Компьютерные инструменты в образовании, 2021
№ 3: 57–75
УДК: 004.432.42
http://cte.eltech.ru
doi:10.32603/2071-2340-2021-3-57-75

информатика

О ФУНКЦИОНАЛЬНОМ ПРОГРАММИРОВАНИИ

Городняя Л. В.[1,2], кандидат физико-математических наук, ✉ lidvas@gmail.com,
orcid.org/0000-0002-4639-9032

[1]Новосибирский государственный университет, ул. Пирогова, 1, 630090, Новосибирск, Россия
[2]Институт систем информатики им. А. П. Ершова СО РАН,
пр. Академика Лаврентьева, д. 6, 630090, Новосибирск, Россия

Аннотация

Статья посвящена описанию особенностей функционального программирования, рассматриваемого как методология решения новых и исследовательских задач прикладного и системного программирования. Привлечена методика анализа и сравнения парадигм программирования, учитывающая приоритеты принятия решений в процессе разработки программ. Методика сравнения языков и парадигм программирования основана на неформальном определении термина «парадигма программирования», согласно которому при сравнении парадигм следует выделять отличительные тестируемые особенности, допускающие проверку. В качестве таких признаков оказалось полезным использовать приоритеты при принятии решений на разных этапах изучения постановки задачи, а затем разработки и отладки программы её решения. Учёт приоритетов позволяет прогнозировать сложность процессов применения программируемых решений, начиная с планирования, изучения и организации разработки долгоживущих программ.

Материал статьи имеет несколько дискуссионный характер. В статье дана четкая формулировка принципов парадигмы функционального программирования, отличающих её от других парадигм. На этих принципах выполнен вывод следствий, позволяющих успешно применять функциональное программирование при решении сложных задач, они конкретизированы на задачи организации параллельных вычислений и повышения производительности программ, созданных в рамках парадигмы функционального программирования. Показана сложность создания программ для решения новых задач на примере параллельных вычислений и описаны требования к универсальному мультипарадигмальному языку параллельных вычислений. Для задач функционального программирования правильность и полнота решений важнее эффективности и производительности полученных программ. Именно этот выбор приоритетов позволяет функциональное программирование рассматривать как общую методику подготовки прототипов или функциональных моделей. Можно сказать, что функциональное программирование выполняет роль проектно-конструкторского бюро для производственного программирования.

Ключевые слова: *функциональное программирование, язык программирования, парадигма программирования, система программирования, прагматика, метапарадигма, новые задачи, параллельные вычисления.*

Цитирование: Городняя Л. В. О функциональном программировании // Компьютерные инструменты в образовании. 2021. № 3. С. 57–75. doi: 10.32603/2071-2340-2021-3-57-75

1. ВВЕДЕНИЕ

Функциональное программирование — одна из первых парадигм, направленных не столько на получение эффективной реализации заранее созданных и хорошо изученных алгоритмов, сколько на решение новых и исследовательских задач [1–7]. Для таких задач правильность и полнота решений важнее эффективности и производительности полученных программ. Именно этот выбор приоритетов позволяет функциональное программирование (ФП) рассматривать как общую методику подготовки прототипов или функциональных моделей, позволяющих в облегчённой форме отладить основные механизмы поведения и применения программы. Очередь эффективности и производительности приходит после того, как достигнута правильность и полнота решений задачи, получена оценка целесообразности многократного применения программируемых решений. В пионерскую эпоху применения компьютеров была характерна практика программирования вычислений по известным, ранее созданным, алгоритмам, правильность которых сомнений не вызывала или была доказана задолго до появления первых компьютеров. Было достаточным обеспечить эффективность программ.

Для описания особенностей ФП в данной статье используются результаты анализа парадигм программирования (ПП). Методика сравнения языков программирования (ЯП), представленная в [8–10], основана на неформальном определении термина «парадигма программирования», данном в [10], в котором говорится, что при сравнении парадигм следует выделять отличительные тестируемые особенности, допускающие проверку. В качестве таких признаков оказалось полезным использовать приоритеты при принятии решений на разных этапах изучения постановки задачи, а затем разработки и отладки программы её решения. Учёт приоритетов позволяет прогнозировать сложность процессов применения программируемых решений, начиная с планирования, изучения и организации разработки долгоживущих программ.

Рост популярности ФП обычно происходит при смене элементной базы, порождающей обширные классы новых задач. Новизна этих задач обусловлена не только неосвоенными возможностями технических новинок, но и сменой поколений специалистов, обладавших навыками исследования постановок задач и компьютерного эксперимента, а также расширением круга пользователей программного обеспечения [11]. ФП помогает повышать производительность программ благодаря удобству предварительной подготовки их прототипов или функциональных моделей, после отладки которых приходит время для более эффективной или производительной версии программы в рамках любой парадигмы.

Дискуссионными остаются два вопроса:

1. Какие особенности ФП способствуют его полезности при смене элементной базы?
2. Может ли ФП стать доминирующей парадигмой производственного программирования?

Изложение начинается с рассмотрения основных принципов ФП и их конкретизации, возникающей при переносе методов ФП на возможные приложения, например на организацию параллельных вычислений [12]. Дано описание семантических и прагматических принципов ФП и следствий этих принципов, дающее ответ на первый дискуссионный вопрос. Отмечена повышенная сложность программирования при решении новых задач. Особо подчеркивается перспектива ФП как универсального метода решения задач, отягощенных трудно верифицируемыми и плохо совместимыми требованиями. Отдельно рассматривается оценка сложности параллельного программирования с учетом сте-

пени изученности задач параллельных вычислений (ПВ) и уровня образования специалистов [13]. В заключении описаны требования к новому практичному учебнопроизводственному языку параллельного программирования и предварительный ответ на второй дискуссионный вопрос.

