые конечные домены, или, точнее, домены, элементы которых можно перечислить явным образом.

Наконец, после перечисления доменов перейдем к определению конструкторов доменов, под которыми понимаются операции построения новых доменов на основе имеющихся. Иначе говоря, *определим способы комбинирования* доменов.

Перечислив основные типы элементарных доменов, перейдем к их *комбинированию* посредством конструкторов.

Существует всего четыре типа конструкторов. Тем не менее, такой набор является вполне достаточным для построения домена, моделирующего семантику сколь угодно сложной предметной области или языка программирования.

Приведем определения перечисленных способов комбинирования доменов.

Под функциональным пространством из домена D_1 в домен D_2 будем понимать домен $[D_1 \to D_2]$, содержащий всевозможные функции с областью определения из домена D1 и областью значений из домена D_2:

$[D_1 \to D_2] = \{f \mid f : D_1 \to D_2\}$.

Под декартовым (или, иначе, прямым) произведением доменов D_1, D_2, ... D_n будем понимать домен всевозможных n-ок вида

$$[D_1 x D_2 x ... x D_n] = (d_1 x d_2 x ... x d_n) | d_1 \in D_1, d_2 \in D_2, d_n \in D_n ...,$$

.

Под последовательностью D* будем понимать домен всевозможных конечных последовательностей вида $d=(d_1,d_2,...,d_n)$ из элементов $d_1,d_2,...,d_n,...$ домена D, где n>0 .

Наконец, под дизъюнктной суммой будем понимать домен с определением

$$[D_1 + D_2 + ... + D_n] = (d_i, i) | d_i \in D_i, 0 < i < n+1,$$

где принадлежность элементов d_i компонентам D_i однозначно устанавливается специальными функциями принадлежности.

Тема 9 Формализация понятий «Доказательство», «Алгоритм», «Определимость» и «Эффективная вычислимость»

1. Понятие алгоритма.
2. Работа машины Тьюринга.
3. Эффективная вычислимость.

I. В подходах к определению понятия алгоритма можно выделить *три основных направления*.

Первое направление связано с уточнением понятия *эффективно вычислимой функции*. Этим занимались А.Черч, К.Гедель, С.Клини, определившие алгоритм как последовательность построения сложных функций из более простых. В результате был выделен класс так называемых *частично-рекурсивных функций*, имеющих строгое математическое определение. Анализ идей, приведших к этому классу функций, дал им возможность высказать гипотезу о том, что класс эффективно вычислимых функций совпадает с классом частично рекурсивных функций.

Второе направление связано с *машинной математикой*. Здесь сущность понятия алгоритма раскрывается путём рассмотрения процессов, осуществляемых в машине. Впервые это было сделано Тьюрингом, который предложил самую общую и вместе с тем самую простую *концепцию вычислительной машины*. Её описание было дано Тьюрингом в 1937 году. При этом Тьюринг исходил лишь из общей идеи работы машины как работы вычислителя, оперирующего в соответствии с некоторым строгим предписанием.

Третье направление связано с понятием нормальных алгоритмов, введённым и разработанным российским математиком А.А.Марковым. В рамках этого направления алгоритм рассматривается как конечный набор правил подстановок цепочек символов.

Для алгоритмических проблем типичным является то обстоятельство, что требуется найти алгоритм для решения задачи, в условия которой входят значения некоторой конечной системы целочисленных параметров $x_1, x_2,\ldots, x_n$, а искомым результатом также является целое число y. Следовательно, стоит вопрос о существовании алгоритма для вычисления значений числовой функции y, зависящей от целочисленных значений аргументов $x_1, x_2, \ldots, x_n$.

Определение 1. Функция называется *эффективно вычислимой*, если существует алгоритм, позволяющий вычислить ее значения.

Переход от алгоритма к эффективно вычислимой функции дает определенные преимущества. Дело в том, что все требования, которые предъявляются к алгоритмам, выполняются и для совокупности всех вычислимых функций, которая получила название совокупности рекурсивных функций.

Класс вычислимых функций можно построить следующим образом. Для начала выбираются *простейшие функции*:

1. $\square(x) = x + 1$ (оператор сдвига),
2. $O(x) = 0$ (оператор аннулирования),
3. $I_n^m(x_1, x_2,\ldots, x_n) = x_m$ (оператор проектирования).

Ясно, что все три простейшие функции всюду определены и интуитивно вычислимы.

II. Работа машины Тьюринга.

Работа машины полностью определяется заданием в первый (начальный) момент:

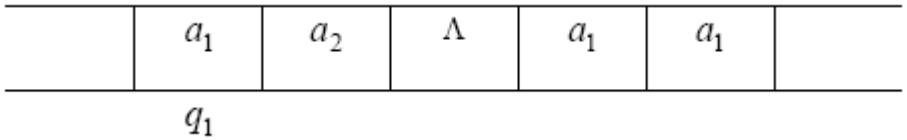
1. слова на ленте, т.е. последовательности символов, записанных в клетках ленты (слово получается чтением этих символов по клеткам ленты слева на право);
2. положения головки;
3. внутреннее состояние машины.

Совокупность этих трех условий (в данный момент) называется
конфигурацией (в данный момент). Обычно в начальный момент внутренним состоянием машины является q_1, а головка находится либо над первой слева, либо над первой справа клеткой ленты.

Заданное слово на ленте с начальным состоянием q_1 и положение головки над первым словом называется начальной конфигурацией. В противном случае говорят, что машина Тьюринга не применима к слову начальной конфигурации.

Другими словами, в начальный момент конфигурация представима в следующем виде: на ленте, состоящей из некоторого числа клеток, в каждой клетке записан один из символов внешнего алфавита А, головка находится на первой слева или первой справа клеткой ленты и внутренним состоянием машины является q_1. Получившееся в результате реализации этой команды слово на ленте и положение головки называется заключительной конфигурацией.

Например, если в начальный момент на ленте записано слово $a1\ \ a2a1a1$, то начальная конфигурация будет иметь вид:



(под клеткой, над которой находится головка, указывается внутреннее состояние машины).

