Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

Л.В. Городняя

ФУНКЦИОНАЛЬНЫЙ ПОДХОД К ОПИСАНИЮ ПАРАДИГМ ПРОГРАММИРОВАНИЯ

Препринт
152

Новосибирск 2009

Препринт посвящен исследованию и спецификации основных парадигм программирования. Проанализированы особенности языков программирования разного уровня от ассемблера до языков параллельного программирования. Дана характеристика функционального подхода к сравнительному описанию реализационной семантики основных парадигм. Предложена схема описания и определения парадигматических характеристик языка программирования. В качестве иллюстрации использованы фрагменты языков программирования разного уровня, относящиеся к парадигмам машинно-ориентированного, системного, императивного, функционального, логического, объектно-ориентированного и высокопроизводительного программирования.

Работа поддержана грантом РФФИ № 08-01-00899-а.

© Институт систем информатики им. А. П. Ершова СО РАН, 2009

Siberian Branch of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

L.V. Gorodnyaya

A FUNCTIONAL APPROACH TO PROGRAMING PARADIGM DESCRIPTION

Preprint
152

Novosibirsk 2009

The work describes research and specification of basic paradigms of programming. The author analyzes special features of programming languages of different levels, from Assembler to parallel programming languages. A functional approach to comparative description of implementation semantics of basic paradigms is proposed.

The author proposes a scheme of describing and defining paradigm features of programming languages. The approach is illustrated with fragments of programming languages of different levels, which belong to machine-oriented, system, imperative, functional, logical, object-oriented, and high-performance programming.

The work is supported by the RFBR grant # 08-01-00899-a.

© A. P. Ershov Institute of Informatics Systems, 2009

1. ВВЕДЕНИЕ

Парадигмы программирования различаются нишей в жизненном цикле программ, приоритетами при оценке качества программ, выбором инструментов и методов обработки данных [86, 115, 146, 147]. Значимость используемых при этом критериев по существу зависит от условий применения программируемых решений. Упорядочение критериев нередко претерпевает изменения по мере развития сферы применения программы, роста квалификации пользователей, модернизации оборудования, информационных технологий и программотехники, что и приводит к появлению новых парадигм. Первые парадигмы автоматного и машинно-ориентированного императивного программирования сформировались как кодирование готовых алгоритмов, сложившихся и отлаженных в докомпьютерную эпоху [11, 13, 18, 67, 110]. Системное и функциональное программирование появляются при расширении класса задач, требующих разработки и отладки новых алгоритмов и приведения их в удобную для обычного, непосредственного, программирования форму [16, 20, 9, 12, 15, 21–24, 31, 34, 40, 42, 81, 62,63, 66, 69, 80, 82, 95, 105, 111, 117, 123, 126, 130, 131, 136, 139–142, 98, 1, 53]. Затем приходит пора языков представления знаний и, соответственно, декларативного (логического) программирования рецептов решения новых задач, где факт существования решений важнее их эффективной реализации [48, 54, 55, 75, 92]. Профессионализация программирования приводит к объектно-ориентированному программированию, поддерживающему повторное использование запрограммированных готовых решений в разных системах [51, 94, 118, 155]. На повестке дня – создание удобных языков параллельного программирования [7, 25, 28, 35, 44, 45, 56-59, 74, 83, 85, 87, 89, 96, 112, 120, 124, 128, 137, 144].

Новые возможности компьютерных сетей, суперкомпьютеров, общедоступных баз данных, массовое распространение мобильных устройств – все это меняет сферу применения информационных технологий и влечет кристаллизацию новых и более общих парадигм компьютерных языков. В данной работе представлена попытка характеризовать парадигмы программирования взаимодействием четырех основных семантических систем: обработки данных, их хранения и структурирования, управления обработкой данных [74].

В препринте приведен аналитический обзор парадигм программирования, рассмотрены особенности применения языков программирования, от-

ражающие методы их реализации и класс решаемых на их основе задач. Отмечены ключевые моменты развития парадигм программирования, анализируются закономерности освоения новых методов обработки данных посредством реализации систем программирования и других информационных систем. Определен ряд опорных языков для сравнительного анализа языков программирования при определении их соответствия конкретным парадигмам [12, 17, 142, 99, 114, 1, 81, 57, 89, 92, 118, 126, 127, 143].

При отборе материала использован исторический подход [6, 90, 97, 116, 122]. Материал сформирован в процессе подготовки лекций для студентов 2-го курса ММФ и 3-го курса ФИТ НГУ, а также Интернет-университета информационных технологий в 2004–2008 гг.

2. РЕАЛИЗАЦИОННАЯ СЕМАНТИКА

Средства и методы программирования складывались на фоне расширения сферы применения компьютерных технологий. Произошло расслоение парадигм программирования в зависимости от глубины и общности технических решений по организации процессов обработки данных. Обнаружились явные сложности классификации языков программирования (ЯП) и определения их принадлежности конкретной парадигме программирования.

При отработке методики проявления парадигматической характеристики ЯП в виде взаимодействия основных семантических систем[1], таких как обработка данных, их хранение и структурирование, управление обработкой данных, выделены три уровня парадигм, отражающие расширение языковой поддержки жизненного цикла программ (ЖЦП) и рост реализационной сложности определения ЯП:

- низкоуровневое кодирование (ЯНУ);
- программирование на языках высокого уровня (ЯВУ);
- подготовка программ на базе языков сверх высокого уровня (ЯСВУ).

Для каждой парадигмы существуют причисляемые к ней, ей соответствующие, можно сказать, «референтные» или «опорные» языки программирования, и во многих языках представлен или реализован ряд парадигм. При исследовании и спецификации парадигматических свойств таких языков естественно определение языка представлять объединением определений монопарадигматических подъязыков, являющихся фрагментами исходного языка [98, 141].

[1] Понятие «семантическая система» предложено С.С. Лавровым [77].

Проблемой является неоднозначность семантической декомпозиции сложных определений [37], трудоемкость определения реализационных особенностей полных интерпретаторов и компиляторов, а также немалое число команд в абстрактных машинах для реальных ЯП. Кроме того, как показывает опыт определения классов объектов в языках ООП, критериев функционального назначения и обусловленности типами данных не достаточно для выделения простых семантических систем, удобных для сравнения. Поэтому возникает потребность в разложении определения языка на концентры и слои [4, 8, 93]. Для классификации наиболее важно выделение минимальных учебного концентра и реализационного ядра. В результате множество ЯП можно структурировать на классы операционно/реализационно подобных и содержательно сравнимых языков, обладающих общими, точнее – эквивалентными, концентрами или слоями[2]. Нередко реализационное ядро языка содержит ряд понятий, не имеющих прямого представления в анализируемом ЯП. Дополнительные понятия обеспечивают реализационную полноту языка, существенно влияющую на понимание механизмов эффективного программирования.

Декомпозиция учебного концентра, объединенного с реализационным ядром, определения ЯП на основные семантические системы дает не слишком сложные параметры для сравнительного анализа и установления парадигматических отношений между ЯП. Относящиеся к одной парадигме подъязыки разных языков легко сравнить на уровне структурированной таким образом операционной семантики, анализируя определения их абстрактных машин [3, 50] и интерпретаторов. Это позволяет с каждой парадигмой связать варианты реализационных особенностей основных семантических систем и правил их взаимодействия, что можно назвать реализационной семантикой [39]. К основным системам реализационной семантики отнесены выполнение вычислений, работа с памятью, управление вычислениями и организация структур данных [37, 74]. Выделение таких систем обусловлено различиями в схеме применения операций к операндам, связанными с методами реализации СП и их поддержкой на уровне аппаратуры [38, 43].

Вычисления характеризуются совпадением типов результата и аргументов. Традиционные формы вычислений позволяют строить потоки вычислений и конвейерные процессы, управляемые готовностью данных, без именования промежуточных значений. Такие системы можно различать по

[2] Согласно Венской методике определения ЯП эквивалентными считаются языки, сводимые к одному абстрактному синтаксису [141].

мощности множества допустимых значений и набору встроенных операций над ними.

Работа с памятью осуществляется как операции над неявной таблицей, связывающей адреса и хранимые значения. Встречаются разные ограничения на обращение к этой таблице, формулируемые как дисциплина доступа к памяти или правила видимости именованных данных. Кроме того различается техника обработки таблицы и ее структура. Чаще всего встречаются системы памяти, ориентированные на работу с ассоциативно именуемыми значениями (value-oriented), с именованными переменными (name-oriented), с прямыми указателями (pointer-oriented) и неявными копиями значений в стеке (stack-oriented)

Управление вычислениями позволяет варьировать ход порожденных программой процессов в зависимости от данных или событий, символизирующих определенную логику корректности вычислений или успеха функционирования системы, связанных со спецификой реализации предикатов, представления различимых событий и реакций на события. Противопоставляются системы с фиксированным или программируемым набором схем управления. Встречаются системы с защитой процессов и/или данных. Возможна организация приостановок, учета временных отношений, обработки прерываний.

Организация структур данных обеспечивает конструктивность сложных построений, возможность восстановления составляющих вплоть до элементарных данных и их эффективной обработки «по частям». Противопоставляются целостные и распределенные структуры данных, статическое и динамическое размещение данных в памяти, аналитические, счетчиковые и программные методы повторного использования памяти для структур данных.

Допустимые вариации при определении соответствия конкретной парадигме на уровне основных семантических систем в компьютерных языках проиллюстрированы на примерах парадигм низкоуровневого кодирования, системного, функционального, логического и объектно-ориентированного программирования, а также языков сверх высокого уровня, показывающих характерные идеи, средства и методы, послужившие основанием для кристаллизации отдельной парадигмы. Пары «интерпретатор (универсальная функция)» и «абстрактная машина», задающие семантику фрагментов рассматриваемых языков, используются как параметр при классификации парадигм и сравнении языков, анализе их подобия и сопоставимости. Определения интерпретаторов ограничены рассмотрением фрагментов языков, включающих следующие сквозные понятия:

Таблица 1

Список понятий ЯП, различаемых интерпретаторами, определяемыми для сравнения

Атом (Элементарное)
Структура
Переменная
Значение
Выражение
Действие/Операция
Условие/истина
Функция/ подпрограмма/ оператор/макрос/процедура/ тип данных/метод [3]
Аргумент
Вызов Фн/подпр
Определение Фн/подпр
Идентификатор

Принимая, что в изучении парадигм программирования центральную роль играет семантика, следует вспомнить основные положения Венской методики[4], согласно которой языки программирования определяют в терминах операционной семантики, используя типичные понятия программирования при спецификации систем программирования, что удобно для сравнительного описания парадигм программирования [98, 141, 142, 29, 39, 33].

Благодаря хорошо разработанной концепции абстрактных объектов, позволяющей концентрировать внимание лишь на существенном и игнорировать второстепенные детали, Венский метод годится для описания и машин, и алгоритмов, и структур данных, особенно при обучении основам системного программирования. Согласно концепции абстрактных объектов (абстрактный синтаксис, абстрактная машина, абстрактные процессы) интерпретирующий автомат содержит в качестве компоненты состояния управляющую часть, содержимое которой может изменяться с помощью

[3] Операторы управления рассматриваются как отдельная категория функций.

[4] Венский метод (ВМ) определения языков программирования был разработан в 1968 году в Венской лаборатории IBM под руководством П. Лукаса на основе идей, восходящих к Дж. Маккарти [98, 141, 142].

операций, подобно прочим данным, имеющимся в этом состоянии [50]. При сравнении сложности интерпретаторов для разных языков можно использовать нормализованные предикатные формы, сопоставляя предикаты с выделяемыми понятиями.

Модель автомата с состояниями в виде древовидных структур данных, созданного согласно Венской методике для интерпретации программ, является достаточно простой[5]. Тем не менее, она позволяет описывать основные нетривиальные понятия программирования, включая локализацию определений по иерархии блоков вычислений, вызовы процедур и функций, передачу параметров. Такая модель поддерживает представление понятий программирования, полезное в качестве стартовой площадки для исследования разных парадигм программирования и сравнительного анализа стилей программирования.

Методы верификации программ активно используют структурную и рекурсивную индукцию[6] при доказательстве таких свойств рекурсивно определенных функций, как эквивалентность. Поэтому операционная семантика языка программирования, заданная над списочными структурами и поддерживающая все уровни определения языка от текста до кода, включая моделирование средств различных парадигм программирования, образует технологичную основу для решения актуальных проблем разработки надежных информационных систем.

При неформальной характеристике стиля программирования отмечаются различия в акцентах при ответе на следующие вопросы:

- Кто является главным субъектом при определении программы?
- В чем заключается метод обработки программы?
- Где сосредоточена результативная активность программы?
- Когда принимается решение о продолжении начатых вычислений?
- До каких пор, в каких пределах, сколько планируется функционирование участков повторяемости?

---

[5] При сравнении императивного и функционального подходов к программированию П.Лэндин (P.J.Landin) предложил абстрактную машину SECD для спецификации машинно-зависимых аспектов семантики Лиспа [126]. Это автомат, работающий над четырьмя структурными регистрами: стеком для промежуточных результатов, контекстом для размещения именованных значений, управляющей вычислениями программой, резервной памятью (Stack, Environment, Control list, Dump). Регистры приспособлены к хранению выражений в форме атомов или списков. Состояние машины полностью определяется содержимым этих регистров. Поэтому функционирование машины можно описать достаточно точно в терминах изменения содержимого регистров при выполнении команд, что выражается следующим образом:

s e c d > s' e' c' d' – переход от старого состояния к новому.

[6] Предложена Дж. Маккарти

- Каким способом гарантирована корректность сложной информационной обработки?