2. ПРИНЦИПЫ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ

2.1. Общее представление

Термин «функциональное программирование» в настоящее время допускает два толкования. Исторически в начале 1960-х годов Дж. Маккарти провозгласил, что все понятия программирования могут трактоваться как функции или результат применения функций [1]. В середине 1970-х годов Дж. Бекус привлек внимание к ФП, призывая преодолеть узость так называемого «бутылочного горлышка» оператора присваивания с помощью языков ФП [2]. Собственно определение термина не было дано, оно использовалось интуитивно без особых разночтений и постепенно стало восприниматься как отдельная парадигма программирования, в которой предполагается, что **правильность важнее эффективности**. ФП даёт приоритет организации вычислений и сложных структур данных, а вопросы распределения и обработки памяти, а также управления процессами вычисления уходят на второй план.

Примерно с середины 1990-х кристаллизовалась идея «чистого функционального программирования» как раздела дискретной математики, исследующего однозначные функции в дискретном пространстве над значениями типа целых чисел, символьных строк или графов. Чистое ФП можно рассматривать как математическую основу более общего производственного ФП. Постепенно это «чистое функциональное программирование» стали называть просто «функциональным программированием». Такое разделение смыслов примерно соответствует разнице в стандартах на академические и производственные языки программирования. Теперь общее определение ФП нередко начинают с утверждения, что характеристическая его особенность заключается в том, что одинаковые формулы в одинаковом контексте имеют одинаковое значение. Из этого выводят исключение присваиваний, глобальных переменных, передач управления, побочных эффектов, работы с внешними устройствами. Такая формула уязвима из-за отсутствия единого точного определения термина «контекст».

Понятие «контекст» в программистском лексиконе обладает некоторой двойственностью. Это одновременно и фрагмент программы, иногда называемый областью существования или видимости, и контекстная таблица соответствия используемых в этом фрагменте символов и их значений. Семантика ЯП может по-разному ограничивать правила воздействия на таблицу соответствия. В одном ЯП могут сосуществовать две и более контекстных таблицы для одного фрагмента или синтаксической позиции — статика и динамика, а кроме того, глобальный и локальный контекст. Системы программирования (СП) для одного ЯП могут по-разному решать вопрос о порядке перебора таблиц при определении значения формулы, более того, такой порядок может быть изменён по опциям из программы или задания на её компиляцию.

Можно обратить внимание, что при оптимизирующей компиляции нередко выделяют линейные участки между двумя соседними присваиваниями. В таком случае можно считать, что на внутреннем языке компилятора происходит приведение программы к структуре из участков однократного присваивания, удобной для обнаружения не вы-

числяемых, константных или дублирующихся формул. Компилятор заменяет константные формулы на вычисленную при компиляции константу, а дублирующиеся формулы — на переменную, значение которой будет вычислено при выполнении программы. Не исключено, что возможность при решении сложных задач явно и безопасно использовать при подготовке программ подобные оптимизации является сильным мотивом использования чистого ФП. Эта же причина позволяет ФП быть полезным дополнением для любой парадигмы программирования.

Если контекстом считать участок однократного присваивания, то приведённая выше характеристика не отличает парадигму ФП от других парадигм, допускающих запись программ в таком стиле. Кроме того, ещё в начале 1960-х Маккарти отмечал, что роль глобальных переменных могут выполнять самые внешние локальные переменные [1]. Современные чисто функциональные языки программирования, такие как Haskell, для работы с побочными эффектами и внешними устройствами расширяют язык специальными «монадами» [14], другие — просто библиотечными модулями или пакетами [15]. Отсутствие передач управления во многих функциональных ЯП компенсируется введением понятия «продолжение» (continuation), позволяющего через параметры передавать имя следующей функции [16]. Можно вспомнить, что ещё при создании языка Algol-68 была замечена равносильность понятий «вызов процедуры» и «передача управления», а также присваиваний и передачи параметров [17]. Таким образом, при переходе к чистому ФП происходит не «исключение» вышеперечисленных понятий программирования, а «изменение» их формы вхождения в программы[1].

Обычно ФП подразумевает поддержку ряда семантических и прагматических принципов, которые способствуют созданию функциональных моделей на этапе компьютерных экспериментов, полезных при решении новых и исследовательских задач. При подготовке программы программист может следовать семантическим принципам, отчасти выраженным или провозглашённым в определении семантики языка программирования. Прагматические принципы предоставляются системой программирования, освобождая программиста от решений, не связанных с характером постановки задачи. Описания таких принципов при определении языка обычно выглядят как примечания. Большинство семантических и прагматических принципов были заложены Дж. Маккарти в первых реализациях языка Lisp [1]. Некоторые сложились позднее в практике разработки новых систем ФП при поиске более эффективных и производительных системных решений [16].

2.2. Семантические принципы

Функциональное программирование поддерживает семантические принципы представления функций, такие как всеобщность (универсальность), самоприменение, равноправие параметров.

Всеобщность / Универсальность. Понятия «функция» и «значение» представлены теми же символами, что и любые данные для компьютерной обработки. Каждая функция, применяемая к любым данным, за конечное время выдаёт результат или диагностическое сообщение. Этот принцип позволяет строить представления функций из их частей — символов и вычислять части представления по мере поступления данных. Они могут формироваться даже в процессе вычислений и обработки информации о них. Нет никаких препятствий для обработки представлений функций так же, как и для обработ-

[1] «Гони природу в дверь, она влетит в окно» Н. М. Карамзин «Чувствительный и холодный. Два характера»

ки любых данных. В принципе, нет ограничений на манипулирование средствами языка, функциями из определения семантики языка, конструкциями для реализации языка в СП, а также выражениями в программе. Все, что было необходимо при реализации ЯП, может пригодиться при его использовании. Это определяет открытость систем ФП. Строго говоря, в отличие от математики, программирование вообще не имеет дело ни со значениями, ни с функциями. Программирование работает исключительно с данными, которые могут представлять значения и/или функции. Это давно отмечали С. С. Лавров и Г. С. Цейтин [18, 19].