Работа машины Тьюринга состоит в последовательном применении команд, причем, применение той или команды определяется текущей конфигурацией. Так в приведенном выше примере должна применится команда с левой частью q_1a_1 .

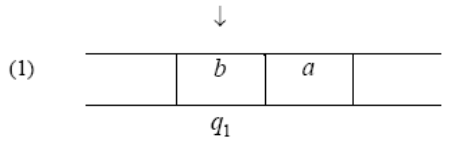
Итак, зная программу и задав начальную конфигурацию, полностью определяем работу машины над словом в начальной конфигурации.

Если в работе машины Тьюринга в некоторый момент t выполняется команда, правая часть которой содержит q_0 , то в такой момент работа машины считается законченной, и говорят, что машина применима к слову на ленте в начальной конфигурации. В самом деле, q_0 не встречается в левой части ни одной команды – этим и объясняется название q_0 «заключительное состояние». Результатом работы машины в таком случае считается слово, которое будет записано на ленте в заключительной конфигурации, т. е. в конфигурации, в которой внутреннее состояние машины есть q_0 . Если же в работе машины ни в один из моментов не реализуется команда с заключительным состоянием, то процесс вычисления будет бесконечным. В этом случае говорят, что машина не применима к слову на ленте в начальной конфигурации.

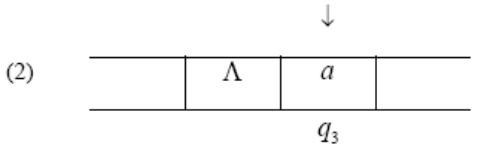
Пример 1. Построить машину Тьюринга T_1, которая применима ко всем словам с внешним алфавитом $\{a,b\}$ и делает следующее: любое слово $x_1x_2\ldots x_n$, где $x_i = a$ или $x_i = b$ (i =1,2,...n) преобразует в слово $x2...xn\ x1$, т. е., начиная работать при слове $x1x2...xn$ на ленте в начальной конфигурации, машина остановится, и в заключительной конфигурации на некотором участке ленты будет записано слово $x_2\ldots x_nx_1$, а все остальные клетки ленты (если такие будут) окажутся пустыми.

Решение. За внешний алфавит машины T_1 возьмем множество $A\ \ \{\ \ ,a,b\}$, а за внутренний – $Q\ \ \{q_0, q_1, q_2, q_3\}$. Команды определим следующим образом: $q_1a\ \ \ \Pi q_2$, $q_1b\ \ \ \Pi q_3$, $q_iy\ \ \ y\Pi\Pi_i$, где $y\in\{a,b\}$, i 2,3; , $q_2\ \ \ aHq_0$, $q_3\ \ \ bHq_0$

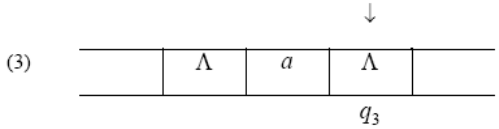
Рассмотрим работу машины $T1$ над словом ba . В работе машины над словом ba начальная конфигурация имеет следующий вид:



На первом шаге действует команда: $q_1b\ \ \ \Pi q_3$. В результате создается следующая конфигурация:

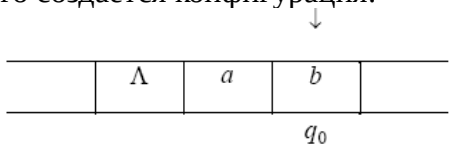


На втором шаге действует команда $q_3a\ \ \ a\Pi\Pi_3$, и на машине создается конфигурация



Наконец, третий шаг обусловлен командой $q_3 \quad bHq_0$. В результате чего создается конфигурация:

(4)



Эта конфигурация является заключительной, так как машина оказалась в состоянии остановки q_0.

Таким образом, слово *ba* переработано в слово *ab* .

Полученную последовательность конфигураций можно записать более коротким способом. Конфигурация (1) записывается в виде следующего слова в алфавите $A \cup Q$: $q_1 \quad ba$ (содержимое обозреваемой ячейки записано справа от состояния, в котором находится данный момент машина). Далее, конфигурация (2) записывается так: q_3a , конфигурация (3) – aq_3 , и наконец, (4) – abq_0. Вся последовательность записывается так: $q_1ba \Rightarrow \quad q_3a \Rightarrow aq_3 \quad \Rightarrow abq_0$.

3. Эффективная вычислимость.

Допустим что у вас есть следующий код программы:

1. A:=A*2;
2. B:=1;
3. X:=X+M[B];
4. B:=B+1;
5. Если B<100 то перейти на шаг 3.

Для эффективной работы программы нужно как можно большее её оптимизировать.

Здесь слабым местом является первая строка, потому что там используется умножение. Умножение всегда выполняется дольше, и если заменить его на сложения (A:=A+A) или еще лучше на сдвиг, то вы выиграете пару тактов процессорного времени. В этом коде используется цикл: «Пока B<100, будет выполняться операция X:X+M[B]». Это значит что процессору придется выполнить 100 переходов с шага 5 на шаг 3. Здесь у нас внутри цикла выполняется две строки: 3 и 4, если мы размножим их 2 раза:

1. B:=1;
2. X:=X+M[B];
3. B:=B+1;
4. X:=X+M[B];
5. B:=B+1;
6. Если B<50 то перейти на шаг 3.

Вторую и третью операцию мы повторили два раза. Это значит, что за один проход

цикла мы выполним два раза строки 3 и 4, и только после этого перейдем на строку 3, чтобы повторить операцию. Такой цикл уже нужно повторить толь 50 раз (потому что за один раз выполняется два действия). Это значит, что мы сэкономили 50 операций переходов. Это уже несколько сотен тактов процессорного времени.

А что, если внутри цикла написать строки 2 и 3 десять раз? Это значит что за один проход цикла строки 2 и 3 будут вычисляться 10 раз, чтобы получить в результате 100. А это уже экономия 90 операций переходов.

Недостаток этого подхода – увеличился код программы, зато повысилась скорость, и очень значительно. С одной стороны, увеличивается скорость, а с другой – увеличивается размер. А большой размер – это враг любой программы.