Первые языки программирования обладали машинной ориентированностью и поддерживали принципиально важную, но небольшую по длительности и трудозатратам часть ЖЦП − от 2-х до 5 %, заключающуюся в кодировании готовых алгоритмов в терминах автоматов. Появление ЯВУ расширило языковое покрытие ЖЦП примерно до 10 % для хорошо поставленных задач, имеющих алгоритмы решения над типовыми структурами данных, приспособленные для нисходящих методик программирования. Парадигма функционального программирования посягнула на ЖЦП для задач с исследовательским компонентом и расширила его языковое покрытие до 50 % благодаря механизмам хранения и накопления информации о свойствах информационных объектов, полезной при отладке и модификации программ, составленных из небольших универсальных компонент, допускающих как нисходящую, так и восходящую методику разработки. Логическое программирование распространило эти механизмы на не вполне определенные постановки задач, что дало языковую поддержку предварительному сбору фактического материала, созданию демонстрационных версий, пробному прототипированию, отчасти тестированию и довело общее языковое покрытие ЖЦП почти до 60 % при восходящей методике разработки. Появление объектно-ориентированного программирования преодолело итеративность ЖЦП для задач, связанных с развивающимися областями приложения, что довело языковое покрытие примерно до 80 %.

Далее полнота языкового покрытия ЖЦП обеспечивается компьютерными языками (КЯ), возникающими в связи с информационными технологиями, телекоммуникациями, распределенными информационными системами, электронным общением, автоматизацией и самоорганизацией управления проектами, а также специализированными языками для решения конкретных классов задач. Анализ концепций и классификация представительного свода КЯ в контексте профессиональной речевой практики в настоящее время стихийно разворачивается при формировании открытых онтологий и энциклопедий, что расширяет возможности для решения проблемы глубинного анализа содержания обучения информатике и информационным технологиям.

Выводы:

1. Определение парадигматической характеристики ЯП начинается с формулировки ясной общей идеи, задающей принципы, по которым можно отнести язык к конкретной парадигме.

2. Нужен анализ понятий, общих для разных языков.
3. Реализационная семантика языка характеризуется списком семантических систем, при описании которых могут потребоваться дополнительные понятия.
4. Базовые возможности языка характеризуются его абстрактной машиной.
5. Расширенные возможности языка задаются в форме интерпретатора.
6. При декомпозиции определения языка на фрагменты и подсистемы используется фразеологическая характеристика операционной семантики языка, что позволяет прагматически выбрать критерии выделения компонентов, обладающих общностью для разных языков.

3. ЯЗЫКИ НИЗКОГО УРОВНЯ

Низкоуровневое кодирование ассоциируется с одноуровневыми структурами данных, обусловленными архитектурой и оборудованием[7]. При хранении данных и программ используется глобальная память. Обработка данных сводится к императивной машинно-ориентированной модели управления процессом выполнения действий, порожденных программой. Результативен автоматный подход к реализации алгоритмов, нацеленных на свободный доступ к любым возможностям оборудования. В центре внимания − конфигурация оборудования, состояние памяти, система команд, передачи управления, очередность событий, исключения и неожиданности, время реакции устройств и успешность процессов обработки информации. Кодирование осуществляется на фоне применения вспомогательных средств, таких как блок-схемы и документирование, отчасти компенсирующих отсутствие в ЯНУ понятий уровня программистской фразеологии, а в естественных языках − понятий, возникающих при такой «сверхточной» детализации программ. В принципе достижима предельная эффективность программ, но их отладка осложнена сочетанием «низкий старт – высокий финиш». Иными словами, легко достичь успеха в первых упражнениях, но трудно создать программный продукт и обеспечить его квалифицированное сопровождение. Для ЯНУ характерна однозначность соответствия между программой и процессом, порождаемым при ее реализации. Поэтому анализ операционной семантики ЯНУ достаточен для вывода свойств программ и процессов, подготовленных с помощью ЯНУ. Как правило, при

[7] Ассемблер «Эльбрус» и автокод «Инженер» − контрпримеры, показывающие недостаточность чисто приаппаратной оценки уровня языка [100].

определении абстрактной машины ЯНУ достаточно трех регистров, назначение которых соответствует реализации понятий «результат», «контекст» и «программа».

Традиционно к языкам низкого уровня (ЯНУ) относят языки ассемблера, макропроцессоры, машинно-ориентированные языки, языки управления заданиями [3, 52, 72, 17, 12, 134, 136]. Для ЯНУ характерно, что все действия в программе выражены явно. Программа – произвольная смесь команд, соседство которых практически не ограничивается. Точно определены все решения по представлению значений и структур данных в памяти и схема управления их обработкой, что позволяет четко относить ЯНУ к одной конкретной парадигме.

Семантика ЯНУ обычно содержит целочисленную арифметику, ограниченную разрядностью адресов или машинных слов, использует работу с общими, глобальными идентификаторами, поддерживает последовательное управление вычислениями и осуществляет организацию структур данных по принципу соседства элементов, расположенных в памяти (вектора, строки, стеки, очереди, файлы). Можно рассчитывать, что так определена часть учебного концентра (элементарного уровня) любого ЯНУ. Элементарные уровни языков ассемблера, макропроцессора, вычислений над стеком и управления заданиями могут рассматриваться как расширения такого концентра.

3.1. Машинно-зависимое программирование. Языки ассемблера

Для языков типа ассемблера в дополнение к общей семантике элементарного уровня ЯНУ характерно наличие системы вычислений над вещественными числами и системы работы с кодами. Работа с идентификаторами дополняется аппаратными возможностями адресации памяти (прямой, неявной, индексной, модифицированной, косвенной, относительной, стековой), обычно обеспечивающей доступ практически к любому хранимому в памяти данному или команде. Управление вычислениями может учитывать результаты промежуточных вычислений (ветвления и переключатели), выполнять итерирование участков повторяемости (циклы) и вызовы подпрограмм, а также обрабатывать внутренние и внешние прерывания. В качестве поддержки структур данных можно рассматривать пересылки блоков заданной длины, а также средства работы с текстом программы. В качестве опорных языков рассмотрены ассемблеры MIX, MSX, MASM и БЭМШ [70, 135, 99] .

Абстрактная машина ассемблера может быть задана тройкой

<сумматор: значение, память: Адр-Знач,
указатель_на_текущую_команду: Адр[Ком]>

Интерпретатор ассемблера различает следующие категории команд:

- обработка текста программы;
- вычисления с результатом в сумматоре, для которых следующая команда расположена по соседству;
- изменение части состояния общей памяти, при котором следующая команда расположена по соседству;
- управление, обладающее своими правилами свободного выбора следующей команды.

Система команд − фиксированный словарь, не изменяющийся при выполнении программы.

ассемблер = (Текст -> {Код | Адрес}): Пам [ячейка] -> Пам

```
LD HL, ADR1 откуда
LD DE, ADR2 куда
LD BC, 2048 сколько
LDIR
```

Рис. 1. Пример из ассемблера MSX: пересылка блока (ТД не имеет значения)

Таблица 2

Конкретизация понятий в языках ассемблера

Понятие	Ассемблер
Атом (Элементарное)	|слово|
Структура	блок, стек, ссылки, косвенность, индексы
Переменная	Ид -> Адр (Знач)
Значение	|Адр|, |Число|, |Код|

Выражение	|простые| одна операция, над небольшими числами=адресами
Действие/Операция	Ариф ПУ цикл-ветвл
Условие/истина	= 0/~=0 спец. Регистр шкала прерываний
Функция/ подпрограмма	адр. Подпр с ПУ в конце для возврата
Аргумент	Выделенные регистры – адреса
Вызов Фн/подпр	ПУ с точкой возврата
Определение Фн/подпр	Ид -> Адр (Подпр)
Идентификатор	Метка = Адрес

В качестве модели интерпретатора можно использовать определение интерпретатора SECD-машины (без регистра D) [34, 126].

3.2. Макропроцессоры

Для макропроцессоров семантика ЯНУ обычно сопровождает средства работы со строками в стиле открытых процедур [17]. Программа представляет собой поток макроопределений и макровызовов. Имена макросов могут рассматриваться равноправно с базовыми средствами. При определении и реализации макросов используется понятие позиции и шаблона для подстановки параметров. Возможен стиль нумерации позиций – можно обойтись без их именования в виде переменных. В результате подстановки макросов формируется текст, равноправный с исходным, возможно, содержащий вторичные макросы. Макропроцессор часто используется в паре с ассемблером (макроассемблер) и другими ЯП. В качестве опорных рассмотрены GPM, TRAC [17], а также два макропроцессора из системы подготовки программ на языке SETL [145].

Абстрактная машина макропроцессора может быть задана тройкой

<Результирующий_текст, Таблица_макросов,
указатель_на_текующую_позицию_в_исходном_тексте>

Интерпретатор макропроцессора при последовательном сканировании текста выделяет в нем следующие категории строк:

– макроопределение, которое следует поместить в таблицу макросов;

– макровызов без параметров, определение которого следует скопировать из таблицы в результат;
– макровызов с параметрами, значения которых следует установить, а затем подставить в определение, и преобразованное определение разместить в результат;
– простая строка, копируемая в результат;
– конец текста.

Макро = (Прог = Текст): Прог -> Прог'

```
§A, §Def, A , <D>; §def, B, <C>;;
```

Рис. 2. Пример макропрограммы на GPM. Моделирование « if A=B then C else D » обеспечено побочным эффектом на глобальной таблице имен

Таблица 3

Конкретизация понятий в языках макропроцессора

Понятие	Ассемблер	Макропроцессоры
Атом (Элементарное)	|слово|	символ - слово
Структура	блок, стек, ссылки, косвенность, индексы	строка - список,
Переменная	Ид -> Адр (Знач)	Имя, позиция в списке -> Ид (Знач)
Значение	|Адр|, |Число|, |Код|	строка
Выражение	|простые| – одна операция, над небольшими числами = адресами	§Макрос (а, в, ...) вызов макроса

Действие/ Операция	Ариф ПУ цикл-ветвл	Ввод определений, подстановка значения переменной
Условие/ исти- на	= 0/~=0 спец. регистр шкала прерываний	0 пустая строка
Функция/ под- програм- ма/макрос	адр. подпр с ПУ в конце для возврата	Макрос: Имя -> Ид (шаблон подстановки)
Аргумент	выделенные регистры – адреса	элемент списка при вызове макроса
Вызов Фн/подпр	ПУ с точкой возврата	замена вызова макроса на преобразуемый шаблон
Определение Фн/подпр	Ид -> Адр (Подпр)	Имя -> представление шаблона
Идентификатор	Метка = Адрес	номер в таблице определений – в словаре

Модель макропроцессора может быть определена как композиция функций подстановки текста типа Subst и Sublis, дополненная таблицами для хранения значений параметров макросов для GPM и функциями сопоставления образцов для TRAC.

3.3. Машинно-ориентированное программирование. Forth

Для стековых языков, таких как Forth [12], система численных вычислений распадается на подсистемы по величине обрабатываемого слова (16 и 32, возможно, 64). Основа работы с памятью – стек. Средства управления вычислениями обогащены средствами блокировки и кодирования программ, что позволяет повышать эффективность информационной обработки. Используется механизм замкнутых процедур с неявными – стековыми – параметрами. Стек реализован как указатель на текущий элемент в предположении, что перед ним по порядку расположены предшествующие элементы.

Абстрактная машина для языка Forth представляется тройкой

<Стек, словарь, указатель_на_текущее_слово_в_программе>

Программа – отдельный поток, использующий расширяемый словарь. Принята постфиксная запись, удобная для стековой обработки данных. Стек-ориентированная дисциплина обработки освобождает от необходимости в понятии «переменная», хотя оно при необходимости м.б. смоделировано.

Интерпретатор языка Forth сортирует слова по принадлежности словарю.

- не найденные в словаре слова записываются в стек для предстоящей обработки;
- словарным словам, встроенным в интерпретатор, соответствует правило преобразования стека;
- возможно определение новых слов, запоминаемое в словаре (от «:» до «;»);
- за корректность воздействий на стек отвечает программа;
- результатом считается состояние стека при завершении программы.

```
*
OVER
-
SWAP
DUP
*
+
```

Рис. 3. Программа на языке Forth для подсчета по формуле (x,y,z -> x**2 + y * z – x)

Forth: текст/словарь -> стек: стек -> стек'
: слово -> словарь

Таблица 4

Конкретизация понятий в машинно-ориентированном языке Forth

Понятие	Ассемблер	GPM	Forth
Атом (Элементарное)	|слово|	символ слово	Слово
Структура	блок, стек, ссылки, косвенность, индексы	Строка список,	Стек последовательность

Переменная	Ид -> Адр (Знач)	Имя, позиция в списке → Ид (Знач)	позиция в стеке Имя → Ид (Знач) в словаре
Значение	|Адр|, |Число|, |Код|	Строка	|код|
Выражение	|простые| одна операция над небольшими числами= адресами	§Макрос (а, в, ...) вызов макроса	Постфиксные
Дейст-вие/Операция	Ариф ПУ цикл-ветвл	Ввод определений, подстановка значения переменной	над стеком ариф
Условие/истина	0 спец. регистр шкала прерываний	0 пустая строка	0 на вершине стека
Функция/ подпрограмма	Адр. подпр с ПУ В конце для возврата	Макрос: Имя → Ид шаблон подстановки	Имя → Ид последовательность слов в словаре
Аргумент	Выделенные регистры – адреса	элемент списка при вызове макроса	позиция в стеке
Вызов Фн/подпр	ПУ с точкой возврата	замена вызова макроса на преобразуемый шаблон	результат в стеке
Определение Фн/подпр	Ид → Адр (Подпр)	Имя представле-ние шаблона	Имя → строка занесение в словарь
Идентификатор	Метка = Адрес	номер в таблице определений/в словаре	номер в словаре

Работа со стеком достаточно просто моделируется на списках интерпретатором, подобным SECD [34, 126].