Исторически близкое понятие — принцип хранимой программы. Идея хранимой программы впервые сформулирована в описании аналитической машины Чарльза Беббиджа, через сто лет реализована в компьютерах Конрада Цузе и в определении машины Алана Тьюринга, позднее провозглашена в архитектуре Джона фон Неймана. В результате принципиально подтверждена достаточность универсального представления информации, при котором нет особой разницы в природе данных для представления значений и функций. Следовательно, нет и препятствий для обработки данных, представляющих функции, теми же средствами, какими обрабатываются данные, представляющие значения.

Самоприменение. Представления функций могут использовать сами себя прямо или косвенно, что позволяет создавать четкие и краткие рекурсивные символьные формы. Представление как значений, так и функций может быть рекурсивным.

Примеры самоприменения дают многие математические функции [20], особенно рекурсивные, такие как факториал, числа Фибоначчи, суммирование рядов и многие другие, определение которых использует математическую индукцию. В технологии программирования методы пошаговой или непрерывной разработки программ, а также экстремальное программирование имеют некоторое сходство. Эти методы сводят организацию процесса программирования к серии шагов, каждый из которых обеспечивает либо работоспособную версию программы, либо инструмент для выполнения следующих шагов разработки.

Равноправие параметров. Порядок и метод вычисления параметров значения не имеют. Параметры функции не зависят друг от друга.

Можно заметить, что параметры при вызове функции вычисляются на одном уровне иерархии, в одном и том же контексте. Поэтому некоторые параметры могут вычисляться перед вызовом функции, а другие могут быть вычислены позже, но в том же контексте. Следовательно, представление любой выделенной формулы из определения функции можно превратить в параметр этой функции. Это и означает, что части представления функции могут быть вычислены в зависимости от промежуточных результатов, а функции могут быть построены с учетом условий их использования, в частности, расположения их определений и вызовов на разных уровнях иерархии представления программы. Любая символьная форма в определении функции может быть извлечена из неё, превращена в параметр и, наоборот, подставлена в неё.

Повторное использование данных обеспечивается именованием или идентификацией. Параметры имеют имена, часто называемые переменными, хотя в ФП они не меняют значения в одном и том же контексте. Чисто на уровне понятий переменная — это именованная часть памяти, предназначенная для многократного доступа к изменяющимся данным, а константа — к неизменяемым данным. Изменение отношения между именем и значением в ФП возможно только путем перехода в другой контекст, что эквивалентно

изменению имени. Функциональные переменные допустимы и равноправны обычным постоянным функциям и могут быть значениями аргументов или формироваться как результаты других функций. Редкость навыков работы с функциональными переменными означает лишь то, что пора освоить такую возможность, потенциал которой может превзойти ожидания сейчас, когда программирование становится более компонентно-ориентированным.

2.3. Прагматические принципы

Функциональное программирование поддерживает прагматические принципы организации вычислений, такие как гибкость ограничений, неизменяемость данных и строгость результата. Прагматические принципы поддерживаются системой программирования, точнее, разработчиками СП. Методы поддержки таких принципов обычно не определены в языке ФП, их выбирает разработчик системы ФП.

Гибкость ограничений. Для предотвращения необоснованных простоев памяти поддерживается оперативный анализ достижимости данных, освобождение и повторное использование памяти, хранящей недостижимые данные.

Бывает, что памяти не хватает не на всю задачу, а только на отдельные блоки данных, возможно, не так уж важные для ее решения. Такая проблема в системах ФП решается принципом гибкости ограничений на пространственные характеристики. Возникают ситуации, когда одни из этих частей исчерпаны, а другие не используют выделенное им пространство. В системах ФП эту проблему решает специальная функция — «сборщик мусора» (garbage collector), которая пытается автоматизировать перераспределение или освобождение памяти, когда какой-либо области памяти недостаточно [1]. Это означает, что данные могут быть любого размера. Новые реализации сборщика мусора эффективно учитывают преимущества восходящих процессов на больших объемах памяти [16]. Многие новые системы программирования теперь включают такие механизмы независимо от парадигмы.

Неизменяемость данных. Представление каждого результата функции помещается в новую часть свободной памяти без искажения аргументов этой функции, они могут быть полезны для других функций.

В любое время возможен доступ к данным, ранее полученным в результате вычислений. Таким образом, значительно упрощается отладка программ и обеспечивается обратимость любых действий. Всегда есть уверенность, что все промежуточные результаты сохранены, их можно проанализировать и повторно использовать в любое время. Если определение функции является статической конструкцией, то процесс её выполнения можно рассматривать как композицию входящих в её определение функций, развернутых в динамике в соответствии с этой конструкцией, что при анализе допускает символьные вычисления.

Отдельный спорный аспект связан с переходом от целых чисел к действительным числам, возможно, требующим изменения точности представления в процессе вычислений. Логически они остаются константами, но СП обрабатывает их как переменные.

Строгость результата. Любое количество результатов функции может быть представлено в виде единой символьной формы, из которой при необходимости может быть выбран желаемый результат. Часто этот принцип интерпретируется как требование единственного или даже исключительно скалярного результата функции, что приводит к со-

мнениям в законности использования функций целочисленного деления, извлечения корня, обратных тригонометрических функций и многих других категорий математических функций. В этом плане примечательны соображения Г. М. Фихтенгольца, приведённые в предисловии к учебнику по матанализу. Он отмечает, что если прямые функции часто бывают однозначными, то обратные, как правило, таким свойством не обладают [21].