Тема 10 Модульный подход и рекурсивные методы решения задач (программирования)

Рекурсия -- это такой способ организации обработки данных, при котором программа вызывает сама себя непосредственно, либо с помощью других программ.

Итерация -- способ организации обработки данных, при котором определенные действия повторяются многократно, не приводя при этом к рекурсивным вызовам программ.

Математическая модель рекурсии заключается в вычислении рекурсивно определенной функции на множестве программных переменных. Примерами таких функций могут служить факториал числа и числа Фибоначчи. В каждом из этих случаев значение

функции для всех значений аргумента, начиная с некоторого, определяется через предыдущие значения.

Факториал $n!$ целого неотрицательного числа n задается следующими соотношениями:

$$0! = 1,$$

$$n! = n \cdot (n-1)! \quad \text{для} \quad n > 0.$$

Числами Фибоначчи f_n называют последовательность величин 0, 1, 1, 2, 3, 5, 8, ..., определяемую равенствами:

$$f_0 = 0,$$

$$f_1 = 1,$$

$$f_n = f_{n-1} + f_{n-2} \quad \text{для} \quad n > 1.$$

При анализе рекурсивной программы возникает, как обычно, два вопроса:

(а) почему программа заканчивает работу?
(б) почему она работает правильно, если заканчивает работу?

Для (б) достаточно проверить, что (содержащая рекурсивный вызов) программа работает правильно, предположив, что вызываемая ею одноименная программа работает правильно. В самом деле, в этом случае в цепочке рекурсивно вызываемых программ все программы работают правильно (убеждаемся в этом, идя от конца цепочки к началу).

Чтобы доказать (а), обычно проверяют, что с каждым рекурсивным вызовом значение какого-то параметра уменьшается, и это не может продолжаться бесконечно.

Математическая модель итерации сводится к повторению некоторого преобразования (отображения) $T\colon X \to X$ на множестве переменных программы X (прямом произведении множеств значений отдельных переменных). Программной реализацией итерации является обычно некоторый цикл, тело которого осуществляет преобразование T.

В качестве примера можно рассмотреть схему вычисления факториала натурального числа в соответствии с его другим определением: $n! = 1 \cdot 2 \cdot \ldots \cdot n$. При написании программы в соответствии с ним нужно работать с двумя величинами целого типа $\mathbb{Z}_M$: числом i, которое будет играть роль счетчика и изменяться от 1 до n включительно, и величиной k, в которой будет накапливаться произведение чисел от 1 до i.

Пространством X в данном случае будет $\mathbb{Z}_M^2$, в качестве начальной точки в этом пространстве возьмем точку $(1, 1)$ (что соответствует $i = k = 1$), а преобразование $T\colon X \to X$ будет иметь вид $T(i, k) = (i + 1, k * i)$. В случае, например, трехкратного применения преобразования T получим $T(T(T(1,1))) = T(T(2,1)) = T(3,2) = 6$, что обеспечит вычисление факториала числа 3.

Рекурсия и итерация взаимозаменяемы. Более точно, справедливо следующее утверждение.

Любой алгоритм, реализованный в рекурсивной форме, может быть переписан в итерационном виде, и наоборот.

1. Вводится натуральное число N. Вывести его факториал. (N! =1*2*3...N)
2. Вводится натуральное число N. Выводится N-е число Фибоначчи. (Числа Фибоначчи представляют собой последовательность: 0, 1, 1, 2, 3, 5, 8 ..... , т.е. каждое число, начиная с третьего, равняется сумме двух предыдущих.)

Тема 11 Модульный подход и рекурсивные методы решения задач (программирования)

Верификация и валидация являются видами деятельности, направленными на контроль качества программного обеспечения и обнаружение ошибок в нем. Имея общую цель, они отличаются источниками проверяемых в их ходе свойств, правил и ограничений, нарушение которых считается ошибкой. *Верификация* проверяет соответствие одних создаваемых в ходе разработки и сопровождения ПО артефактов другим, ранее созданным или используемым в качестве исходных данных, а также соответствие этих артефактов и процессов их разработки правилам и стандартам. В частности, верификация проверяет соответствие между нормами стандартов, описанием требований (техническим заданием) к ПО, проектными решениями, исходным кодом, пользовательской документацией и функционированием самого ПО. Кроме того, проверяется, что требования, проектные решения, документация и код оформлены в соответствии с нормами и стандартами, принятыми в данной стране, отрасли и организации при разработке ПО, а также — что при их создании выполнялись все указанные в стандартах операции, в нужной последовательности. Обнаруживаемые при верификации ошибки и дефекты являются расхождениями или противоречиями между несколькими из перечисленных документов, между документами и реальной работой программы, между нормами стандартов и реальным процессами разработки и сопровождения ПО. При этом принятие решения о том, какой именно документ подлежит исправлению (может быть, и оба) является отдельной задачей.

Валидация проверяет соответствие любых создаваемых или используемых в ходе разработки и сопровождения ПО артефактов нуждам и потребностям пользователей и заказчиков этого ПО, с учетом законов предметной области и ограничений контекста использования ПО. Эти нужды и потребности чаще всего не зафиксированы документально — при фиксации они превращаются в описание требований, один из артефактов процесса разработки ПО. Поэтому валидация является
менее формализованной деятельностью, чем верификация. Она всегда проводится с участием представителей заказчиков, пользователей, бизнес-аналитиков или экспертов в предметной области — тех, чье мнение можно считать достаточно хорошим выражением реальных нужд и потребностей пользователей, заказчиков и других заинтересованных лиц. Методы ее выполнения часто используют специфические техники выявления знаний и действительных потребностей участников. Различие между верификацией и валидацией проиллюстрировано на Рис. 1.

Рисунок 1. Соотношение верификации и валидации.