3.4. Языки управления заданиями

На уровне операционной системы (ОС) информационная обработка выглядит как семейство взаимодействующих процессов, выполняемых по отдельным программам – заданиям или сценариям, размещенным в файлах. [72, 143] Языки для ОС работают с очередями, которые могут быть представлены как строки или файлы. В памяти хранится контекст задания и его сценарий. Контекст содержит перечень доступных файлов. При управлении процессами выполнения заданий используются условия готовности и вырабатываются сигналы, символизирующие успех выполнения действий. Сигналы также хранятся в контексте. Действия могут быть организованы в конвейеры или последовательности и обусловлены успехом предшествующих действий. Очередь может быть пополнена. В качестве опорного языка рассмотрен Bash [143].

Функционирование ОС обеспечивает следующие явления и критерии:

- порождение новых файлов и процессов по ходу дела;
- время жизни файлов и процессов произвольно – нет гарантий;
- неограниченная динамика событий;
- содержание может быть незавершенным;
- изменение содержания и состава;
- очередь процессов с условиями готовности.

Абстрактная машина языка заданий может быть определена четверкой:

< буфер_данных, контекст_процесса, текущий_процесс, очередь_отложенных_процессов>

Команды интерпретатора ОС выполняют обмен данными между буферами, файлами и контекстом, обработку очереди, файлов и контекста, проверку ряда условий над данными, контекстом и очередью, выработку сигналов, включая установку сигнала о завершении процесса.

```
$ ls doc[c-d]
$ set 0 noclobber
$ cat newletter1 newletter2 >! Oldletters
$ at 8:15 jobs
```

Рис. 4. Пример программы управления заданиями

Таблица 5

Конкретизация понятий в языке управления заданиями

Понятие	Ассемблер	GPM	Forth	ОС
Атом (Элементарное)	|слово|	символ слово	Слово	Строка имя файла, номер процесса, устройство
Структура	блок, стек, ссылки, косвенность, индексы	Строка список	Стек последовательность	файл, директория
Переменная	Ид → Адр (Знач)	Имя, позиция в списке → Ид (Знач)	позиция в стеке Имя → Ид (Знач) в словаре	Имя → Ид (Знач)
Значение	|Адр|, |Число|, |Код|	строка	|код|	Число строка
Выражение	|простые| одна операция, над небольшими числами= адресами	§Макрос (а, в, ...) вызов макроса	Постфиксные	Командная строка
Действие/ Операция	Ариф ПУ цикл-ветвл	Ввод определений, подстановка значения переменной	над стеком ариф	над файлами и процессами
Условие/ истина	0 спец. регистр шкала прерываний	0 пустая строка	0 на вершине стека	Успех Существование
Функция/ подпрограмма/ сценарий	адр. подпр с ПУ в конце для возврата	Макрос: Имя → Ид (шаблон подстановки)	Имя → Ид (последовательность слов) - в словаре	Имя → Ид (подпр) I/O, Error

Аргумент	выделенные регистры адреса	элемент списка при вызове макроса	позиция в стеке	поля в командной строке < > | &
Вызов Фн/подпр	ПУ с точкой возврата	замена вызова макроса на преобразуемый шаблон	результат в стеке	Процесс -> < протокол, код завершения >
Определение Фн/подпр	Ид -> Адр (Подпр)	Имя → представление шаблона	Имя → строка занесение в словарь	Имя → подпр Данное смена статуса файла
Идентификатор	Метка = Адрес	номер в таблице определений - в словаре	номер в словаре	имя файла имя переменной

Представление процессов и файлов унифицировано, что видно по сопоставлению команд по обработке файлов и манипулированию процессами:

Таблица 6

Параллелизм команд над файлами и процессами

	Файлы	Процессы
Вывести список всех	Ls	Ps
Копировать	Cp	Bg Fg
Удалить	Rm	Kill
Сцепить последовательно	Cat	| конвейер
Показать контекст/задания	Env	Jobs
Создать новый файл/процесс	Cp – в новый файл	Fork
Копировать на старое место	Cp – в старый файл	Exec
Выбрать/Ждать	Указатель	Wait
Вхождение строки/ статус процесса	Поиск строки в файле	Проверка завершения-готовности процесса Notify

/переход от процесса к его результату		Eval
Формат файла/процесса	Таблица строк/записей	Таблица список команд с параметрами

Построение модели языка управления заданиями требует дополнительных операций по работе с очередями, что может быть устроено как «ленивый» список, в конец которого можно встраивать новые элементы функцией Conc.

3.5. Машинно независимые языки

Для разнообразия можно рассмотреть и другие ЯНУ, например, язык Эпсилон, ориентированный на обработку векторов, и язык Little, предназначенный для обработки битовых строк [116].

Little

```
X = 23
.f. 1,10, a (2)  = 44  / *  в поле с 1 по 10-й разряд записать 44 */
.ch. 5, text = ‘A’      /*   в 5-й символ текста  записать ‘A’ */
```

Рис. 5. Примеры операторов присваивания в языке Little

Текст → acc : бит-стр → бит-стр'

Эпсилон

```
Начало
  Слово A [4:2]; константа B = ‘3’;
  A [2]  := B;
  A [1]  :=  A [2];
  стоп
Конец
```

Рис. 6. Пример программы на языке Эпсилон

Текст → {Код | Адр} : Пам [Адр] → Пам

Таблица 7

Сравнение конкретизации понятий
в машинно-ориентированных языках общего назначения

Понятие	Ассемблер	Эпсилон	Little	Forth
Атом (Элементарное)	|слово|	|Имя|	Ключ.слова, имя	Слово
Структура	блок, стек, ссылки, косвенность, индексы	Вектор список [8], Строка	Вектор, разбивка кода на поля	Стек, последова-тельность
Переменная	Ид → Адр (Знач)	|Имя| → Ид (Знач)	Имя → Ид (Знач)	позиция в стеке, Имя → Ид (Знач) в словаре
Значение	|Адр| |Число| |Код|	|код|	бит.строка	|код|
Выражение	|простые| одна операция, над небольшими числами= адресами	Простые без вложенности	Как фFortran	Постфиксные
Действие/ Операция	Ариф ПУ Цикл ветвл	Ариф , Выборки из строк и списков, Обработка кодов	Выбор поля из бит.стр, упр как в фFortrane	над стеком ариф
Условие/истина	0 спец. Регистр шкала пре-рываний	«истина»	0	0 на вершине стека

[8] «Вектор» называется «список»

Функция/ подпрограмма	адр. подпр с ПУ в конце для возврата	Имя → Ид (подпр)	Имя → Ид (подпр)	Имя → Ид последовательность слов в словаре
Аргумент	выделенные регистры – адреса	элемент списка при вызове подпр	элемент списка при вызове подпр	позиция в стеке
Вызов Фн/подпр	ПУ с точкой возврата	Подпр (а, в, …) подстановка	Подпр (а, в, …)	результат в стеке
Определение Фн/подпр	Ид → Адр (Подпр)	Имя → подпр Замкнутые и открытые	Имя → подпр	Имя → строка занесение в словарь
Идентификатор	Метка Адрес	Метка, Имя	Метка, имя	номер имени в словаре

Выводы:

- Определение парадигм для ЯНУ не вызывает затруднений – в них явно видна ключевая идея, и семантические системы сравнительно изолированы в определении языка.
- Основные различия сосредоточены на конкретизации понятия «значение» и спектра средств укрупнения осмысленных единиц при подготовке программы.
- Реализационная семантика ЯНУ, как правило, требует введения дополнительных понятий (очередь, логика, словарь, точка возврата, позиция в стеке, шкала прерываний и т.п.), возникающих на уровне схем программ и программисткой терминологии.
- Функциональные модели ЯНУ достаточно просты. По уровню сложности они проще интерпретатора SECD, т.к. не гарантируют защиту контекста. Отличаются составом команд.

4. ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Программирование на языках высокого уровня (ЯВУ) приспособлено к представлению иерархии понятий, отражающей природу понимания человеком решаемых задач и организации процессов их решения [5, 9, 11, 14, 15, 20–24, 26, 30, 36, 41, 48, 49, 61, 63, 68, 71, 78, 80, 91, 100–111, 127]. Ис-

пользуются сложные структуры данных, стереотипы техники программирования, локализуемые области видимости имен объектов и процедур их обработки, подчиненные структурно-логической модели управления, допускающей сходимость пошагового процесса отладки программ [47, 73, 77, 125, 133]. Результативны графические интерфейсы и компонентные технологии, поддерживающие перенос отлаженных результатов в разные системы. В центре внимания – интеграция с библиотеками процедур, эффективная компиляция программ, контроль типов данных, соответствие стандартам области применения программ и технологиям быстрой разработки удобно сопровождаемых программ. Ряд проблем решается включением в ЯВУ низкоуровневых средств. Практически исчезает необходимость в блок-схемах, а методика самодокументирования и реализации справочных подсистем смягчает роль документирования [62, 121]. Программе на ЯВУ обычно соответствует семейство допустимых процессов, определение которого представлено формальной семантикой языка. Система программирования (СП), поддерживающая ЯВУ, как правило, порождает один из процессов этого семейства. Такое сужение диктуется необходимостью воспроизведения процессов при отладке программ.

Текст программы на ЯВУ обычно обретает билинейность – линия представления процесса обработки данных в нем совмещена с описанием типов данных. Возникает нечто вроде пространственной аппроксимации процесса, используемой при проверке корректности программ − статический или динамический контроль типов данных. Проработка понятий ЯВУ характеризуется пропорциями между императивностью – аппликативностью, пространствами константных и переменных значений, элементарных и составных данных, открытыми и замкнутыми процедурами, средствами обработки строк и файлов, возможностями активного и «ленивого» управления процессами, использованием последовательных и параллельных схем управления вычислениями. Все ЯВУ используют стек при реализации укрупненных конструкций и защите локализуемых данных. Стиллистика ЯВУ тесно связана со структурами данных, алгоритмами, методами синтаксического анализа и компиляции программ с акцентом на критерии теории программирования [2, 10, 19, 64, 76, 106–108, 113, 127, 129].

Обычно высокий уровень языка обеспечивается программными средствами, но с появлением программаторов и микропрограммирования разница между программой и аппаратурой стала условной. Lisp, Pascal, Smalltalk, Algol и другие ЯВУ были реализованы как входные языки на правах машинного кода [31, 156]. Ассемблер Эльбрус – яркий пример отечественной реализации ЯВУ аппаратными средствами [100, 101].

При анализе парадигм ЯВУ необходимо учитывать следующие их особенности:

- практикуются неявные формы представления отдельных понятий ради лаконизма записи программ;
- выражения чаще всего рассчитаны на схему предвычисления над скалярами разной длины или сложными значениями;
- разнообразны виды ветвлений, циклов, функций и процедур;
- типы данных конструируются по фиксированным в ЯП правилам;
- схемы управления фиксированы в языке и конкретно реализованы в системе программирования;
- взаимодействие и соответствие средств и методов, относящихся к разным семантическим системам, при их реализации в системе программирования определено по традиции и прецедентам, причем в коде оно скрыто – не структурировано;
- эффективность программирования базируется на знании методов реализации значений и обработчиков структур данных в памяти.

Таблица 8

Названия некоторых ЯВУ, относящихся к разным парадигмам программирования

СП	ФП	ЛП	ООП	УЯ
Fortran Algol Альфа C Modula Oberon Эльбрус ЯРМО Occam YACC Lex	Lisp ML Рефал Python Cmucl Erlang Haskell Interlisp MuLisp Scheme Hope Clean Dylan Miranda	Planner Snobol Prolog	Simula-67 Smalltalk-80 C++ Eiffel Clos Java	Pascal Basic Grow Logo Робик Рапира

(СП – стандартное/системное программирование, ФП – функциональное/аппликативное программирование, ЛП – логическое/декларативное

программирование, ООП – объектно-ориентированное программирование, УЯ – учебные языки программирования).

В сравнении с языками низкого уровня семантика ЯВУ содержит арифметику, разделенную на ряд систем обработки целых и вещественных чисел в разных диапазонах, обычно зависящих от разрядности адресов и машинных слов. Имеются системы работы с указателями, символами и строками и система конструирования типов данных. Поддерживается набор стандартных операторов управления вычислениями – ветвления, циклы и вызовы процедур/функций. А главное – кроме работы с глобальными объектами используются разные схемы локализации хранимых в памяти объектов, что обеспечено организацией структур данных по принципу иерархии.

Учебный концентр ЯВУ можно ограничить одной или двумя арифметическими системами. Элементарные уровни опорных ЯВУ разных парадигм можно описать как расширения или конкретизации такого концентра.

При анализе парадигм ЯВУ следует отметить их обусловленность эволюцией технологий системного, императивно-процедурного программирования. Первый ЯВУ Fortran связан с появлением раздельной компиляции, снизившей трудозатраты на отладку благодаря выделению техники сборки модулей, что обеспечило формирование общих библиотек, а заодно и устойчивость позиций языка Fortran до наших дней [69]. Универсальный язык Lisp дал ход машинно-независимому стилю обработки данных, рассматриваемых как символьные списки или строки, что привело к парадигме функционального программирования [31, 40, 55, 63, 82, 111, 136, 139, 142, 148–152]. Популярность языка С связана с технологией машинно-зависимого переноса программ путем выделения в двухслойном языке компактного машинно-ориентированного ядра и самоописания Си-компилятора, что одновременно обеспечило перенос на множество архитектур и стыковку с близкими языками, компилируемыми на тот же уровень сборки модулей [16, 24]. Логическое программирование на языке Prolog показало возможности накопления и наследования частных рецептов [54, 92]. Появление С++ и ООП сопряжено с расширением сферы приложения информационных систем, при конструировании которых необходимо минимизировать объем повторного программирования [24, 51, 94, 118, 155]. Представление специфических деталей парадигм при сравнении ЯВУ можно выразить в форме нормализованной предикатной формы интерпретатора, различающего сквозные понятия сравниваемых языков.