2.4. Следствия

Представление алгоритмов в виде функциональных программ дает практически важные следствия. Мета-программирование, верификация и факторизация вытекают из семантических принципов. Прагматические принципы приводят к моделям непрерывности процессов, обратимости действий и унарных функций. Эти следствия служат основой для интуитивного оперирования функциональными моделями, которые позволяют ставить и понимать прямой компьютерный эксперимент.

Метапрограммирование является следствием принципа универсальности, который позволяет обрабатывать и создавать представления программ так же, как любые данные.

Данные, представляющие значение или функцию, могут быть составлены из частей вплоть до букв. Любые части данного могут быть вычисляемыми фрагментами. Любая функция может действовать как предикат для выбора фрагмента, который будет размещён как часть представления функции, а также использован как определение параметра или способа вычисления функции. Это обеспечивает поддержку мета-компиляции, включая синтаксически-ориентированные методы обработки, генерации и анализа программ. Также поддерживаются однородные представления программ, внешне сохраняющие аналогию или сходство с обрабатываемыми данными, например БНФ при конструировании компилятора или прототипами программ. Возможны смешанные и частичные вычисления, оптимизирующие преобразования, макрогенерация и многое другое, необходимое при создании операционных систем и СП. Всё это можно делать практически в любой СП, но обычно нет ни такой традиции, ни примеров.

Верификация основана на связи принципа самоприменения с методами рекурсии, математической индукции и логики.

Большинство систем верификации программного обеспечения создаются в рамках ФП. Становится возможным логически выводить отдельные свойства программ и благодаря этому обнаруживать некоторые тонкие ошибки. Если представление данного объекта имеет подобие некоторой логике вывода, то его свойства могут быть выведены с использованием этой логики. Это увеличивает надежность и безопасность программ, хотя и не позволяет решить проблему корректности в полной мере. Трудности связаны с недостаточностью классической логики по отношению к неклассическим логикам программирования, а также с некоторыми интуитивными ошибками при оценке истинности логических и вероятностных формул.

Автономность развиваемых модулей напрямую следует из принципа равноправия параметров с учетом принципа универсальности.

Части данных, любые подформулы, могут быть значениями параметров функции. Для любого заданного набора из одного или группы выбранных фрагментов можно представить функцию, параметрами которой могут быть эти выбранные фрагменты, после замены которых получается данное, эквивалентное исходному. Это позволяет использовать концепцию выбора проекций или слоёв программы, частичного вы-

числения, или факторизации программы, а также раздельной компиляции модулей, возможно по существу определившей долгую жизнь языка Fortran. Следует отметить, что в Lisp-системах обычно компилируются функции, а не целостные программы. Кратность применения модулей, процедур или функций при разработке новых СП обычно намного выше, чем использование готовых программ. Следует учесть, что изготовление новых СП становится всё более критичным участком в современных ИТ. Благодаря факторизации можно выделять из программы проекции, аналогичные схеме программы, допускающей верификацию.

Любой отмеченный набор программных фрагментов может быть удален из данного, представляющего программу, и связан с определенным именем, чтобы обеспечить возможность восстановления исходного представления. Можно заметить, что параметры при вызове функции вычисляются на том же уровне иерархии, в общем контексте в соответствии с принципом неизменности данных. Следовательно, порядок, в котором были вычислены параметры, не имеет значения, он может быть произвольным. Именно это позволяет разложить программу на автономно разрабатываемые составляющие. Отсутствие взаимосвязей между составляющими удобно для независимого их развития и для их многократного использования в разных программах. Это пригодится при представлении параллельных потоков, ленивых или опережающих вычислений. Кратность использования таких составляющих может быть существенно выше, чем кратность применения программ. Повышенная кратность использования составляющих модулей, подобно библиотекам стандартных процедур, на практике вызывает высокий уровень доверия. Можно сказать, что программа может быть представлена в виде, факторизованном по различным параметрам, в зависимости от цели ее преобразования и дальнейшего развития. За счет обратимости действий, основанной на неизменности данных, становится более удобным при отладке программы довести её до необходимого соответствия постановке задачи.

Следствия поддержки прагматических принципов в системах ФП формируют интуитивные образы, такие как непрерывность процесса (бесконечность), обратимость действий и унарные функции, которые обеспечивают основу для построения функциональных моделей. Кроме того, комплекс семантических и прагматических принципов обеспечивает поддержку подготовки функциональных моделей программ для организации параллельных вычислений, которые могут быть сведены к комплексам независимых, недетерминированных потоков.

Непрерывность процессов интуитивно следует из прагматической поддержки принципа гибкости ограничений.

После выполнения любой функции можно выполнить любую другую функцию. Команда STOP не является функцией. У нее нет аргументов или результатов. Это просто сигнал процессору перестать работать. При выполнении любой функции можно одновременно выполнять другие функции, также и перед выполнением функции можно выполнять другие функции. Это позволяет значительную часть работы программы основывать на модели неограниченной памяти без особого беспокойства о ее границах и разнообразии характеристик скорости доступа к различным структурам данных. Во многих языках ФП поддерживается имитация работы с бесконечными структурами данных [14, 15].

Обратимость действий основана на иллюзии неизменяемости данных, механизмы которой скрыты в системе программирования.

После выполнения любой функции можно вернуться к точке ее вызова. Любая функция может повторяться с теми же параметрами, ее можно проинтерпретировать иначе

или вместо нее можно выполнить любую другую функцию. Применение обратимости действий почти не требует заботы во время подготовки программы и основной части отладки. Это позволяет поддерживать механизм мемоизации функций для ранее обработанных аргументов. Фактически необходимые изменения данных, такие как повторное использование памяти, просто автоматизированы. Программист может позволить себе не вмешиваться в реализацию таких средств, пока нет проблем с производительностью.

Унарные функции связаны с принципом строгого результата.