Приведенные определения получены некоторым расширением определений из стандарта IEEE 1012 на процессы верификации и валидации [12]. В стандартном словаре терминов программной инженерии IEEE 610.12 1990 года [13] определение верификации по смыслу примерно то же, а определение валидации несколько другое — там говорится, что валидация должна проверять соответствие полученного в результате разработки ПО исходным требованиям к нему. В этом случае валидация являлась бы частным случаем верификации, что нигде в литературе по программной инженерии не отмечается, поэтому, а также потому, что оно поправлено в IEEE 1012 2004 года, это

определение следует считать неточным. Частое использование фразы B. Boehm’a [14]: «Верификация отвечает на вопрос "Делаем ли мы продукт правильно?", а валидация — на вопрос "Делаем ли мы правильный продукт?"» также добавляет путаницы, поскольку афористичность этого высказывания, к сожалению, сочетается с двусмысленностью. Однако многочисленные труды его автора позволяют считать, что он подразумевал под верификацией и валидацией примерно те же понятия, которые определены выше. Указанные разночтения можно проследить и в содержании стандартов программной инженерии. Так, стандарт ISO 12207 [15] считает тестирование разновидностью валидации, но не верификации, что, по-видимому, является следствием использования неточного определения из стандартного словаря [13]. В среде исследователей, занимающихся теоретической информатикой (computer science), широко распространено более узкое понимание термина «верификация» — только как формальной верификации. В данном обзоре мы будем пользоваться более широким, инженерным понятием, хотя различные методы формальной верификации тоже будут рассмотрены.

1.2. Характеристики качества программного обеспечения

Поскольку основной задачей верификации, как и валидации, является контроль *качества программного обеспечения*, необходимо обратиться к этому понятию. Наиболее широко используется определение качества ПО в виде системы атрибутов или факторов, которые могут быть оценены с помощью ряда метрик. Такой подход позволяет конструктивно оценивать качество ПО в целом, во всех необходимых аспектах. Впервые он был сформулирован в 1977 году в работе МакКола с соавторами [16], затем несколько раз пересматривался в работах [17-20], и в итоге был принят в качестве стандартной модели качества ISO 9126 [21] в 1991 году. Ниже описана модель качества ПО, описанная в действующей на сегодняшний день версии этого стандарта, принятой в 2001 году. В ходе идущей переработки этого и других связанных с оценкой качества ПО стандартов по новой схеме SQauRE [25] серьезные изменения в модели качества не планируются.

Рисунок 2. Факторы и атрибуты внешнего и внутреннего качества ПО по ISO 9126. Стандарт ISO 9126 [21-24] предлагает учитывать три разных точки зрения при рассмотрении качества ПО: точку зрения разработчиков, которые воспринимают *внутреннее качество ПО*, точку зрения руководства и аттестации ПО на соответствие сформулированным к нему требованиям, в ходе которой определяется *внешнее качество ПО*, и точку зрения пользователей, ощущающих *качество ПО при использовании*. Во всех трех случаях для описания качества используется предложенная МакКолом многоуровневая модель, состоящая из целей или факторов, атрибутов или критериев и метрик качества. Цели (факторы) позволяют на верхнем уровне определять основные характеристики, которые ПО должно иметь или уже имеет. Каждый фактор состоит из набора атрибутов (критериев), позволяющих качественно описать желаемые или полученные характеристики более детально. Каждый атрибут поддерживается набором метрик, которые позволяют количественно оценивать наличие соответствующей характеристики. Для двух точек зрения — внешнего качества и внутреннего качества — в рамках ISO 9126 предложена модель качества, состоящая из 6 факторов и 27 атрибутов и схематически представленная на Рис. 2.

Определения факторов и атрибутов качества в этой модели приведены ниже.

1 Функциональность (functionality) — способность ПО в определенных условиях решать задачи, нужные пользователям. Определяет, что именно делает ПО.

1 о Функциональная пригодность (suitability) — способность решать нужный набор задач.

2 о Точность (accuracy) — способность выдавать нужные результаты.

3 о Способность к взаимодействию, совместимость (interoperability) — способность взаимодействовать с нужным набором других систем.

4 о Соответствие стандартам и правилам (compliance) — соответствие ПО имеющимся стандартам, нормативным и законодательным актам, другим регулирующим нормам.

5 о Защищенность (security) — способность предотвращать неавторизованный и не разрешенный доступ к данным, коммуникациям и др. элементам ПО.

2 Надежность (reliability) — способность ПО поддерживать определенную работоспособность в заданных условиях.

1 о Зрелость, завершенность (maturity) — величина, обратная частоте отказов ПО. Определяется средним временем работы без сбоев и величиной, обратной вероятности возникновения отказа за данный период времени.

2 о Устойчивость к отказам (fault tolerance) — способность поддерживать заданный уровень работоспособности при отказах и нарушениях правил взаимодействия с окружением.

3 о Способность к восстановлению (recoverability) — способность восстанавливать определенный уровень работоспособности и целостность данных после отказа в рамках заданных времени и ресурсов.

4 о Соответствие стандартам надежности (reliability compliance).

3 Удобство использования (usability) или практичность — способность ПО быть удобным в обучении и использовании, а также привлекательным для пользователей.

1 о Понятность (understandability) — показатель, обратный к усилиям, которые затрачиваются пользователями на восприятие основных понятий ПО и осознание способов их использования для решения своих задач.

1 о Удобство обучения (learnability) — показатель, обратный к усилиям, затрачиваемым пользователями на обучение работе с ПО.

2 о Удобство работы (operability) — показатель, обратный трудоемкости решения пользователями задач с помощью ПО.

3 о Привлекательность (attractiveness) — способность ПО быть привлекательным для пользователей.

4 о Соответствие стандартам удобства использования (usability compliance).

2 Производительность (efficiency) или эффективность — способность ПО при заданных условиях обеспечивать необходимую работоспособность по отношению к выделяемым для этого ресурсам.

1 о Временная эффективность (time behaviour) — способность ПО решать определенные задачи за отведенное время.

2 о Эффективность использования ресурсов (resource utilisation) — способность решать нужные задачи с использованием заданных объемов ресурсов определенных видов. Имеются в виду такие ресурсы, как оперативная и долговременная память, сетевые соединения, устройства ввода и вывода, и пр.