В пробном описании парадигм СП, ФП, ЛП и ООП в качестве опорных языков используются C, Pure Lisp, Prolog и C++/Clos [9].

4.1. Стандартное программирование

Стандартное, императивно-процедурное программирование (СП) рассматривает процесс обработки информации как конечную последовательность локальных изменений состояния памяти – императивно-процедурный стиль. Для СП характерно четкое разделение понятий “программа” и “данные” с учётом статических методов контроля типов данных и оптимизации программ при компиляции. Общий механизм интерпретации стандартной программы естественно представить как автомат с отдельными таблицами имен для переменных, значения которых подвержены изменениям, и для меток и процедур, определения которых неизменны.

Абстрактная машина (АМ) может рассматриваться как развитие и обобщение АМ для ассемблера, обусловленное возможностью использовать более сложные структуры данных в качестве регистров:

<сумматор: значение, память: Адр-Знач, текущая_команда>

Сумматор расширяется до стека промежуточных значений, к памяти присоединяется регистр «Локалы» для локализации хранимых объектов, появляется дополнительный регистр для хранения копий состояний памяти, которые могут понадобиться для восстановления контекста вычислений при ветвлениях и процедурных вызовах:

<Стек_ значений, (Локалы, Память), Текущая_команда, Копии>

Регистр «Локалы » может быть устроен подобно регистру «E» из SECD-машины Лендина [126], только хранит он не значения, а ссылки на них в «Памяти». Начальное состояние – вектор глобальных переменных, адресов подпрограмм и меток.

АМ различает следующие категории команд:

- засылка значений из памяти в стек;
- вычисления над безымянными операндами в стеке при обработке выражений;
- пересылка значений из стека в память с учетом локализации;
- организация переходов по метке в программе;
- организация ветвлений и циклов;

[9] CLOS – Объектно-ориентированная библиотека функций для Common Lisp.

– организация вызовов процедур и функций с сохранением/восстановлением контекста.

При интерпретации планируется распределение памяти для значений переменных в зависимости от их типов данных, выполняется размещение локальных данных в памяти, частичный контроль доступа к переменным и совместимости операций и операндов по типам данных, вычисление значений выражений (констант, переменных, элементов структур данных, результатов операций и вызовов функций), манипуляции по управлению вычислениями.

СП = (Текст → {Код | Адрес}): Пам [Переменная] → Пам

Таблица 9

Конкретизация понятий в стандартных императивно-процедурных языках на примере языка С

Понятие	Ассемблер	СП: Си
Атом (Элементарное)	|слово|	Имя |Скаляр|
Структура	Блок Стек Ссылки Косвенность, Индексы	Вектор, Строка, Структура = Запись, Объединение
Переменная	Ид → Адр (Знач)	Ид → Адр (Знач)
Значение	|Адр|, |Число|, |Код|	Адрес, |Скаляр|
Выражение	|простые| одна операция, над небольшими числами= адресами	Произвольные, со скобками и приоритетами
Действие/Операция	Ариф ПУ Цикл Ветвление	Арифметика , Доступ к элементам структур, Присваивания, Работа со строками. Формирование ТД
Условие/истина	= 0/~=0 спец. Регистр шкала прерываний	0

Функция/ оператор	адр. Подпр с ПУ в конце для возврата	Организация схем управления Поствычисления в операторах
Аргумент	Выделенные регистры – адреса	Локальные переменные на стеке
Вызов Фн/подпр	ПУ с точкой возврата	Организация новой области имен и исполнение определения подпрограммы – блок
Определение Фн/подпр	Ид → Адр (Подпр)	Пополнение таблицы функций
Идентификатор	Метка = Адрес	Адрес метки, константы, переменной, функции

```
Main (argc, argv, envp)
 Int argc;
 Char **argv;
 Char **envp;
{
For (i=0; i < argc; i++)
      Printf (”arg%i:%s\n”, i, argv [i]);
For (p=0; *p != (char*)0; p++)
      Printf (”%s\n”, *p);
}
```

Рис. 7. Пример.
Программа распечатки параметров командной строки и переменных среды на языке Си [16]

Prog-форма языка Lisp 1.5 [142] может рассматриваться как абстрактная модель СП, в которой при интерпретации используются отдельные ассоциативные таблицы для хранения параметров доступа к значениям переменных и определениям функций и процедур в памяти. Это позволяет независимо задавать механизмы доступа к именам переменных и наименованиям процедур.

В дальнейшем в качестве «опорных» предполагается привлечь языки Fortran, Альфа, С, Modula, ЯРМО [69, 11, 16, 22, 23, 31].

Сложность разработки больших программ, функционирующих в стиле локальных изменений состояния памяти, привело к идеям структурного программирования, налагающим на стиль представления программ ряд ограничений, способствующих удобству отладки программ и приближающих технику стандартного программирования к функциональному программированию [53]:

- дисциплина логики управления с исключением переходов по меткам (goto_less_style);
- минимизация использования глобальных переменных в пользу формальных параметров процедур (global_variable_harmful);
- полнота условий в ветвлениях, отказ от отсутствия ветви “else”;
- однотипность результатов, полученных при прохождении через разные пути.

4.2. Функциональное программирование

Функциональное программирование рассматривает процесс обработки данных как композицию их отображений с помощью универсальных функций [123, 126, 139]. Программа при таком подходе не более чем одна из разновидностей данных. Функциональное программирование сумело преодолеть синтаксический разнобой независимо развиваемых средств и методов организации информационных процессов. Это удалось благодаря нацеленности на проявление и ортогонализацию семантических подсистем в организации программируемых процессов. Не менее важна развиваемость представления унифицируемых структур данных и комплектов функциональных объектов [79]. Все это позволяет языки функционального программирования рассматривать как средство сопряжения разнородных конструкций. На их основе возможно осуществлять любые интегрированные построения, представимые в машинно-независимом стиле.

Языки функционального программирования используют гибкие списки и абстрактные атомы, нацелены на полный контроль типов значений при исполнении программ, допускающих динамический анализ и управление с откладыванием вычислений, автоматизацию повторного использования памяти – «сборки мусора».

Язык программирования Lisp [142] и сложившееся на его основе функциональное программирование реально показали свои сильные стороны более чем убедительно как инструментарий исследования и освоения новых областей применения вычислительной техники. Это аргумент в пользу функционального подхода к решению новых сложных задач:

- доступны преобразования программ и процессов;
- осуществима лингвистическая обработка информации;
- поддерживается гибкое управление базами данных;
- возможна оптимизация информационных систем, вплоть до полного исключения потерь информации;
- моделирование общения;
- и многое другое.

Во многих системах функционального программирования обеспечена организация параллельных процессов, имеются средства стандартного и объектно-ориентированного программирования. Поддержано управление компиляцией и конструирование компиляторов. Существенно, что операции функционального языка обладают машинно-независимой семантикой. Поэтому для функциональных программ проблема переноса стоит намного мягче, чем для стандартных языков, таких как C, Pascal, Fortran [16, 49, 69]. Это делает функциональные ЯП перспективными для образовательных применений. Успех Logo в детском и любительском программировании не случаен [55].

```
(defun map-comp (fn al vl)          ; fn покомпонентно применить
                                    ; к соотвественным элементам AL и VL
   (cond
        (xl (cons (funcall fn (car al) (car vl))
                            ; Вызов данного FN как функции от элементов двух
списков
                  (map-comp (cdr al) (cdr vl))
)   ) ) )
(map-comp #'+ '(1 2 3) '(4 6 9))
                            ; = (5 8 12 ) Суммы
(map-comp #'* '(1 2 3) '(4 6 9))
                            ; = (4 12 27 ) Произведения
(map-comp #'cons '(1 2 3) '(4 6 9))
                            ; = ((1 . 4)(2 . 6)(3 . 9)) Пары
(map-comp #'eq '(4 2 3 ) '(4 6 9))
                           ; = (T NIL NIL ) Сравнения
```

Рис. 8. Определение функции покомпонентной обработки двух списков с помощью заданной функции. Покомпонентные действия над векторами, представленными с помощью списков, списки сумм, произведений, пар и результатов проверки на совпадение

```
(DEFUN evl(e a) (COND
                    ( (atom e)              (cdr(assoc e a)) )
                    ( (eq (car e) 'QUOTE)  (cadr e))
                    ( (eq(car e) 'IF)       (evcon(cdr e) a))
                    ( T              (apply (car e) (evlis(cdr e) a) a) ))    )
(defun apply (fn x a) (COND
            ((atom fn)(cond
                    ((eq fn 'CAR)   (caar x))
                    ((eq fn 'CDR)   (cdar x))
                    ((eq fn 'CONS) (cons (car x)(cadr x)) )
                    ((eq fn 'ATOM)(atom (car x)) )
                    ((eq fn 'EQ)     (eq (car x)(cadr x)) )
                    (T              (apply (evl fn a) x a)) ) )
             )
((eq(car fn)'LAMBDA) (eval (caddr fn)
                           (pairlis (cadr fn) x a) ))
((eq (car fn) 'LABEL) (apply (caddr fn) x
                        (cons (cons (cadr fn)(caddr fn)) a)))))
(DEFUN evcon (c a) (COND
                    ((evl (car  c) a) (evl (cadr c) a) )
                    ( T  (evl (caddr c) a)    ) ))
(DEFUN evlis (m a) (COND
                        ((null m) Nil )

                    ( T        (cons(evl (car m) a)
                                    (evlis(cdr m) a)   ))  )
```

Рис. 9. Самоописание интерпретатора, предложенное Дж. Маккарти в начале 1960-х годов [142]

Функциональные языки программирования достаточно разнообразны. Существует и активно применяется более трехсот диалектов языка Lisp: Interlisp, muLisp, Clisp [140], Sheame, Cmucl [150], Logo и родственных ему языков: ML, Hope, Clean, Erlang [151], Dylan [152], Haskell [148], Miranda и др. При сравнении языков и парадигм программирования часто классифицируют функциональные языки более детально по следующим критериям: “ленивый” или аппликативный, последовательный или параллельный. Например, ML является аппликативным и последовательным, Erlang – аппли-

кативным и параллельным, Haskell – “ленивым” и последовательным, а Clean – параллельным и “ленивым”.

Таблица 10

Конкретизация понятий
в универсальном языке программирования Lisp

Понятие	Forth	СП: C	ФП: Lisp
Атом (Элементарное)	Слово	Имя, |Скаляр|	Атом, Числа и строки без ограничений на длину
Структура	Стек последовательность	Вектор, Строка, Запись, Объединение	Список
Переменная	позиция в стеке Имя → Ид (Знач) в словаре	Ид → Адр (Знач)	Атом → ассоциативный список
Значение	|код|	Адрес |Скаляр|	Данное
Выражение	Постфиксные	Произвольные, со скобками и приоритетами	Переменная, Префиксное в виде списка, начинающегося с функции
Дейст-вие/Операция	над стеком ариф	Арифметика , Доступ к элементам структур, Присваивания, Работа со строками. Формирование ТД	Обработка списков, Арифметика
Условие/истина	0 на вершине стека	0	Не NIL
Функция/ оператор	Имя → Ид последо-вательность слов в словаре	Организация схем управле-ния Поствычисле-ния в операторах	SUBR – подпро-грамма, EXPR – выражение

Аргумент	позиция в стеке	Локальные переменные на стеке	Локальные переменные в ассоциативном списке
Вызов Фн/подпр	результат в стеке	Организация новой области имен и исполнение определения попрограммы – блок	Пополнение ассоциативного списка
Определение Фн/подпр	Имя → строка - занесение в словарь	Пополнение таблицы функций	Пополнение ассоциативного списка
Идентификатор	номер в словаре	Адрес метки, константы, переменной, функции	Атом

4.3. Логическое программирование

Логическое программирование (ЛП) сводит обработку данных к выбору произвольной композиции определений (уравнений, предикатных форм), дающей успешное получение результата [126, 92]. Именно обработка формул является основой – вычисления рассматриваются как операция над формулой. При неуспехе происходит перебор вариантов определений. В языках логического программирования считают возможным прямой перебор вариантов, сопоставляемых с образцами, и организацию возвратов при неудачном выборе. Перебор вариантов выглядит как обход графа в глубину. Имеются средства управления перебором с целью исключения заведомо бесперспективного поиска.

В отличие от множества элементов набор вариантов не требует одновременного существования всех составляющих. Поэтому программирование вариантов можно освободить от необходимости формулировать все варианты сразу. В логическом программировании можно продумывать варианты отношений между образцами формул постепенно, накапливая реально встречающиеся факты и их сочетания. Содержательно такой процесс похож и на уточнение набора обработчиков прерываний на уровне оборудования. Кроме основной программы, выполняющей целевую обработку данных, отлаживается коллекция диагностических реакций и процедур продолжения счета для разного рода неожиданных событий, препятствующих получению результата программы.