Для любой функции с произвольным количеством параметров можно построить ее эквивалент с одним параметром, являющимся структурой из первоначальных параметров. Поскольку результаты часто являются аргументами объемлющих функций, логически возникает сопутствующий принципу строгого результата принцип унарных функций. Кроме того, возможность перейти от списка параметров или результатов к одному аргументу или строгому результату и обратно позволяет отойти от обычной схемы операций, которая сопоставляет два операнда одному результату, к операциям, которые сопоставляют один набор операндов другому набору результатов.

2.5. Приложения для параллельных вычислений

Параллелизм опирается на общий комплекс семантических и прагматических принципов, который позволяет при необходимости рассматривать любой объем представленных данных и при необходимости реорганизовывать пространство потоков. Такой комплекс делает ФП удобным для работы с программами, направленными на организацию параллельных процессов. Прежде всего, это принцип равноправия, который гарантирует одинаковый контекст при вычислении параметров функции. Становится возможным представлять независимые потоки и объединять их в многопоточные или многопроцессорные программы в общий программный комплекс. Кроме того, в параллелизме используются принципы строгого результата и универсальности, что позволяет при необходимости учитывать любое количество представленных потоков и реорганизовывать пространство потоков. Чистое ФП не совсем удобно для моделирования взаимодействующих и императивно синхронизированных процессов. Поэтому в дальнейшем потребуется определенное уточнение принципов, упрощающее подготовку высокопроизводительных программ параллельных вычислений.

Оттеснение маловероятных вычислений направлено на предотвращение затрат на выполнение чрезмерного количества потоков, соответствующих слишком редким ситуациям.

Принцип универсальности имеет два аспекта: равноправие программ с данными и полноту определений функций. При решении задач параллельных вычислений сохраняется важность равноправия программ с данными, традиционно востребованная в задачах операционных систем и управления процессами. Полнота функций, удобная для построения программ из универсально отлаженных модулей, может создавать проблемы из-за увеличения количества потоков в многопроцессорных программах, связанных с различными редкими диагностическими ситуациями.

Любой фрагмент, выполнение которого маловероятно или невозможно, может быть перемещён из представления функции в отложенное действие. Ветви для практически неактуальных ситуаций можно перенести в отладочную версию. Иногда эту проблему преодолевают путем выбора выражений, не требующих ветвления, чаще — путем про-

верки типов данных. Объем необходимой диагностики можно частично сократить за счет статического анализа типов данных.

Балансировка нагрузки сокращает реальное время выполнения многопоточной программы, что можно рассматривать как распространение принципа гибкости на временные ограничения.

Ленивые или преждевременные вычисления дают возможность быстро и оперативно перераспределять нагрузку. Сложное определение функции иногда можно свести к двум функциям, первая из которых выполняет часть определения, откладывая выполнение остальных, а вторая возобновляет выполнение отложенной части. Возможно выполнение второй функции произойдет одновременно с первой или даже начнётся раньше. В языке mpC предлагается объем вычислений оперативно перераспределять при обнаружении неравномерной загрузки процессоров [20].

Самоприменение в виде рекурсивных функций обычно рассматривается как осложнение, которое приводит к опасному росту стека. Здесь следует отметить, что многие системы ФП предлагают ряд практичных решений. Кроме того, в ФП существуют отложенные действия, мемоизация, восходящая рекурсия, методы динамического программирования и оптимизация рекурсий за счет сведения к циклам, что во многих случаях позволяет практически устранить чрезмерное разбухание стека. А поддержку работы со стеком в рамках принципа гибкости ограничений удаётся поддерживать более эффективно, чем в большинстве языков и систем программирования [16]. Да и факторизация программ на схемы и фрагменты позволяет разделять компоненты по уровню сложности отладки и наследовать корректность ранее отлаженных модулей. Используются функции, не требующие предварительного вычисления параметров, например макротехника.

Пространство итераций используется в качестве основного параметра для управления распараллеливанием циклов.

Любое конечное множество может работать как пространство итераций для определенной на нем функции. Если есть набор данных, на котором вычисление функции на одном элементе не требует ее результатов на других, то использование такого набора в качестве пространства итераций удобно для одновременного выполнения этой функции на всех элементах набора. Подобная техника распространения операций и функций имеется в языках APL, Альфа, БАРС и Sisal [23–25]. Роль равноправия параметров растет, она обеспечивает решение проблем реорганизации потоков при настройке на различные конфигурации многопроцессорных систем, требующих декомпозиции программы на схемы и фрагменты. Техника и концепция пространств итераций убедительно поддерживаются в языке Sisal, в котором такие пространства строятся над перечислимыми множествами с использованием операций скалярного и декартового произведения [24].

Несколько сложнее с прагматическими принципами, требующими пересмотра системных решений на уровне разработки СП. Здесь важную роль играет автоматическое распараллеливание, реже встречаются идентичность повторных прогонов и представление многоконтактных фрагментов.

Автоматическое распараллеливание заключается в извлечении из программы автономных частей, допускающих независимое выполнение.

Если существует функция с известным временем выполнения, которую можно разложить на две или более функций так, что время выполнения каждой из них будет заметно меньше. тогда, если они независимы, они могут быть выполнены. в произвольном порядк или одновременно, и время выполнения исходной функции станет меньше.

Идентичность повторных прогонов для целей отладки программ и измерения их производительности.

Переход к суперкомпьютерам показал, что при слишком большом числе процессоров исчезает нужная для отладки и измерений возможность пронаблюдать повторное исполнение программы. При очередном прогоне могут быть сбои на других процессорах. Функциональное программирование здесь может допускать специальные интерпретации программы, учитывающие протоколы и результаты ранее выполненных прогонов с отслеживанием идентичности исполнения, а также анализ выполнения программ с искажениями [26].

Многоконтактные фрагменты, такие как схемы управления или операции, имеющие несколько операндов и дающие более одного результата, на первый взгляд, противоречат принципу строгого результата.