3 о Соответствие стандартам производительности (efficiency compliance).

3 Удобство сопровождения (maintainability) — удобство проведения всех видов деятельности, связанных с сопровождение программ.

1 о Анализируемость (analyzability) или удобство проведения анализа — удобство проведения анализа ошибок, дефектов и недостатков, а также удобство анализа необходимости изменений и их возможных последствий.

2 о Удобство внесения изменений (changeability) — показатель, обратный трудозатратам на выполнение необходимых изменений.

3 о Стабильность (stability) — показатель, обратный риску возникновения неожиданных эффектов при внесении необходимых изменений.

4 о Удобство проверки (testability) — показатель, обратный трудозатратам на проведение тестирования и других видов проверки того, что внесенные изменения привели к нужным результатам.

1 о Соответствие стандартам удобства сопровождения (maintainability compliance).

2 Переносимость (portability) — способность ПО сохранять работоспособность при переносе из одного окружения в другое, включая организационные, аппаратные и программные аспекты окружения. Иногда в русскоязычной литературе эта характеристика называется мобильностью. Однако термин «мобильность» стоит зарезервировать для перевода «mobility» — способности ПО и системы в целом сохранять работоспособность при ее физическом перемещении в пространстве.

1 о Адаптируемость (adaptability) — способность ПО приспосабливаться к различным окружениям без проведения для этого действий, помимо заранее предусмотренных.

2 о Удобство установки (installability) — способность ПО быть установленным или развернутым в определенном окружении.

3 о Способность к сосуществованию (coexistence) — способность ПО сосуществовать в общем окружении с другими программами, деля с ними ресурсы.

4 о Удобство замены (replaceability) другого ПО данным — возможность применения данного ПО вместо других программных систем для решения тех же задач в определенном окружении.

5 о Соответствие стандартам переносимости (portability compliance).

Указанные выше характеристики используются для описания качества ПО с точки зрения его разработчиков и их руководства. Для

пользовательской точки зрения, т.е. качества ПО при использовании, стандарт ISO 9126 [21] предлагает другую систему факторов.

1 Эффективность (effectiveness) — способность решать задачи пользователей с необходимой точностью при использовании в заданном контексте.

2 Продуктивность (productivity) — способность предоставлять определенные результаты в рамках ожидаемых затрат ресурсов.

3 Безопасность (safety) — способность обеспечивать необходимо низкий уровень риска нанесения ущерба жизни и здоровью людей, бизнесу, собственности или окружающей среде.

4 Удовлетворение пользователей (satisfaction) — способность приносить удовлетворение пользователям при использовании в заданном контексте.

В дальнейшем мы будем использовать систему из шести факторов, предназначенную для оценки и описания внешнего или внутреннего качества ПО, поскольку все методы верификации затрагивают именно эти его аспекты.

3. Методы верификации программного обеспечения

В данном разделе рассматриваются методы верификации ПО, в основном нацеленные на оценку технических артефактов жизненного цикла. Такие методы в имеющейся литературе разделяются на следующие группы.

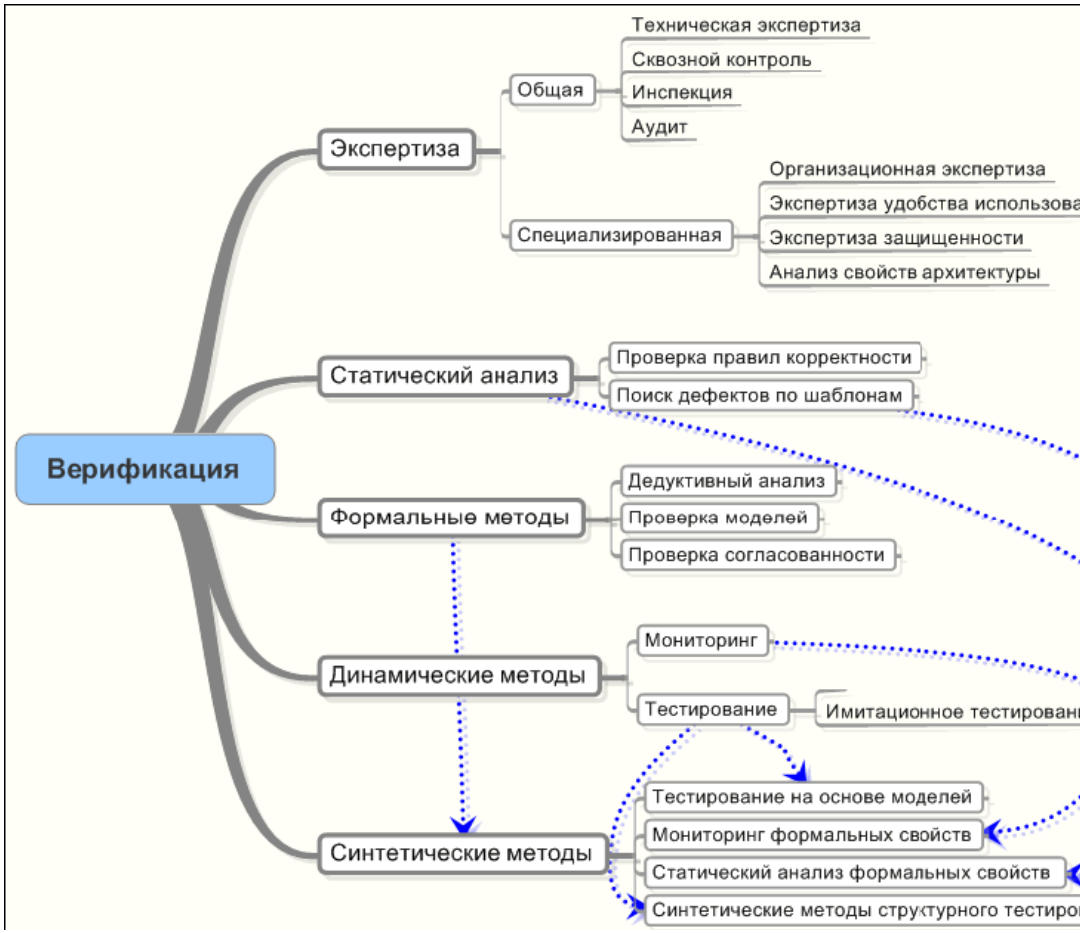


Рисунок 4. Схема используемой классификации методов верификации.