Если варианты в таком выражении рассматривать как равноправные компоненты, то не ясно, как предотвратить преждевременный выбор пустого списка при непустом перечне вариантов. Чтобы решить эту задачу, в теории ЛП вводится специальная форма **ESC (ТУПИК)**, действие которой заключается в том, что она как бы "старается" **по возможности не исполняться**. Иными словами, при выборе вариантов предпочитаются варианты, не приводящие к исполнению формы ESC. Такая же проблема возникает при обработке пустых цепочек в грамматиках. Аналогичная проблема решена при моделировании процессов интерпретированными сетями Петри [74, 87] соглашением о приоритете нагруженных переходов в сравнении с пустыми.

Уточненное таким образом определение выбора произвольного элемента списка можно представить формулой вида:

```
(любой L) = { (car L)
            | (любой  (cdr L))
            | ESC  }
```

Рис. 10. Пример: Определение функции, выбирающей произвольный элемент списка

В какой-то момент L становится пустым списком, и его разбор оказывается невозможным, тогда действует вариант ESC.

Следует иметь в виду, что варианты не образуют иерархии. Их аксиоматика подобна так называемой упрощенной теории множеств [138]. Принципиальная особенность – совпадение предикатов принадлежности и включения.

Интерпретирующий автомат для выполнения недетерминированных процессов можно представить как цикл продолжения вычислений при попадании в диагностическую ситуацию. Продолжение заключается в выборе другого варианта из набора определений функционального объекта. Вычисление признается неудачным, лишь если не удалось подобрать комплект вариантов, позволяющий вычислить значение формулы.

s e c d r → s' e' c' d' r' - переход от старого состояния к новому.

Таблица 11

Конкретизация понятий в языках логической парадигмы на примере языка Prolog

Понятие	Макропроцессоры	ЛП: Prolog
Атом (Элементарное)	символ слово	Слово
Структура	строка список	Строка
Переменная	Имя позиция в списке → Ид (Знач)	Именованная позиция в шаблоне
Значение	строка	Строка
Выражение	§Макрос (а, в, ...) – вызов макроса	Строка с вхождениями переменных
Дейст-вие/Операция	Ввод определений, подста-новка значения переменной	подстановка значения переменной
Условие/истина	0 пустая строка	
Функция/ уравнение	Макрос: Имя → Ид (шаблон подстановки)	Вычисление или подстановка, управляемые сопоставлением шаблонов
Аргумент	элемент списка при вызове макроса	Строка, сопоставленная по шаблону
Вызов Фн/подпр	замена вызова макроса на преобразуемый шаблон	Подстановка аргументов из шаблона в его определение
Определение Фн/подпр	Имя → представление шаблона	Шаблон (имя, …) = ветвь (имя,…)
Идентификатор	номер в таблице определений в словаре	Имя

```
Factorial (N, F)
  Factorial (0, 1)
  Factorial (x+1, F)  ← F := Factorial (x, y) * (x + 1)
```

Рис. 11. Пример: Вычисление факториала

4.4. Объектно-ориентированное программирование

Объектно-ориентированное программирование рассматривает информационный процесс как частичную обработку объектов с помощью методов, выбираемых в зависимости от типа обрабатываемых данных. ООП структурирует множество частных методов, используемых в программе, в соответствии **с иерархией классов объектов**, обрабатываемых этими методами, в предположении, что определяемые в программе построения могут локально видоизменяться при сохранении основных общих схем информационного процесса. Это позволяет выполнять модификации программы объявлением новых подклассов и дописыванием методов обработки объектов отдельных классов без радикальных изменений в ранее отлаженном тексте программы.

Связь методов с классами объектов позволяет вводить одноименные методы над разными классами объектов (**полиморфизм**), что упрощает логику управления: на уровне текста программы можно не распознавать принадлежность классу: это сделает система программирования. Таким образом обычно реализовано сложение[10], одинаково изображаемое для чисел, строк, векторов, множеств и т.п.

Чтобы сравнить дистанцию ООП с ФП П.Грехем (Paul Graham) в описании стандарта языка Common Lisp предлагает рассмотреть модель встроенного в Lisp объектно-ориентированного языка (ОО-язык), обеспечивающего основы ООП [140]. Встраивание ОО-языка показывает характерное применение ФП – моделирование разных стилей программирования (начиная со стандартного программирования в виде prog-формы, предложенной Дж. Маккарти). В языке Lisp есть разные способы размещать коллекции свойств. Один из них – представлять объекты как хэш-таблицы и размещать свойства как входы в нее. В [140] приведен пример (8 строк) реализа-

[10] И другие операции.

ции ООП на базе хэш-таблиц. Фактически наследование обеспечивает единственная особенность языка Lisp: все это работает благодаря реализации рекурсивной версии GETHASH. Впрочем, Lisp по своей природе изначально был ОО-языком. Определение методов может достичь предельной гибкости благодаря возможности генерировать определения функциональных объектов с помощью DEFMACRO [140] или функцией категории FSUBR [142].

Другая модель ООП, полученная на базе обычных списков свойств (атрибутов), также иллюстрирует глубинное родство ФП и ООП [34]. Нужно лишь уточнить определение Lisp-интерпретатора, чтобы методы рассматривались как особая категория функций, обрабатываемая специальным образом.

Показанный в [140] пример Грехема работает по первому аргументу (выбор подходящего метода рассчитан на то, что достаточно разобраться с одним аргументом), CLOS делает это более обще на всех аргументах, причем с рядом вспомогательных средств, обеспечивающих гибких перебор методов и анализ классов объектов. **CLOS** естественно использует модель обобщенных функций, но у Грехема описана независимая модель, используя более старые представления, тем самым показано, что концептуально ООР – это не более, чем перефразировка идей языка Lisp. ООП – это одна из вещей, которую Lisp изначально умеет делать. Для функционального стиля программирования нет ничего неожиданного в переходе к ООП. Это просто небольшая конкретизация механизма перебора ветвей функциональных объектов.

Вопрос, что еще может дать функциональный стиль и традиция реализации функциональных систем программирования?

Ответ напрашивается сам собой: систему мер для сравнения языков и их реализационной семантики.

Абстрактная машина для ОО-языка по структуре похожа на машину для ЛП.

s e c d r → s' e' c' d' r' – переход от старого состояния к новому.

Основное отличие – вместо перебора вариантов работает выбор метода в зависимости от типа данных.

Таблица 12

Конкретизация понятий в языках ООП на примере языка C++ и системы CLOS

Понятие	ОС	ООП: C++ Clos
Атом (Элементарное)	Строка имя файла, номер процесса, устройство	Имя объекта, поля, метода, Скаляр
Структура	файл, директория	Класс с разметкой доступа
Переменная	Имя → Ид (Знач)	Имя поля, Имя → Ид (Знач)
Значение	Число строка	|скаляр|, Объект
Выражение	Командная строка	Объект.Метод
Дейст-вие/Операция	над файлами и процессами	Над скалярами, Доступ к полю, Применение метода
Условие/истина	Успех Существование	
Функция/ метод/ оператор	Имя → Ид (подпр) Объект I/O, Error	Имя → Ид (подпр), Локализованы по иерархии классов
Аргумент	поля в командной строке < > | &	Локальные встеке, Поле объекта
Вызов Фн/подпр	Процесс → < протокол, код завершения >	Выбор метода по ТД, принадлежности к классу
Определение Фн/подпр	Имя → подпр Данное смена статуса файла	Пополнение списка методов для класса
Идентификатор	имя файла имя переменной	Адрес объекта, поля, переменной

```
//HELLO.CPP
#include <iostream.h>
Void main ()
{ cont << ”\nHello, World!\n” ;
}
```

*Рис. 12. П*ример: Программа на языке С++

Таблица 13

Специфика основных семантических систем в разных парадигмах программирования на ЯВУ

	Выч	Пам	Упр	Структ
СП	Скаляры = слова СД-ТД предвычисления	Стат.распред := Области видимости – дин вызовов – статика вложенности текста – глоб Описания Категории имен составные имена exprt-import	Goto If Call Case Select Switch For While Until Do Trap Catch Ситуации ОС	Array Record Union Pointer
ФП	Длинные скаля-ры Списки Программы Пред и пост lazy	GC – FS Без описаний До исчерпания Дин вызовов – БД (атомы) Списки свойств Let Declare	Quote-eval Lazy Cond Lambda Defun Evalquote Ловушки Мультизна-чения Параллелизм Необязатель-ные параметры	Списки Файлы Строки Потоки Модели массивов, таблиц, вариантов

ЛП	Варианты с возвратами Шаблоны в строке Подстановка	Накопление рецептов	Обход лабиринта Параллелизм – обход графа в ширину	Иерархия принятия решений
ООП	Обработка полей записи	Разметка доступа Дружественный доступ	Выбор метода по ТД или по ситуации Привязка к интерфейсу	Иерархия классов

Выводы:

- Описания основных парадигм ЯВУ несложно моделируются средствами ФП. Основные различия связаны с вариациями дисциплины доступа к отдельным категориям имен.
- Можно выделять ведущую парадигму ЯП и ряд дополнительных парадигм.
- В «опорных» ЯП можно выделять фрагменты, полностью соответствующие одной парадигме.
- Фрагменты, соответствующие дополнительной парадигме ЯП, могут выполнять роль модели этой парадигмы при сравнении с другими языками.
- Реализационная семантика парадигм ЯВУ зависит от отношений на основных множествах сигнатур семантических систем, которые проще выражать в денотационной форме – денотационная проекция.
- При определении языка Lisp Дж. Маккарти уделил внимание аксиоматическим определениям специфики базовых операций над символьными выражениями. Возможно элегантная строгость таких определений повлияла на надежность и гибкость ФП.
- Структурное программирование сближает стандартное программирование с ранее возникшим функциональным, которому посвящен следующий раздел.
- Следует обратить внимание на появление учебных языков программирования (A++), поддерживающих четыре основных парадигмы программирования, что показывает их взаимную дополнительность и образовательное значение.

5. ЯЗЫКИ СВЕРХВЫСОКОГО УРОВНЯ

Подготовка программ на базе языков сверхвысокого уровня (ЯСВУ) нацелена на длительный срок жизни запрограммированных решений особо важных и сложных задач [7, 56, 83, 74, 89, 57, 137]. Удлинение жизненного цикла достигается представлением обобщенных решений с определенной степенью свободы по отношению к полным пространствам допустимых смежных компонент, реализованных ранее или планируемых на будущее. Реализационное сужение семейства процессов, допускаемых семантикой ЯСВУ, противоречит его целям или концепциям по следующим прагматическим мотивам:

- высокий уровень абстрагирования программируемых решений;
- решаются задачи, зависящие от непредсказуемых внешних факторов;
- базовые средства и/или алгоритмы вычислений используют параллелизм;
- актуальны прагматические требования к темпу и производительности вычислений;
- эксплуатируются динамически реконфигурируемые многопроцессорные комплексы.

Обычно создатели нового ЯСВУ используют в качестве исходного материала один или несколько базовых ЯВУ и встраивают в них изобретаемые средства и методы. От базовых ЯВУ наследуются парадигмы удовлетворительного решения сопутствующих задач. В таких случаях парадигматическая характеристика ЯСВУ может формулироваться относительно базовых ЯВУ, хотя внешнее синтаксическое сходство языковых конструкций иногда скрывает совсем другую семантику.

Для ЯСВУ характерно применение регулярных, математически ясных и корректных, абстрактных структур, при обработке которых возможны преобразования данных и программ, использование подобий и доказательных построений [27, 45], гарантирующие высокую производительность вычислений, надежность процесса разработки программ и длительность их жизненного цикла. Типичны алгебраические спецификации, теоретико-множественные построения, параллелизм, модели процессов разработки программ. Изобретаются специальные системные средства, повышающие емкость представлений, их общность и масштабируемость. Естественный резерв производительности компьютеров – параллельные процессы. Их организация требует контроля и детального учета временных отношений и

неимперативного стиля управления действиями. Суперкомпьютеры, поддерживающие высокопроизводительные вычисления, нуждаются в особой технике системного программирования, которая еще не сложилась, хотя уже имеется опыт эффективного решения особо важных задач [157].

Результативен графово-сетевой подход к представлению систем и процессов для параллельных архитектур, получивший выражение в специализированных языках параллельного программирования и суперкомпиляторах, приспособленных для отображения абстрактных структур управления процессами уровня задач на конкретную пространственную структуру процессоров реального оборудования [119]. Появляются языковые средства поддержки полного жизненного цикла программ, совмещенного с процессом изучения класса решаемых задач [65, 117, 120]. Для простоты изучения ЯСВУ могут выглядеть как расширение привычных ЯВУ, возможно со средствами низкоуровневого управления процессами. Характерна ортогональность семантических систем, варьирование методов реализации и обобщение схем управления процессами, включая разработку необычных/непривычных средств.

Здесь рассмотрены лишь ключевые идеи ряда ЯСВУ, ориентированных на параллельное программирование, без описания деталей реализационной семантики. При переходе к КЯ предстоит рассмотреть языки разработки, тестирования, отладки, спецификации и верификации программ, СУ-конструирование, средства и методы оптимизации программ и компонентного программирования, а также ряд новых технологий разработки распределенных информационных систем [88].

5.1. Параллельное программирование. APL

Параллельное программирование на языке APL можно рассматривать как укрупнение единиц обработки − распространение методов стандартного программирования со скалярных значений на векторы произвольной размерности [89]. Каждая команда может локально изменять не просто значение, а целый вектор. Синтаксис языка APL создан независимо, без оглядки на традиции представления программ и алгоритмов. Реализационная семантика языка APL сложилась как композиция из основных семантических систем вычислений, структуризации данных и управления парадигмы стандартного программирования, распространенных с обработки скаляров на обработку однородных векторов произвольной размерности. Основное отличие – организация памяти. Память функционирует в стиле без побочных эффектов (value-oriented), характерном для парадигмы функционального программирования. Спецификой реализации памяти для языка APL являет-

ся использование паспортов данных, хранимых независимо от собственно блоков данных, и допускающих программную обработку.