Однако возможность перехода от строгого результата позволяет строить многопоточные функции, которые принимают параметры от ряда одноранговых потоков и генерируют ряд результатов в терминах числа потоков. Таким образом, можно, как и в функциональном языке параллельного программирования Sisal, переключиться на операции, которые отображают одну строку операндов в другую строку результатов [24]. Это может соответствовать структуре некоторых особо эффективных аппаратных узлов и, таким образом, допускать представление более эффективных решений.

2.6. Повышение производительности

При переходе к многократно применяемым программам и параллельным вычислениям успех приложения и производительность программ становятся важнее, чем их формальная корректность и эффективность. Чистое ФП можно рассматривать как метод функционального моделирования для создания прототипов решения сложных проблем. Более широкая парадигма производственного ФП позволяет переходить от таких функциональных моделей или прототипов к более эффективным структурам данных, принимая практичные решения и компромиссы по их обработке в зависимости от реальных условий, если это необходимо.

В дополнение к принципам и следствиям в реальных языках программирования и системах производственное ФП обычно включает механизмы практичного компромисса, которые внешне в языке программирования выглядят как специальные функции. Например, Lisp 1.5, Clisp, Cmucl и другие члены семейства Lisp обычно предоставляют следующие функции такого рода:

— **контроль типов данных** смягчает *принцип универсальности* функциями статического и динамического анализа типов данных;
— **схемы циклов**, моделирующие несколько расширенное множество знакомых механизмов управления вычислениями, позволяют преодолеть типичные опасения по поводу сложности реализации *принципа самоприменения*;
— **возможность восстановления данных** при использовании деструктивных функций, имеющих безопасные аналоги, позволяет исключить чрезмерное потребление памяти, частично отклоняясь от *принципа неизменности данных*;
— **программируемые прогнозы** объема памяти и скорости выполнения позволяют по мере необходимости выполнять распределение памяти, нейтрализуя грубоватые механизмы *принципа гибкости ограничений*;
— **псевдофункции**, помимо генерации результата, выполняют нагрузку в виде воздействия на внешнюю память или взаимодействия с устройствами, включая опе-

Таблица 1. Ключевые взаимосвязи принципов функционального программирования

	Семантика	**Прагматика**
Принципы	универсальность, самоприменение, равноправие параметров	гибкость ограничений, неизменяемость данных, строгость результата
Следствия	метапрограммирование, верификация, автономность развиваемых модулей	непрерывность процессов, обратимость действий, унарные функции
Приложения к параллельным вычислениям	оттеснение мало вероятных вычислений, балансировка нагрузки, пространства итераций	автоматическое распараллеливание, идентичность повторных прогонов, многоконтактные фрагменты
Практичный компромисс	контроль типа данных, схемы циклов, восстановление данных	программируемый учет прогнозов, псевдофункции доступа к устройствам, мемоизация

рации ввода-вывода и файловые операции, что несколько влияет на *принципы равноправия параметров и неизменяемости данных*;

— **мемоизация** позволяет радикально снижать сложность многократно повторяемых вычислений, делая понятную уступку успешному опыту против *принципов строгого результата и верификации*. Общее взаимодействие принципов, следствий, приложений и практичных компромиссов в системах ФП представлено в таблице 1.

Таким образом, в рамках ФП можно учитывать особенности параллельных вычислений, влияющие на выбор методов их решения в зависимости от приоритетов в выборе языковых средств и возможностей реализации. Парадигмальные ошибки, обнаруженные при работе знакомых СП на современном многопроцессорном и сетевом оборудовании, показывают, что некоторые из них были просто невидимы до появления сетей, мобильных устройств и суперкомпьютеров. Следствия семантических и прагматических принципов ФП и высокая моделирующая сила аппарата функций, дополненная специальными функциями практичных компромиссов, позволяют дополнять основные парадигмы параллельных вычислений для практической работы по повышению производительности программы.

3. ПАРАДИГМЫ ЯЗЫКОВ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

Рассматривая идеи ФП параллельных вычислений, следует отметить, что уже создано значительное количество языков параллельного программирования. Возможно, что одной из причин сложности разработки программ для параллельных вычислений является их скрытая мультипарадигмальность [13]. Чтобы разработать программу для параллельных вычислений, многое должно быть продумано разными способами одновременно в разных парадигмах, удобных для решения отдельных подзадач без возможности решения полного комплекса подзадач в общей среде. Очевидно, что задачи масштабирования вычислений, синхронизации взаимодействий в многопоточных программах, представления естественного асинхронного параллелизма и достижения высокой производительности программы весьма различны [13]. Отдельная задача — ознакомиться с явлениями параллелизма, чтобы дать специалистам интуитивное представление о ме-

тодах решения задач, зависящих от непривычных феноменов. Новые работы по ФП, по сути, направлены на поиск более эффективных решений задач параллельного программирования [27].

Создано заметное количество языков и систем программирования, позволяющих решать некоторые из этих задач в рамках определенных парадигм, поддержка которых представлена на разных языках программирования. Для каждой из трудно решаемых задач параллельных вычислений уже сформирована отдельная удобная парадигма ее решения и создан ряд ЯП, поддерживающих эту парадигму. Любая из таких парадигм может быть дополнена моделями и методами ФП. Различие между парадигмами проявляется в упорядочении важности средств и методов, используемых при решении отдельных задач, другие задачи требуют иного упорядочения.