1 *Экспертиза* (review) различных артефактов жизненного цикла ПО. Обычно в качестве видов экспертиз выделяют организационные экспертизы (management review), технические экспертизы (technical review), сквозной контроль (walkthrough), инспекции (inspection) и аудиты (audit). С середины 1990-х активно

развиваются методы оценки архитектуры ПО на основе сценариев (scenario based software architecture evaluation), обычно не

1 соотносимые с «традиционными» экспертизами. От других методов верификации экспертизу отличает возможность выполнять ее, используя только сами артефакты жизненного цикла, а не их модели (как в формальных методах) или результаты работы (как в динамических).

Экспертиза применима к любым свойствам ПО и любым артефактам жизненного цикла и на любом этапе проекта, хотя для разных целей могут использоваться разные ее виды. Она позволяет выявлять практически любые виды ошибок, причем делать это на этапе подготовки соответствующего артефакта, тем самым минимизируя время существования дефекта и его последствия для качества производных артефактов. В то же время экспертиза не может быть автоматизирована и требует активного участия людей. Эмпирические наблюдения показывают, что эффективность экспертиз в терминах отношения количества обнаруживаемых дефектов к затрачиваемым на это ресурсам несколько выше, чем для других методов верификации. Так, различные отчеты показывают, что от 50 до 90% всех зафиксированных в жизненном цикле ПО ошибок может быть обнаружено с помощью экспертиз [56-61]. За счет их раннего обнаружения может быть достигнута существенная экономия ресурсов — затраты на обнаружения ошибки составляют от 5 до 80% от таких же затрат при использовании тестирования [59-63]. Кроме того, регулярное участие в экспертизах является важным фактором в обучении сотрудников и способствует повышению качества результатов их работы. В то же время эффективность экспертизы существенно зависит от опыта и мотивации ее участников [58-61], организации процесса, а также от обеспечения корректного взаимодействия между различными участниками. Это накладывает дополнительные ограничения на распределение ресурсов в проекте и может приводить к конфликтам между разработчиками, если руководство проекта обращает мало внимания на коммуникативные аспекты проведения экспертиз.

1 *Статический анализ* свойств артефактов жизненного цикла ПО используется для проверки формализованных правил корректного построения этих артефактов и поиска часто встречающихся дефектов по некоторым шаблонам.

2 Такой анализ хорошо автоматизируется и может быть практически полностью возложен на инструменты, хотя иногда необходимо вручную определить, например, принятые в проекте стандарты кодирования. Однако применим он лишь к коду или к определенным форматам представления проектных артефактов, и способен обнаруживать только ограниченный набор типов ошибок. Одной из известных проблем статического анализа является также следующая дилемма: либо используются строгие методы анализа, не допускающие пропуска ошибок (тех типов, что ищутся), но приводящие к большому количеству сообщений о возможных ошибках, которые таковыми не являются, либо точным является набор сообщений об ошибках, но возникает возможность пропустить ошибку. Инструменты автоматической верификации на основе статического анализа применяются достаточно широко, поскольку не требуют специальной подготовки и достаточно удобны в использовании. Большинство техник статической проверки корректности программ, доказавших свою эффективность на практике, рано или поздно становятся частью компиляторов или даже преобразуются в семантические правила языков программирования.

1 *Формальные методы верификации* используют для анализа свойств ПО формальные модели требований, поведения ПО и его окружения. Анализ формальных моделей выполняется с помощью специфических техник, таких как *дедуктивный анализ* (theorem proving), *проверка моделей* (model checking) или *абстрактная интерпретация* (abstract interpretation). Формальные методы применимы только к тем свойствам, которые выражены формально в рамках некоторой математической модели, а также к тем артефактам, для которых можно построить адекватную формальную модель. Соответственно, для использования таких методов в проекте необходимо затратить значительные усилия на построение формальных моделей. К тому же, построить такие модели и провести

их анализ могут только специалисты по формальным методам, которых не так много, и чьи услуги стоят достаточно дорого. Построение формальных моделей нельзя автоматизировать, для этого всегда необходим человек. Анализ их свойств в значительной мере может быть автоматизирован, и сейчас уже есть инструменты, способные анализировать формальные модели промышленного уровня сложности, однако чтобы эффективно пользоваться ими часто тоже требуется очень специфический набор навыков и знаний (в специфических разделах математической логики и алгебры). Тем не менее, в ряде областей, где последствия ошибки в системе могут оказаться чрезвычайно дорогими, формальные методы верификации активно используются. Они способны обнаруживать сложные ошибки, практически не выявляемые с помощью экспертиз или тестирования. Кроме того, формализация требований и проектных решений возможна только при их глубоком понимании, и поэтому вынуждает провести тщательнейший анализ этих артефактов, для чего часто необходима совместная работа специалистов по формальным методам и экспертов в предметной области. В последние 10 лет появились основанные на формальных методах инструменты [64-68], решающие достаточно ограниченные задачи верификации ПО из определенного класса, но зато способные эффективно работать в промышленных проектах и требующие для применения минимальных специальных навыков и знаний. Гораздо чаще, чем к программам, формальные методы верификации на практике применяются к аппаратному обеспечению [69-72]. Их использование в этой области имеет более долгую историю, что привело к созданию более зрелых методик и инструментов. Это обусловлено более высокой стоимостью ошибок для аппаратного обеспечения, более однородной его структурой и более простыми примитивными элементами, более широким многократным использованием проектной информации, а также большей привычностью строгих ограничений и точных описаний для инженеров.