СП = (Текст → {Код | Адрес}): Пам [Переменная] → Пам

Таблица 14

Конкретизация понятий в языке APL

Понятие	ФП: Lisp	СП: Си	APL
Атом (Элементарное)	Атом, Числа и строки без ограничений на длину	Имя, |Скаляр|	Имя, |скаляр|
Структура	Список	Вектор, Строка, Запись, Объединение	Вектор
Переменная	Атом → ассоциативный список	Ид → Адр (Знач)	Ид → Адр (Знач), []
Значение	Данное	Адрес, |Скаляр|	Адрес <Паспорт, последовательность>
Выражение	Переменная, Префиксное в виде списка, начинающегося с функции	Произвольные, со скобками и приоритетами	Унарные, Бинарные, C := A + B
Действие/ Операция	Обработка списков, Арифметика	Арифметика , Доступ к элементам структур, Присваивания, Работа со строками. Формирование ТД	Арифметика, Выборки, = обработка паспортов
Условие/истина	Не NIL	0	0
Функция/ процедура	SUBR − подпрограмма, EXPR − выражение	Организация схем управления Поствычисления в операторах	

Аргумент	Локальные переменные в ассоциативном списке	Локальные переменные на стеке	Локальные переменные на стеке
Вызов Фн/подпр	Пополнение ассоциативного списка	Организация новой области имен и исполнение определения подпрограммы – блок	
Определение Фн/подпр	Пополнение ассоциативного списка	Пополнение таблицы функций	
Идентификатор	Атом	Адрес метки, константы, переменной, функции	Адрес

```
▼ SUM v; sp
[1]  vsp ← (0 <, v)/, v
[2]   sp ← +/ vsp
[3]   r ← sp ÷ +/ 0<, v
[4] ▼
```

Рис. 13. Пример: Определение операции

5.2. Теоретико-множественное программирование. Setl

Декларативное программирование на теоретико-множественном языке Setl сводит информационную обработку к автоматизации создания (развертывания) структур данных, над которыми все представляющие программу выражения становятся истинными высказываниями [83, 145]. При создании языка Setl в качестве базового ЯВУ был применен язык Fortran, конструкции которого определили вид семантических систем вычислений и управления процессами, с тем изменением, что основная структура данных − не векторы, а множества, причем предельно приближенные к математической традиции классического понимания множеств. Семантика эффективной работы с памятью на уровне указателей (pointer-oriented) сопряжена

с весьма абстрактной логической схемой выбора структур данных при размещении значений в памяти в зависимости от результатов анализа программы и контекста ее исполнения с использованием перебора вариантов, что сближает реализационную семантику работы с памятью с парадигмой логического программирования. Параллелизм программируется как обработка элементов множеств.

Реализационные варианты СД в языке Setl для множеств предусматривают различия в методах хранения и обработки кортежей, отображений, формирователей множеств над базовыми множествами, хэш-таблиц с разными расстановочными функциями, неполностью вычисленных множеств. При этом обеспечивается профилактика потерь памяти. Программы обработки множеств не зависят от разных методов их представления в памяти. Выразительная сила языка достаточна для его применения в качестве теоретико-множественной спецификации программ.

Таблица 15

Конкретизация понятий в языке Setl

Понятие	СП (Fortran)	ЛП (Prolog)	Setl
Атом (Элементарное)	Имя, |Скаляр|	Имя, Число	Имя, Число
Структура	Вектор, Строка, Запись, Объединение	{} [] <>	{;} {|,,,} [,,] <,,,>
Переменная	Ид -> Адр (Знач)	Имя	Имя
Значение	Адрес, |Скаляр|	Скаляр, {}	{} скаляр
Выражение	Произвольные, со скобками и приоритетами	Формирователь Кванторы Операторы над формулами	Формирователь, Кванторы, Операторы
Действие/ Операция	Арифметика , Доступ к элементам структур, Присваивания, Работа со строками. Формирование ТД	Выбор произвольного	= + элемент и теор-множ
Условие/истина	0		

Функция/ Подпрограмма	Организация схем управления Поствычиления в операторах		Подпрограмма, Таблица
Аргумент	Локальные переменные на стеке		На стеке
Вызов Фн/подпр	Организация новой области имен и исполнение определения подпрограммы – блок		
Определение Фн/подпр	Пополнение таблицы функций	Как Fortran, Таблица	Перечисление, Как Fortran, Перегрузка (приоритеты)
Идентификатор	Адрес метки, константы, переменной, функции		

```
IF      C1,   'T T'
        C2,   'T F'
THEN (  S1; 'X  '
        S2; '   X' )
```

Рис. 14. Пример «if c1 then if c2 then s1 else s2» с помощью таблицы условий на языке SEL (версия Setl)

5.3. Высокопроизводительное программирование. БАРС

Программирование на языке БАРС нацелено на обеспечение высокопроизводительных вычислений и организацию асинхронных параллельных процессов [74]б При создании языка БАРС в качестве базового ЯВУ был применен популярный язык Pascal, в те годы перераставший из учебного в производственный язык системного программирования. При сохранении основных принципов семантики вычислений были существенно обобщены средства структуризации данных на основе понятия «комплекс», приспособленного к именованию элементов структур данных и учета кратности их использования. Работа с именованной памятью (Name-oriented) дополнена

возможностью задавать дисциплину доступа к элементам памяти в терминах их создания-уничтожения и обработки.

Радикальное продвижение в повышении уровня программирования, предложенное в языке БАРС, заключается в переносе идеи типизации данных на проблему типизации схем управления. Процесс обработки данных рассматривается как распределенная система, находящаяся под сетевым управлением. Узлы такой системы могут сработать в зависимости от условий готовности разной природы: доступность ресурсов, сигналы монитора, внутри сетевые отношения, иерархия сетей, правила функционирования разносортных подсетей. Вычисления как и в языке APL распространяются со скаляров на сложные структуры.

В целом определение языка БАРС представлено как семейство основных подъязыков, соответствующих основным семантическим системам[11], обеспечивающим программирование схем управления в виде сетей, объявление дисциплины доступа к памяти, наложение на процессы вычисления произвольных схем управления и дисциплины доступа к памяти, что позволяет рассматривать БАРС как язык представления распределенных информационных систем [119].

Таблица 16

Конкретизация понятий в языке БАРС

Понятие	СП (Pascal)	ООП (C++)	БАРС
Атом (Элементарное)	Имя, |Скаляр|	Имя объекта, поля, метода, Скаляр	Имя, скаляр
Структура	Вектор, Строка, Запись, Объединение	Класс с разметкой доступа	комплекс
Переменная	Ид → Адр (Знач)	Имя поля, Имя → Ид (Знач)	
Значение	Адрес, |Скаляр|	|скаляр|, Объект	
Выражение	Произвольные, со скобками и приоритетами	Объект.Метод	алгебра

[11] Понятие «основные семантические системы» и их состав были предложены В.Е. Котовым при разработке языка БАРС [74].

Действие/ Операция	Арифметика , Доступ к элемен-там структур, Присваивания, Работа со строками. Формирование ТД	Над скалярами, Доступ к полю, Применение метода	Над скалярами и комплексами, Отношения над сетями
Условие/истина	0		
Функция/ подпрограмма / модуль	Организация схем управления Поствычисления в операторах	Имя → Ид (подпр), Локализованы по иерархи классов	Схема управления, Дисциплина доступа
Аргумент	Локальные переменные на стеке	Локальные в стеке, Поле объекта	
Вызов Фн/подпр	Организация новой области имен и исполнение определения подпрограммы – блок	Выбор метода по ТД, принадлежности к классу	синхронизация
Определение Фн/подпр	Пополнение таблицы функций	Пополнение списка методов для класса	Именованы переход, наполненный определением
Идентификатор	Адрес метки, константы, переменной, функции	Адрес объекта, поля, переменной	Имя перехода

```
Модуль УРАВНЯТЬ
Если ≠ (↑АО, ↑ВО, ↑СО)
 Вход А = АО, В =  ВО, С = СО
 Выход АО = А, ВО = В, СО = С
   … определение ..
   … строение …
Конец
```

Рис. 15. Пример Обмен данными

5.4. Параллельное функциональное программирование. Sisal

Параллельное программирование на языке Sisal опирается на парадигму функционального программирования [57, 137]. Но замысел языка нацелен на создание конкуренции вечно живому языку Fortran и, кроме того, в качестве базового языка был использован язык VAL, в свою очередь многое унаследовавший от языка Pascal. От языка Fortran унаследован ряд идей по обработке и представлению векторов.

Система вычислений в языке Sisal использует понятие «мультизначение», позволяющее подобно языку APL распространять скалярные действия на данные любой структуры, а их обработку осуществлять на многопроцессорных конфигурациях. Работа с именованной памятью (Name-oriented) освобождена от проблем с побочными эффектами с помощью локализации участков с однократными присваиваниями – SSA-форм, что делает программы удобными для преобразований, оптимизации и компиляции, включая распараллеливание и масштабирование. Отображение значений при информационной обработке рассматривается как исполняемое на многопроцессорных конфигурациях. Результаты отображения могут повергнуться свертке или фильтрации. Формирование конфигурации управляется представлением пространства итераций и учетом зависимостей между одноуровневыми итерациями. Программа строится из участков с однократными присваиваниями, что упрощает технику оптимизационных и распараллеливающих оптимизаций.

Основное продвижение по технике программирования – развитие структуры циклов для их реализации на параллельных архитектурах. Введено понятие «пространство итераций» и предложена специальная конструкция для фильтрации мультизначений, получаемых при совмещенном исполнении итераций и участков повторяемости.

Таблица 17

Конкретизация понятий в универсальном языке программирования Sisal

Понятие	СП (Pascal, Fortran)	ФП (Lisp)	Sisal
Атом (Элементарное)	Имя, |Скаляр|	Атом, Числа и строки без ограничений на длину	Имя, скаляр
Структура	Вектор, Строка, Запись, Объединение	Список	Потоки множества
Переменная	Ид → Адр (Знач)	Атом → ассоциативный список	
Значение	Адрес, |Скаляр|	Данное	Мультизначения – арности
Выражение	Произволь-ные, со скобками и приоритетами	Переменная, Префиксное в виде списка, начинающегося с функции	SSA
Дейст-вие/Операция	Арифметика , Доступ к элементам структур, Присваивания, Работа со строками. Формирование ТД	Обработка списков, Арифметика	Просачивания Потоки фильтры
Условие/истина	0	Не NIL	
Функция/ Подпрограмма/ фильтр	Организация схем управления Поствычисле-ния в операторах	SUBR – подпрограмма, EXPR – выражение	Пространство итера-ций
Аргумент	Локальные переменные на стеке	Локальные переменные в ассоциативном списке	Локалы Мультизначение после цикла для фильтра

Вызов Фн/подпр	Организация новой области имен и исполнение определения подпрограммы – блок	Пополнение ассоциативного списка	
Определение Фн/подпр	Пополнение таблицы функций	Пополнение ассоциативного списка	
Идентификатор	Адрес метки, константы, переменной, функции	Атом	Адрес значения, Вектор, Имя тега, Тип и др.

```
for i in [1..2] dot j in [3..4] do
% для пар индексов [1,3] и [2,4]
    returns  product (i+j)
% произведение сумм
    end for                              %  = 24
```

Рис. 16. Пример: В for-выражениях операция «dot» может порождать пары индексов при формировании пространства итерирования [57]

Выводы:

Для ЯСВУ характерна яркая специфика, связанная с поиском новых средств и методов программирования. Такая специфика может стать основой новой парадигмы.

Ряд ЯВУ, такие как Fortran, Lisp, Algol, Apl, Pascal, используются как базовые при создании новых ЯВУ и ЯСВУ, что позволяет формулировать относительные парадигматические характеристики в лаконичной и легко воспринимаемой форме.

При переходе к анализу парадигм компьютерных языков (КЯ) следует рассмотреть БНФ, SQL, UML, XML, YACC [143], Corba [154], FDD [117], XP [65], .Net [120], грамотное [153], экстремальное и компонентное [144] программирование, языки отладки и тестирования [132] программ, а также языки подготовки презентаций и другие КЯ, дополняющие языковое покрытие жизненного цикла программ.