Обычно на каждом этапе разработки программы используется одна парадигма. Соответственно, в каждом языке программирования есть одна ведущая парадигма. Ряд таких ЯП (Kotlin, APL, VAL, Sisal, Go, Haskell, Erlang, F#) изначально относят к функциональным Другие содержат подъязыки, позволяющие представлять программы в функциональном стиле (Java, Scala). Требования к решению достаточно сложных задач параллельных вычислений связаны с рядом различных задач, что влечет за собой необходимость использования разных парадигм на отдельных этапах их создания и фазах их жизни. При переходе к технологии программирования важно гарантировать получение практических результатов, для чего требуется поддержка всего спектра парадигм, используемых на разных этапах разработки программы, составляющих ее жизненный цикл. Не случайно, что большинство новых ЯП мультипарадигмальны, поддерживают 3-6 парадигм, включая ФП (Go, Rest, Rust).

Сложность использования разных парадигм для решения одной проблемы обычно минимизируют созданием многоязычных систем, которые позволяют при необходимости переходить от одного языка к другому, от одной парадигмы к другой без затрат на освоение различных интерфейсов и систем. Это показывает целесообразность создания мультипарадигмального языка параллельных вычислений, одновременно поддерживающего все основные парадигмы параллелизма.

Как показывает опыт языков BARS и Haskell, в таких случаях удобно разложить определение языка программирования на отдельные подъязыки, которые поддерживают основные парадигмы или монады, нацеленные на конкретные модели для подготовки и представления программ [14, 23]. Важно, чтобы на каждом этапе разработки программы использование одной парадигмы было локализовано, характеризуясь относительно небольшим набором инструментов и методов в рамках одного образа мышления. Каждая парадигма имеет собственное содержание категорий семантических систем и свой порядок их роли в процессе программирования. Рекомендация раздельной реализации парадигм возможна на уровне семантики определения языка программирования, на уровне прагматики она отступает из соображений эффективности или трудоёмкости.

Высокопроизводительное программирование требует перехода от однократного выполнения программы к учету перспектив ее многократного использования и улучшения [26]. Это дает возможность использовать недостаточно используемые мощности многопроцессорного комплекса, выполняя отдельные фрагменты программы для предстоящих вычислений с учетом данных, которые могут потребоваться при будущих запусках программы. Принятие решения в таких целях начинается со схем и моделей вычислений, возможно, с использованием общей памяти, но с приоритетом локальной памяти. Управление программой может учитывать возможность повторного выполнения программы

при ее отладке и применении, включая преимущества наследования результатов между ее запусками и сравнения измеряемых характеристик производительности версий программы. Это приводит к появлению ряда улучшенных версий программы, выбор одной из которых может рациональнее учитывать условия использования, в том числе результаты ранее выполненных вычислений, конфигурацию многопроцессорного комплекса и требования к качеству программы.

Языки программирования-долгожители, как и новые языки программирования нашего века, а также учебные языки обычно мультипарадигмальны. Успешная практика параллельного программирования требует поддержки всего спектра парадигм параллельных вычислений. Необходимо разрешить их развитие, пополнение и применение по мере необходимости, обеспечивая переход к следующей парадигме без изменения языков программирования и системного окружения.

Успех ФП на практике существенно зависит от выбора постановок задач, решения которых представлены системами функций, относительно простыми, не слишком трудоемкими и удобными для отладки. По степени изученности существенно различаются следующие категории постановок задач, влияющие на выбор методов решения задач и сложность их программирования:

— новые,
— экспериментальные,
— практичные,
— точные.

Главный критерий решения новых научных проблем — правильность и полнота решения. Именно на эти критерии нацелено ФП, которое является основным полигоном для исследования и разработки методов верификации программ. Практичные и точные задачи сильнее ориентированы на эффективность и практичность использования программ. Вот где проявляются преимущества императивного и объектно-ориентированного программирования.

Переход к экспериментам на суперкомпьютерах показал, что именно системные решения могут вносить значительный вклад в производительность параллельных вычислений, и такой вклад может превышать теоретические предсказания. Это можно рассматривать как обоснование необходимости более фундаментального подхода к программированию, особенно к системному программированию и его математическим основам. Характерной чертой системного подхода как ведущего метода программирования является переход к классам задач при содержательном анализе постановок задач. Границы классов устанавливаются при выборе процесса решения проблем. Это позволяет создавать высокопроизводительные программы с частичным использованием парадигмы ФП на уровне новых и научных постановок задач.

При создании, формировании и исследовании математических моделей как фундаментальной основы для решения особо сложных задач эффективности, надежности и безопасности программного обеспечения важную роль играет разработка моделей, связанных со временем и ресурсами, которые плохо представлены в курсах классической математики. Квалификация разработчика СП включает в себя формирование умения самостоятельно изобретать решения новых задач и навыков ответственного улучшения качества готовых решений, а также глубокие знания неклассической и фундаментальной математики, знакомство с современной физикой и достижения гуманитарных наук. Это слишком неудобно для образовательной системы, ориентированной на стойкие стереотипы, и противоречит методическим традициям.

4. ЗАКЛЮЧЕНИЕ

В феврале 2021 года прошла 22-я конференция, посвященная современным тенденциям функционального программирования [27]. Представленные доклады убедительно показали нацеленность ФП на решение многих задач организации параллельных вычислений.

Некоторые вопросы пока не получили практического ответа. Возникновение новых парадигм может быть связано с проявлением ряда новых проблем, решение которых до сих пор вызывает трудности. Особенности парадигм лишь частично выражаются на уровне семантики ЯП. Остальные — требования к прагматике ЯП. В их числе проблема, связанная с бедностью языковых решений для представления дисциплины доступа к многоуровневой гетерогенной памяти и протоколов взаимодействия между процессами, что требует некоторой доработки механизмов неизменности данных. Неизменяемость данных поддерживается на уровне локальной памяти потока, но вызывает проблемы при переходе к общей и внешней памяти, доступной разным потокам. Обычно зависимости между функциями, а следовательно, и между потоками реализуются в общей памяти. Возможно, что механизмы общей памяти требуют не столько неизменности данных, сколько возможности их восстановления, помимо математических аспектов работы с разнородной памятью, копиями, репликами и т. д., что аналогично динамическому редактированию сложных структур, уже проиллюстрированному в задачах работы с языками DSL [27].