2 *Динамические методы верификации*, в рамках которых анализ и оценка свойств программной системы делаются по результатам ее реальной работы или работы некоторых ее моделей и прототипов. Примерами такого рода методов являются обычное *тестирование* или

3 *имитационное тестирование*, *мониторинг*, *профилирование*. Для применения динамических методов необходимо иметь работающую систему или хотя бы некоторые ее компоненты, или же их прототипы, поэтому нельзя использовать их на первых стадиях разработки. Зато с их помощью можно контролировать характеристики работы системы в ее реальном окружении, которые иногда невозможно аккуратно проанализировать с помощью других подходов. Динамически методы позволяют обнаруживать в ПО только ошибки, проявляющиеся при его работе, а, например, дефекты удобства сопровождения найти не помогут, однако, обнаруживаемые ими ошибки обычно считаются более серьезными. Для применения динамических методов верификации обычно требуется дополнительная подготовка — создание тестов, разработка тестовой системы, позволяющей их выполнять или системы мониторинга, позволяющей контролировать определенные характеристики поведения проверяемой системы. Но системы тестирования, профилирования или мониторинга могут быть сделаны один раз и использоваться многократно для широких классов ПО, лишь сами тесты необходимо готовить заново для каждой проверяемой системы. В то же время подготовка тестов на ранних этапах создания ПО позволяет обнаружить множество дефектов в описании требований и проектных документах — фактически, разработчики тестов вынуждены в ходе своей деятельности выполнять экспертизу артефактов, служащих основой для тестов. Создание набора тестов, позволяющих получить адекватную оценку качества сложной системы, является довольно трудоемкой задачей. Однако среди разработчиков промышленного ПО сложилось (не вполне верное) мнение, что тестирование является наименее ресурсоемким методом верификации, поэтому на практике оно используется для оценки свойств ПО очень широко. При этом чаще всего применяются не слишком надежные, но достаточно дешевые техники, такие как (нестрогое) вероятностное тестирование, при котором тестовые данные генерируются случайным образом, или же тестирование на

основе простейших сценариев использования, проверяющие лишь наиболее простые ситуации.

1 *Синтетические методы.* В последние 10-15 лет появилось множество исследовательских работ и инструментов, в рамках которых применяются элементы нескольких перечисленных выше видов верификации. Так, в отдельные области выделились динамические методы, использующие элементы формальных, — *тестирование на основе моделей* (model-based testing, model driven testing) [73] и *мониторинг формальных свойств* (runtime verification, passive testing). Ряд инструментов построения тестов существенно использует как формализацию некоторых свойств ПО, так и статический анализ кода. Общая идея таких методов вполне понятна — попытаться сочетать преимущества основных подходов к верификации, купировав их недостатки.

Представленная здесь классификация методов верификации скорее обусловлена историческими причинами, чем основана на существенных характеристиках самих этих методов. Исследователи и разработчики новых методов и инструментов, пытаясь соотнести свои работы с работами предшественников, обычно ищут их в рамках одной из указанных групп, поэтому в настоящий момент такая схема достаточно удобна. Однако, как уже было сказано, в последнее время создаются синтетические методы, сочетающие элементы всех остальных, и через 5-10 лет, после появления достаточно большого количества таких подходов, потребуется более детальная и содержательная классификация методов верификации. Далее при обзоре отдельных методов верификации методы, относящиеся к одному типу, рассматриваются вместе. Чаще всего при этом описывается более-менее детально только один представитель этого типа, а также указываются характеристики, по которым методы того же типа могут отличаться друг от друга.

Тема 12 Логико-математические методы верификации программ

1. Понятие верификация программы.
2. Доказательство конкретности с помощью утверждений.

I. ***Верификация*** - это установление подлинности, проверка истинности; верификация
(в программировании) это доказательство правильности программ. [Першиков, Савинков, 1995]

Верификация (программы) - это установление соответствия между программой и её спецификацией, описывающей цели разработки. [Непомнящий, Рякин, 1988, с.89-90]

С точки зрения математической логики верификация - это вычисление истинности предиката, зависящего от двух переменных: программы (на языке программирования) и спецификации (на языке спецификаций). Истинность этого предиката устанавливает свойство корректности программы, если спецификация не содержит ошибок; однако его ложность, вообще говоря, ещё не означает наличие ошибок в программе, а требует более тонкой проверки.

Формальная верификация программы - это преобразование доказательства правильности программы в доказательство некоторой теоремы в исчислении первого порядка.

Способность строить маленькие программы является необходимым условием для построения больших, хотя, может быть, и недостаточным. Весьма легко убедиться в справедливости этого утверждения на следующем примере. Предположим, что программа состоит из n небольших компонентов (процедур или модулей), каждый из которых правилен с вероятностью p. Тогда вероятность P правильности всей программы, удовлетворяет неравенству $P<p^n$. Так как для любой достаточно большой программы n велико, то надеяться на правильность программы можно лишь при значениях p, близких к 1.

Чтобы быть уверенным в том, что программа работает правильно, лучше всего иметь доказательство её правильности. Здесь существует две возможности:

- доказывать правильность готовых программ (верификация программ);

- строить программы apriori (заведомо) правильными, параллельно с доказательством их правильности (синтез программ).

Верификацией программ будем называть доказательство правильности готовых ("упавших с неба") программ.

Синтезом программ будем называть построение программ параллельно с доказательством их правильности.

На первый взгляд кажется, что проще доказывать правильность готовых программ. Но на практике оказывается, что доказывать правильность уже существующей программы слишком трудно. Поэтому целесообразнее использовать идеи доказательства правильности ещё *при разработке программы* для осознания того, что делать дальше. В этом случае программы получатся яснее, изящнее и зачастую эффективнее, чем программы, написанные бессистемно.

Синтезирующее программирование - это программирование в наиболее чистом виде, т.е. процесс получения программы "из ничего", исходя из условия задачи и метода ее решения. С формальной точки зрения условие задачи записывается в виде *спецификации* задачи. В последующем синтезе программы по спецификациям различают два подхода: *логический* и *аналитический* .