Таблица 18

Схема языкового покрытия фаз ЖЦП для задач разного уровня изученности

Изученность и сложность задач	Фаза ЖЦП	Поддержка ЖЦП языками программирования						Дополнение поддержки ЖЦП другими компьютерными языками
			Парадигмы языков высокого уровня					
Многоуровневое абстрагирование для особо важных и сложных задач со специальными методами решения	Обоснование целесообразности							Языки подготовки презентаций Языки дистанционного общения
	Постановка задачи						ЯСВУ	
	Спецификация требований							Алгебраические языки спецификаций систем реального времени
	Проектирование			ФП			ЯСВУ	
	Реализация	ЯНУ		ФП				
	Тестирование	ЯНУ		ФП				
	Сопровождение							Языки онтологических систем
Бизнес-приложения, зависящие от динамики производственной деятельности человека	Обоснование затрат							графические языки моделирования бизнес процессов, Языки управления проектами
	Постановка задачи					ООП		
	Спецификация требований			ФП				
	Проектирование процессов		СП	ФП		ООП		Языки управления проектами
	Программирование и отладка		СП	ФП		ООП		
	Тестирование и отладка		СП			ООП		Языки тестирования и отладки

	Сопровождение							Языки онтологических систем, языки для распределенных информационных систем
	Развитие					ООП		
Сбор фактов и накопление знаний для не вполне определенных задач	Постановка задачи							Языки подготовки презентаций, языки запросов к базам данных
	Представление СД			ФП	ЛП			
	Тестирование и отладка				ЛП			
	Сопровождение				ЛП	ООП		языки онтологических систем
Исследование и разработка новых алгоритмов и структур данных, включая создание новых КЯ	Постановка исследования						ЯСВУ	
	Спецификация критериев						ЯСВУ	
	Проектирование алгоритмов			ФП				
	Программирование		СП	ФП				
	Тестирование и отладка			ФП				
	Сопровождение							Языки подготовки публикаций
	Развитие			ФП		ООП		
Реализация хорошо изученных алгоритмов для корректных задач	Постановка задачи						ЯСВУ	
	Проектирование СД		СП	ФП				Языки проектирования
	Кодирование	ЯНУ	СП					
	Отладка			ФП				Языки тестирования и отладки
	Сопровождение							Языки подготовки публикаций

ЗАКЛЮЧЕНИЕ

Функциональный подход к парадигматической характеристике языков и систем программирования сложился в 1970-е годы в сфере влияния академика А.П. Ершова, в рамках дискуссий на фоне конференций и семинаров при обсуждении разноплановых экспериментальных проектов, нацеленных на развитие средств, методов и технологий эффективной и надежной реализации программистского инструментария [11]. В те годы число рассматриваемых языков едва превышало пару–тройку сотен, что упрощало выделение ключевых идей в языкотворчестве. В настоящее время актуальность проблемы классификации языков программирования и вообще компьютерных языков и информационных систем существенно возросла.

Определитель парадигмы языка программирования содержит следующие процедуры:

1) Определение уровня языка и его ниши в жизненном цикле программ и деятельности программистов (цели и задачи), базовых языков, использованных при его создании и реализации.
2) Разложение языка на фрагменты по уровням/концентрам и слоям с целью выделения основных семантических систем языка и его реализационного ядра – семантический базис.
3) Декомпозиция семантического базиса языка на основные семантические системы с минимизацией их сложности и, возможно, их описание относительно базовых языков.
4) Определение АМ языка и интерпретатора, формально достаточного для построения расширений, эквивалентных исходному языку – нормализованное определение.
5) Сравнение полученного определения с описаниями известных парадигм.
6) Обоснование выводов относительно парадигмы исследуемого языка.

Предварительный эксперимент по спецификации опорных языков программирования ведется в форме этюдов по курсу «Парадигмы программирования».

СПИСОК ЛИТЕРАТУРЫ

1. Абрамов В.Г., Трифонов Н.П., Трифонова Г.Н. Введение в язык Паскаль. − М.: Наука, 1988.
2. Агафонов В. Н. Синтаксический анализ языков программирования. –Новосибирск, 1981. − 91 с.
3. Айлиф Дж. Принципы построения базовой машины. − М.: Мир, 1973. − 119 с.
4. Андреева Т.А., Ануреев И.С., Бодин Е.В.и др. Компьютерные языки как форма и средство представления, порождения и анализа научных и профессиональных знаний // Тр. XV Всерос. научно-методической конф. «Телематика-2008» − Санкт-Петербург, 2008. − С. 77–78.
5. Андре Ж., Беживин Ж., Небю ж.-Л., Рану Р. Некоторые замечания по поводу морфологии программирования // Создание качественного программного обеспечения: Тр. раб. конф. междунар. федерации по обработке информации. Т. 2. − Новосибирск, 1975. − С. 21–51.
6. Андрей Петрович Ершов – ученый и человек / Отв. ред. А.Г.Марчук. − Новосибирск: Изд-во СО РАН, 2006. − 504 с. (Наука Сибири в лицах)
7. Андрианов А.Н., Задыхайло И.Б., Мямлин А.Н.и др. тенденции в развитии супер-ЭВМ. Внешнее окружение и языковые проблемы // Прикладная информатика. Вып 2 (7) / Сб.статей под ред. В.М. Савинкова. − М.: Финансы и статистика, 1984. − С. 187–199.
8. Ануреев И.С., Бодин Е.В., Городняя Л.В.и др. Проблема классификации компьютерных языков // Материалы 11 национальной конф. по искусственному интеллекту с международным участием (КИИ-08). − Дубна, 2008. – Т. 3. − С. 199–207.
9. Аугустон М.И. Язык программирования Rigal // Системная информатика. Вып. 1. Проблемы современного программирования. − Новосибирск: Наука. Сиб. отд-ние, 1991. − С.76–120.
10. Ахо А.В., Хопкрофт Дж.Э., Ульман Дж.Д. Структуры данных и алгоритмы. − М.: Вильямс, 2000. − 384 с.
11. Ершов А.П., Кожухин Г.И., Поттосин И.В. Руководство к пользованию системой АЛЬФА. − Новосибирск: Наука, 1968. − 179 с.
12. Баранов С.Н., Колодин М.Ю. Феномен Форта. // Системная информатика. Вып. 4. Методы теоретического и системного программирования. − Новосибирск: Наука. Сиб. изд. фирма, 1995. − С. 193–271.
13. Бауэр Ф.Л., Гооз Г. Информатика. Вводный курс. Ч. 1, 2. − М.: Мир, 1990.
14. Бежанова М.М., Поттосин И.В. Современные понятия и методы программирования. − М.: Научный мир, 2000. − 191 с.
15. Безбородов Ю.М. Сравнительный курс языка PL/1 − М.: Наука, 1980. − 191 с.
16. Болски М.И. Язык программирования Си. − М.: Радио и связь, 1988. − 96 с.

17. Браун П. Макропроцессоры и мобильность программного обеспечения. − М.: Мир, 1977. − 253 с.
18. Брой М. Информатика. Основополагающее введение. − М.: Диалог-МИФИ, 1996. − 299 с.
19. Вайнгартен Ф. Трансляция языков программирования. − М.: Мир, 1977. − 190 с.
20. Вегнер П. Программирование на языке Ада. − М.: Мир. 1983. − 239 с.
21. Вдовкин С.В., Кубенский А.А., Сафонов В.О. Реализация языка CLU // Системная информатика. Вып 1. Проблемы современного программирования. − Новосибирск: Наука. Сиб. отд-ние, 1991. − С. 127–130.
22. Вирт Н. От Модулы к Оберону. − // Системная информатика. Вып 1. Проблемы современного программирования. – Новосибирск: Наука. Сиб. отд-ние, 1991. − С. 63–75.
23. Гололобов В. И., Чеблаков Б. Г., Ч и н и н Г. Д. Описание языка ЯРМО. − Новосибирск. – (Препр. / ВЦ АН СССР Сибирское отд-ние; № 247, 248).
24. Голуб А.И. С и С++. Правила программирования. − М.: БИНОМ, 1996. − 271 с.
25. Городняя Л.В. Вычислительные сети для описания базового языка // Параллельные вычислительные и программные системы. − Новосибирск, 1981. − С. 135–151.
26. Городняя Л.В. Интервальная модель процесса программирования // Разработка сложных программных систем. − Ростов-н/Д, 1987.
27. Городняя Л.В. Информационно-инструментальная система анализа и преобразования программ // Оптимизирующая трансляция и конструирование программ. − Новосибирск, 1996. − С. 33–40.
28. Городняя Л.В., Конструирование распределенной измерительной среды для экспериментальных исследований производительности информационных систем // Сб. материалов междунар. конф. “Перспективы систем информатики”. – Новосибирск, 2003. – С. 9–11.
29. Городняя Л.В. Конструирование уточняемых функций при разработке программ // Интеллектуализация и качество программного обеспечения. − Новосибирск, 1994. − С. 13–20.
30. Городняя Л.В. Макетирование программ с помощью тестов и описаний // Языки и системы программирования. − Новосибирск, 1981 − С. 115–123.
31. Городняя Л.В. Некоторые диалекты Лиспа и сферы их приложения // Трансляция и преобразования программ. − Новосибирск, 1984. − С. 60–71.
32. Городняя Л.В. Об одном подходе к синтезу транслятора на примере языка Литтл // Теория и практика системного программирования. − Новосибирск, 1977. − С. 60–71.
33. Городняя Л.В. О конструировании учебных систем программирования // Методы трансляции и конструирования программ. − Новосибирск, 1987.
34. Городняя Л.В. Основы функционального программирования. – М.: Интернет-Университет Информационных технологий. − http://www.intuit.ru. − 272 с.

35. Городняя Л.В. Парадигмы параллельного программирования в университетских образовательных программах и специализации // Тр. Всерос. научная конф. «Научный сервис в сети Интернет: решение больших задач». − Новороссийск–Москва, 2008. − С. 180–184.
36. Городняя Л.В. Парадигмы программирования в профессиональной подготовке информатиков // Проблемы специализированного образования. − Новосибирск, 1998. − С. 115–124.
37. Городняя Л.В. Подход к декомпозиции систем программирования для пошагового их определения // Трансляция и оптимизация программ. − Новосибирск, 1984.
38. Городняя Л.В. Подход к ограничению новизны в программистских экспериментах. – Новосибирск, 1982. − 29 с. – (Препр. / Надзаг ВЦ СО АН СССР; № 357).
39. Городняя Л.В. Программные инструменты для ПЭВМ. – Новосибирск, 1991. − 44 с. – (Препр. / ИСИ СОАН СССР; № 2).
40. Городняя Л.В. Реализация Лисп-интерпретатора. – Новосибирск: ВЦ СО РАН СССР, 1974. − С. 24–35.
41. Городняя Л.В. Схема изучения языка программирования с практикой на ЭВМ // Теория и практика системной информатики и программирования. − Новосибирск, 1988.
42. Городняя Л.В. Функциональное программирование: стиль, метод и потенциал // Тр. междунар. конф. «Космос, астрономия и программирование» (Лавровские чтения). Санкт-Петербургский государственный университет. − Санкт-Петербург, 2008. − С. 46–53.
43. Городняя Л.В. Функциональный подход к системному представлению прикладных программ учебного назначения. − Новосибирск, 1993. − 25 с. – (Препр. / РАН, Сиб. отд-ние. ИСИ; № 16).
44. Городняя Л.В., Евстигнеев В.А., Касьянов В.Н. Вопросы эффективного использования параллельных ЭВМ // Математические модели и численные методы механики сплошных сред. − Новосибирск, 1996. − С. 221–222.
45. Городняя Л.В., Касьянов В.Н. Проблемы и перспективы исследования преобразований программ на базе современных технических средств // Проблемы конструирования эффективных и надежных программ. − Новосибирск, 1995. − С. 7–18.
46. Грабер М. Введение в SQL. − М.: Лори, 1996. − 377 с.
47. Грис Д. Наука программирования. − М.: Мир, 1984.
48. Грисуолд, Р., Поудж Дж., Полонски И. Язык программирования Снобол-4. − М.: Мир, 1980. − 268 с.
49. Грогоно П. Программирование на языке Паскаль. − М.: Мир, 1982. − 382 с.
50. Гуревич Ю. Последовательные машины абстрактных состояний охватывают последовательные алгоритмы // Системная информатика. Вып 9. Формальные модели и модели информатики. − Новосибирск: Изд-во СО РАН, 2004. − С. 7–50.

51. Дал У., Мюрхауг Б., Нюгорд К. Симула -67 универсальный язык программирования. − М.: Мир, 1969. − 99 с.
52. Дао Л. Программирование микропроцессора 8088. − М.: Мир, 1988. − 357 с.
53. Дейкстра Э. Дисциплина программирования. − М.: Мир, 1978. − 275 с.
54. Дехтяренко И.А. Декларативное программирование. – http://www.softcraft.ru/paradigm/dp/dp02.shtml
55. Дьяконов В.П. Язык программирования Лого. − М.: Радио и связь, 1991. − 145 с.
56. Евстигнеев В.А. VLIW-машины: развитие архитектуры и принципов построения программного обеспечения // Системная информатика. Вып 4. Методы теоретического и системного программирования. − Новосибирск: Наука. Сиб. изд. фирма, 1995. − С. 304–333.
57. Евстигнеев В.А., Городняя Л.В., Густокашина Ю.В. Язык функционального программирования SISAL // Интеллектуализация и качество программного обеспечения. − Новосибирск, 1994. − С. 21–42.
58. Евстигнеев В.А., Касьянов В.Н. Графы в программировании: обработка, визуализация и применение. − С-Пб.: ЕХП-Петербург, 2003. − 1104 с.
59. Ершов А.П. Смешанные вычисления: потенциальные приложения и проблемы исследования // Тез. докладов и сообщений. Всесоюз. конф. «Методы математической логики в проблемах искусственного интеллекта и систематическое программирование». Ч. 2. − Вильнюс, 1980. − С. 26–55.
60. Захаров Л.А., Покровский С.Б., Степанов Г.Г., Тен С.В. Многоязыковая транслирующая система. − Новосибирск, 1987. − 151 с.
61. Землянский А.А. Метод композиции программ иерархической структуры // Системная информатика. Вып 4. Методы теоретического и системного программирования. − Новосибирск: Наука. Сиб. изд. фирма, 1995 − С. 135–142.
62. Зуев Е.А. Программирование на языке Turbo-Pascal 6.0 -7.0. − М.: Веста, Радио и связь, 1993.
63. Зубенко В.В., Протасов А.В. Язык программирования Фоли (Форт + Лисп) // Системная информатика. Вып 2. Системы программирования. Теория и приложения. – Новосибирск: Наука. Сиб. изд. фирма, 1993. − С. 183–215.
64. Ингерман П. Синтаксически ориентированный транслятор. − М.: Мир, 1969. − 174 с.
65. Йодан Э. Смертельный марш или трудно выполнимая миссия.
66. Йодан Э. Структурное проектирование и конструирование программ. − М. «Мир» , 1979. − 409 с.
67. Камынин С. С., Любимский Э. З. Алгоритмический машинно-ориентированный язык АЛМО // Алгоритмы и алгоритмические языки. – Новосибирск, ВЦ АН СССР, 1967. – Вып. 1.
68. Капитонова Ю.В., Летичевский А.А., Мищенко Н.М. расширяющиеся языки и системы программирования системы «Проект» // Прикладная информатика. Вып 2 (7) / Сб.статей под ред. В.М. Савинкова. – М.: Финансы и статистика, 1984. − С. 163–171.

69. Катцан Г. Язык Фортран 77. − М.: Мир. 1982. − 208 с.
70. Кнут Д. Искусство программирования. Основные алгоритмы. − М.: Мир, 1976. − 735 с.
71. Коддингтон Л. Ускоренный курс Кобола. − М.: Мир. 1974. − 270 с.
72. Колин А. Введение в операционные системы. − М.: Мир, 1975. − 116 с.
73. Котляров В.П. CASE-технология и возможности современных CASE-средств в поддержке этапов проектирования программного продукта // Системная информатика. Вып 4. Методы теоретического и системного программирования. − Новосибирск: Наука. Сиб. изд. фирма, 1995. − С. 272–303.
74. Котов В.Е. МАРС: архитектуры и языки для реализации параллелизма // Системная информатика. Вып 1. Проблемы современного программирования. − Новосибирск: Наука. Сиб. отд-ние, 1991. − С.174–194.
75. Лавров С.С. Использование вычислительной техники, программирование и искусственный интеллект (перспективы развития) // М.: Микропроцессорные средства и системы. – 1984. – № 3. − С. 3–9.
76. Лавров С.С. Лекции по теории программирования. Учебное пособие. − С-Пб.: Нестор. − 107 с.
77. Лавров С.С. Методы задания семантики языков программирования // Программирование. – 1978. – № 6. − С. 3–10.
78. Лавров С. С. Основные понятия и конструкции языков программирования. − М. Финансы и статистика, 1982. − 80 с.
79. Лавров С.С. Расширяемость языков. Подходы и практика // Прикладная информатика. – 1984. – Вып. 2. − С. 17–23.
80. Лавров С.С. Универсальный язык программирования. − М.: Наука. 1972. − 183 с.
81. Лавров С.С., Городняя Л.В. Функциональное программирование. Интерпретатор языка Лисп // Компьютерные инструменты в образовании. – С-Пб., 2002. – № 5.
82. Лавров С.С., Силагадзе Г.С. Автоматическая обработка данных. Язык Лисп и его реализация − М.: Наука. 1978. − 175 с.
83. Левин Д.Я. Язык сверхвысокого уровня СЕТЛ − Новосибирск: Наука, 1983. − 160 с.
84. Лейнингем И. Освой самостоятельно Python. – М.: Вильямс. − 444 с.
85. Лельчук Т.И., Марчук А.Г. Язык программирования Поляр: описание, использование, реализация. − Новосибирск, 1986. − 94 с.
86. Леман М.М. Программы, жизненные циклы и законы эволюции программного обеспечения. − М.: Мир, ТИИЭР, 1980. – Т. 68-9. − С. 26–45.
87. Ломазова И.А. Вложенные сети Петри. – М.: Научный мир. − 207 с.
88. Лорин Г. Распределенные вычислительные системы. − М. Радио и связь, 1984. − 293 с.
89. Магариу Н.А. Язык программирования АПЛ. − М.: Радио и связь. − 96 с.
90. Малиновский Б.Н. История вычислительной техники в лицах. − Киев: Фирма «КИТ, ПТОО А.С.К.»,1995. − 383 с.

91. Манцивода А.В. Программирование в ограничениях на Фленге // Системная информатика. Вып 4. Методы теоретического и системного программирования. − Новосибирск: Наука. Сиб. изд. фирма, 1995. − С. 118–159.
92. Малпас Дж. Реляционный язык Пролог и его применение. − М.: Наука, 1990. − 463 с.
93. Марчук А.Г., Городняя Л.В., Мурзин Ф.А., Шилов Н.В. Классификация компьютерных языков: состояние, проблемы, перспективы // Тр. междунар. конф. “Космос, астрономия и программирование" (Лавровские чтения). Санкт-Петербургский государственный университет. − Санкт-Петербург, 2008. − С. 15–22.
94. Мейер Б. Основы объектно-ориентированного проектирования. – М.: Интернет-Университет Информационных технологий. – http://www.intuit.ru/department/se/oopbases/, 2007
95. Михелев В. М., Вершубский В. Ю. АСТРА, новые возможности языка. − М., 1961. –. (Препр. / ИПМ АН СССР; № 61).
96. Непейвода Н.Н. Стили и методы программирования. – М.: Интернет-Университет Информационных технологий , 2004. – http://www.intuit.ru/department/se/progstyles/
97. Новосибирская школа программирования. (Перекличка времен) / Под ред. проф. И.В. Поттосина и к.ф.-м.н. Л.В. Городней. − Новосибирск, 2004. − 243 с.
98. Оллонгрен А. Определение языков программирования интерпретирующими автоматами. − М.: Мир, 1977. − 288 с.
99. Парамзин А.В. Введение в программирование на языке Ассемблера. − Новосибирск, 1993. − 180 с.
100. Пентковский В.М. Автокод Эльбрус. − М.: Наука, 1982. − 350 с.
101. Пентковский В.М., Синдеев Б.П. Использование расширяемого языка высокого уровня для определения специализированных языков управления системами // Системная информатика. Вып 4. Методы теоретического и системного программирования. − Новосибирск: Наука. Сиб. изд. фирма, 1995. − С. 93–101.
102. Пеппер П., Экснер Ю., Зюдхольд М. Функциональный поход к разработке программ с развитым параллелизмом // Системная информатика. Вып 4. Методы теоретического и системного программирования. − Новосибирск: Наука. Сиб. изд. фирма, 1995. − С. 334–360.
103. Пересмотренное сообщение об АЛГОЛЕ 68 / Под ред. А. П. Ершова. − М.: Мир, 1979.
104. Пильщиков В.Н. Язык Плэнер. − М.: Наука. 1983. − 207 с.
105. Поттосин И.В. Система СОКРАТ: Окружение программирования для встроенных систем. − Новосибирск, 1992. − 20с. (Препр. / РАН. Сиб. отд-ние. ИСИ; № 11).
106. Пратт Т. Языки программирования: разработка и реализация. − М.: Мир, 1979. − 20 с.
107. Пратт Т., Зелковиц М. Языки программирования. Разработка и реализация / Под общей редакцией А. Матросова. − СПб.: Питер, 2002. – 688 с.

108. Прехельт Л. Эмпирическое сравнение семи языков программирования // М.: Открытые системы. – 2000. – № 12(56). − С. 45–52.
109. Редько В. Н. Языковые процессоры с семантико-синтаксическим управлением // Всесоюз. конф. по методам трансляции. − Новосибирск: ВЦ СО АН СССР, 1981. − С. 21–22.
110. Рендел Б., Рассел Л. Реализация Алгола-60. − М.: Мир, 1967. − 475 с.
111. Романенко Г. Л. Рефал-4 – расширение Рефала-2, обеспечивающее выразительность результатов прогонки. − М., 1987. − 27 с. – (Препр. / ИПМ АН СССР; № 147).
112. Руководство по языку Оккам. − Новосибирск, 1987. − 75 с.
113. Серебрянников В.А. Лекции по конструированию компиляторов. − М.: ВЦ РАН, 1994. − 175 с.
114. Сингер М. Мини-ЭВМ PDP-11: программирование на языке ассемблера и организация машины. − М.: Мир, 1984. − 272 с.
115. Скопин И.Н. Модели жизненного цикла программного обеспечения // Новосибирская школа программирования. Перекличка времен. − Новосибирск, 2004. − С. 120–173
116. Становление Новосибирской школы программирования / Под ред. проф. И.В.Поттосина. − Новосибирск, 2001. − 195 с.
117. Стивен Р. Палмер, Джон М.Фелсинг. Практическое руководство по функционально-ориентированной разработке ПО. − М.: Вильямс, 2002. − 299 с.
118. Страуструп Б. Язык программирования Си++. − М.: Радио и связь, 1992. − 348 с.
119. Таненбаум Э., ван Стеен М. Распределенные системы. Принципы и парадигмы. − СПб.: Питер, 2003. – 877 с.
120. Уоткинс Д., Хаммонд М., Эйбрамз Б. – Программирование на платформе .Net. − М. Вильямс, 2003. − 367 с.
121. Фаронов В.В. Delphi 4. Учебный курс. – М.: Нолидж, 1999. – 464 с.
122. Фет Я.И. История информатики в России: ученые и их школы.
123. Филд А., Харрисон П. Функциональное программирование / Пер. под редакцией В.А.Горбатова. – М. Мир, 1993. – 638 с.
124. Фуксман А.П. Технические аспекты создания программных систем. – М.: Статистика, 1979. – 180 с.
125. Хантер Р. Проектирование и конструирование компиляторов. – М.: Финансы и статистика, 1984. – 232 с.
126. Хендерсон П. Функциональное программирование. – М.: Мир, 1983. – 349 с.
127. Хигман Б. Сравнительное изучение ЯП. – М.: Мир, 1974. – 204 с.
128. Хоар Ч. Взаимодействующие последовательные процессы. – М.: Мир, 1989 – 264 с.
129. Хопгуд Ф. Методы компиляции. – М.: Мир, 1982. – 160 с.
130. Хьювенен Э., Сеппанен Й. Мир Лиспа. Т. 1,2. – М.: Наука, 1994.
131. Хьюз Дж., Мичтом Дж. Структурный подход к программированию. – М.: Мир, 1985.

132. Черноножкин С.Л. Методы тестирования программ.
133. Шнейдерман Б. Психология программирования. Человеческие факторы в вычислительных и информационных системах. – М.: Радио и связь, 1984. – 304 с.
134. Экхауз Р., Морис Л. Мини-ЭВМ: Организация и программирование. – М.: Финансы и статистика, 1983. – 359 с.
135. Эфрос Л.Б. Введение в программирование для ЭВМ БЭСМ-6. – Новосибирск, 1971. – 66 с.
136. Backus J. Can programming be liberated from the von Neumann style? A functional stile and its algebra of programs // Communs. ACM. – 1978. – Vol. 21, N 8. – P. 613–641.
137. Cann D. C. SISAL 1.2: A Brief Introduction and tutorial. – California, May, 1992. – 128 p. – (Prepr. / Lawrence Livermore National Lab., Livermore; UCRL-MA-110620).
138. Henner C.R. A Simple Set Theory for Computer Science – Toronto. – TR # 102, May 1979. – 12 p.
139. Hudak P. Conseption, Evolution and Application of Functional Languages // ACM Computing Serveys. – 1989. – Vol. 21, N 3. – P. 359–411.
140. Graham P. ANSI Common Lisp. – Prentice Hall, 1996. – 432 p.
141. Lucas P., Lauer P., Stigleitner H. Method and Notation for the Formal Definition of Programming Languges. – Venna, 1968. – (Prepr. / IBM Laboratory; TR 25.087).
142. McCarthy J. LISP 1.5 Programming Manual. – The MIT Press., Cambridge, 1963. – 106 p.
143. Ritchie D.M., Tompson K. The UNIX Time-Sharing System // Bell System Technical J. – 1978. – Vol. 57, N 6. – P. 1905–1929.
144. Rogoson D. Inside COM/DCOM. – Microsoft Press. 1997. – 376 p.
145. Schwartz, Jacob T., Set Theory as a Language for Program Specification and Programming. – Courant Institute of Mathematical Sciences, New York University, 1970.
146. Weinberg G.M. The Psychology of Computer Programming. – New York: Van Norstand Reinhold Comp., 1971.
147. http://en.wikipedia.org/wiki/Programming_paradigms
148. http://haskell.org/aboutHaskell.html – Материалы по языку Haskell.
149. http://refal.org/rf5_frm.htm – Материалы по языку Рефал.
150. http://www.cons.org/cmucl/ – Сайт с материалами по особо эффективной реализации Lisp-а – CMUCL.
151. http://www.erlang.org – Официальный сайт языка Erlang.
152. http://www.gwydiondylan.org/drm/drm_7.htm – Официальный сайт языка Dylan .
153. http://www.literateprogramming.com Сайт с материалами по грамотному программированию. Кнут Д. Literate programming.
154. http://www.optim.ru/cs/1998/4/corba/corba.asp – Сайт с материалами по основам CORBA.
155. http://www.smalltalk.ru – сайт с материалами по языку SmallTalk.

156. Черняк Л. IBM и семь гномов – Computerworld. – 16/05/2006. –№18. – http://www.osp.ru/cw/2006/18/1323638/
157. http://www.cynami.com
158. http://www-personal.umich.edu/~jkglenn/beautiful.pdf
159. http://www.cs.wisc.edu/trans-memory/biblio/

Л.В. Городняя

ФУНКЦИОНАЛЬНЫЙ ПОДХОД К ОПИСАНИЮ ПАРАДИГМ ПРОГРАММИРОВАНИЯ

Препринт
152

Рукопись поступила в редакцию 29.10.09
Редактор Т. М. Бульонкова
Рецензент Н. В. Шилов

Подписано в печать 28.12.09
Формат бумаги 60 × 84 1/16 Объем 3.8 уч.-изд.л., 4.2 п.л.
Тираж 60 экз.

Центр оперативной печати «Оригинал 2»
г.Бердск, ул. О. Кошевого, 6, оф. 2, тел. (383-41) 2-12-42