При **логическом подходе** спецификация трактуется как формулировка теоремы, утверждающей существования решения задачи. Синтез программы состоит в поиске доказательства этой теоремы существования в некотором конструктивном логическом исчислении. Если такое доказательство удаётся построить, то существуют формальные правила, позволяющие преобразовать доказательство в программу, находящую решение задачи. Существенно, что существует универсальная контролирующая процедура, которая проверяет правильность доказательства и применения правил перехода от доказательства к программе.

При **аналитическом подходе** спецификация трактуется как уравнение относительно программы. Оказывается, такое уравнение можно подвергать символическим преобразованиям, при которых символ неизвестной программы "рассыпается" на систему вспомогательных неизвестных, а они, в свою очередь, заменяются либо другими неизвестными, либо конкретными программными конструкциями. Таким образом, по мере преобразования синтезируемая программа постепенно "прорастает" в тексте спецификации, в конце концов превращая ее в законченный программный текст. Опять-таки правильность выполнения символических преобразований может формально проверяться на каждом шагу.

II. Доказательство конкретности с помощью утверждений.

Рассмотрим формальное доказательство программы, заданной структурной логической схемой и совокупностью утверждений, задаваемых логическими операторами, комбинациями переменных (true/false), операциями (конъюнкция, дизъюнкция и др.) и кванторами всеобщности и существования (табл. 1).

Таблица 1. Список логических операций

Логические операции		
Название	**Примеры**	**Значение**
Конъюнкция	$x \,\&\, y$	x и y
Дизъюнкция	$x * y$	x или y
Отрицание	$\neg x$	не x
Импликация	$x \rightarrow y$	если x то y
Эквивалентность	$\mathrm{x} = \mathrm{y}$	x равнозначно y
Квантор всеобщности	$\forall \mathrm{x}\ \mathrm{P(x)}$	для всех x, условие истинно
Квантор существования	$\exists \mathrm{x}\ \mathrm{P(x)}$	существует x, для которого $\mathrm{P(x)}$ истина

Список рекомендуемой литературы

1 Нурбекова Ж.К. Теоретико-методологические основы обучения программированию: монография. – Павлодар, 2004. – 225с.

2 Бидайбеков Е.Ы., Гриншкун В.В. Интеграционные методы преподавания алгоритмических языков в университетском курсе информатики// Материалы международной научно-методической конференции «Математическое моделирование и информационные технологии в образовании и науке». - Алматы: АГУ им. Абая, 1998.

3 Дроботун Б.Н. Методическая система обучения логико-алгебраическим дисциплинам в высших учебных заведениях: дисс. ... докт.пед. наук.- Алматы, 2008. – 442 с.

4 Нурбекова Ж.К. Фундаментальное и опережающее обучение программированию студентов по специальности «Информатика»: дисс. …докт. пед.наук.- Алматы, 2007.

5 Джарасова Г.С. Экспериментальное исследование формирования логической культуры будущих информатиков // Вестник ПГУ.- 2009.- №1.- С.62-69.

6 Джарасова Г.С. Обучение будущих информатиков применению рекурсивного метода программирования // Материалы научно-практической конференции «Информационные и коммуникационные технологии в образовании» - Екатеринбург: ГОУ ДПО ИРРО, 2009. – С.280-282; 308 с.

7 Нурбекова Ж.К., Джарасова Г.С., Байгушева К.М. Междисциплинарные связи при формировании логической культуры будуших кадров в области информатики// Бюллетень центра информатики и информационных технологий в образовании. - М.:ИСМО РАО, 2008. - Вып.4.- С.70-74; 152 с.

8 Клини С.К. Математическая логика: Пер. с англ. – М.: Мир, 1973. – 480с.

9 Дроботун Б.Н., Джарасова Г.С. К проблеме формирования алгоритмической культуры студентов// Материалы научной конференции молодых ученых, студентов и школьников, III Сатпаевские чтения. – Павлодар, 2003.- Т.7.- С. 49-54.

10 Дроботун Б.Н., Джарасова Г.С. Порядковые отношения и методологические подходы к их изучению // Республиканский научный журнал «Поиск» .- 2005.- №4.- С.261 – 264.

Дополнительная

11 Дроботун Б.Н., Джарасова Г.С. О концепции изучения алгебраических систем с точностью до изоморфизма // Вестник КазНПУ им. Абая. Серия физико-математические науки.- 2006.- №1(15).- С. 58 – 66.

12 http://cmap.coginst.uwf.edu/info/

13 http://dl.nw.ru/theories/cmaps/

14 Глобализация и конвергенция образования: технологический аспект: научное издание / Под общей редакцией профессора Ю.Б. Рубина. – М.: ООО «Маркет ДС Корпорейшн», 2004. – 540 с.

15 http://www.yugzone.ru/mindmap.htm

16 Дроботун Б.Н., Джарасова Г.С. Из опыта изучения элементов дискретной математики в средней школе// Материалы научной конференции молодых ученых, студентов и школьников, II Сатпаевские чтения.- Павлодар, 2002.- Т.2.- С. 30-38

17 Дроботун Б.Н., Джарасова Г.С. Программа факультатива «Элементы дискретной математики» в школе (9 кл) // В кн: Факультативы по школьным дисциплинам (информатика, математика, биология, физика, химия), серия «Портфель молодого учителя».- Павлодар: НИЦ ПГУ, 2003.- С.19-22.

18 Дроботун Б.Н., Джарасова Г.С. Вводный курс математики. - Павлодар: Научно-издательский центр ПГУ им.С.Торайгырова, 2004. – 300с.

19 Дроботун Б.Н., Джарасова Г.С. К проблеме методического обеспечения дистанционного обучения// Материалы международной научно-практической конференции «Компьютеризация обучения и проблемы гуманизации образования в техническом ВУЗе». – Пенза, 2003.- С.459-462.

20 Дроботун Б.Н. Из опыта пропедевтического понятия «Алгоритм» //Вестник ПГУ им.С.Торайгырова. Серия педагогическая. -2007.- №1.- С. 60 – 66.

21 Дроботун Б.Н. О концепции содержательного мотивационно-ориентированного подхода в системе вузовской

логико-алгебраической подготовки //Высшая школа Казахстана.- 2008.- №2.- С. 55-60.

1.