В целом следует отметить, что следствия из семантических и прагматических принципов ФП и высокая моделирующая сила аппарата функций, расширенные специальными функциями практичных компромиссов, позволяют полезно дополнять основные парадигмы параллельных вычислений и развёртывать работы по повышению производительности программ. В этом плане существенный вклад дают мемоизация, отложенные и опережающие вычисления, мета-программирование, позволяющее строить специализированные проблемно ориентированные DSL-языки программирования. Многие новые производственные ЯП теперь содержат средства ФП. Всё это позволяет ожидать появление мультипарадигмального языка параллельного программирования, содержащего ФП среди основных парадигм в качестве метапарадигмы. Строго говоря, ФП способно выполнять роль проектно-конструкторского отдела для производственного программирования, что и даёт ответ на второй дискуссионный вопрос.

Вопрос: Какие особенности ФП способствуют его полезности при смене элементной базы?

Ответ: При смене элементной базы резко расширяется круг новых актуальных задач, решению которых способствуют принципы ФП, включая его приспособленность к быстрой отладке программ.

Вопрос: Может ли ФП стать доминирующей парадигмой производственного программирования?

Ответ: Доминирующей парадигмой производственного программирования ФП быть не должно, оно просто выполняет в нём свою достаточно важную, ещё не вполне признанную роль.

Список литературы

1. *McCarthy J.* LISP 1.5 Programming Manual. Cambridge: The MIT Press, 1963. 106 p.
2. *Backus J.* Can programming be liberated from the von Neumann style? A functional stile and its algebra of programs. Commun. ACM 21, 8, 1978. P. 613–641.

3. *Хендерсон П.* Функциональное программирование. М.: Мир, 1983. 349 с.
4. *Лавров С. С.* Функциональное программирование // Компьютерные инструменты в образовании. 2002. № 3–4. С. 42–52.
5. *Лавров С. С., Городняя Л. В.* Функциональное программирование // Компьютерные инструменты в образовании. 2002. № 5. С. 49–58.
6. *Городняя Л. В.* Основы функционального программирования. М.: НОУ «ИНТУИТ», 2004. [Онлайн курс] URL: http://www.intuit.ru/studies/courses/29/29/info (дата обращения: 23.09. 2021).
7. *Городняя Л. В.* Первые реализации языка Lisp в СССР // Материалы второй Международной конференции Развитие вычислительной техники и ее программного обеспечения в России и странах бывшего СССР (SoRuCom–2011). Великий Новгород, 2011. С. 95–100.
8. *Лавров С. С.* Методы задания семантики языков программирования // Программирование. 1978. № 6. С. 3–10.
9. *Gorodnyaya L.* On the presentation of the results of the analysis of programming languages and systems // Proc. of the 20th Conf. Scientific Services & Internet (SSI-2018) September 17–22, 2018. Novorossiysk-Abrau, 2018. P. 152–166.
10. *Wegner P.* Concepts and paradigms of object-oriented programming // ACM Sigplan Oops Messenger, 1990. Vol. 1, № 1. P. 7–87. doi: 10.1145/382192.383004
11. *Gorodnyaya L.* Strategic Paradigms Of Programming, Which Was Initiated And Supported By Academician Andrey Petrovich Ershov // Proc. of 2020 5th Int. Conf. On The History Of Computers And Informatics In The Soviet Union, Russian Federation And In The Countries, SoRuCom 2020, 2020. P. 1–11. doi: 10.1109/SORUCOM51654.2020.9464972
12. *Воеводин В. В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.
13. *Городняя Л. В.* О Неявной мультипарадигмальности параллельного программирования // Труды XXIII Всероссийской научной конференции (20–23 сентября 2021 Г., Онлайн). М.: ИПМ им. М. В. Келдыша, 2021. С. 104–116. doi: 10.20948/abrau-2021-6
14. *Душкин Р. В.* Функциональное программирование на языке Haskel. М.: ДМК Пресс, 2016. 606 с.
15. *Сошников Д. В.* Функциональное программирование на языке F#. М.: ДМК Пресс, 2011.
16. *Филд А., Харрисон П.* Функциональное программирование. Перевод под редакцией В. А. Горбатова. М.: Мир, 1993. 638 с.
17. Пересмотренное сообщение об Алголе 68 = Revised Report of the Algorithmic Language ALGOL 68 / ред. А. ван Вейнгаарден и др.; пер. с англ. А. А. Берса. М.: Мир, 1979. 533 с.
18. *Лавров С. С.* Так что же такое информатика? // Компьютерные инструменты в образовании. 2000. № 1. С. 22–25.
19. *Цейтин Г. С.* Ассоциативное исчисление с неразрешимой проблемой эквивалентности // Проблемы конструктивного направления в математике. Тр. Матем. ин-та им. В. А. Стеклова АН СССР. 1958. Т. 52. С. 172–189.
20. *Мальцев А. И.* Алгоритмы и рекурсивные функции. М.: Наука, 1965. 392 с.
21. *Фихтенгольц Г. М.* Основы математического анализа. Том 1. М: Наука, 1968. 440 с.
22. *Ластовецкий А. Л.* Предварительное сообщение о языке программирования mpC // Вопросы кибернетики: Приложения системного программирования, Научный совет по комплексной проблеме «Кибернетика» РАН, Москва, 1995. С. 20–39.
23. *Котов В. Е.* МАРС: архитектура и языки для реализации параллелизма // Системная информатика. Вып. 1. Проблемы современного программирования. Новосибирск: Наука. Сиб. отделение, 1991. С. 174–194.
24. *Cann D. C.* SISAL 1.2: A Brief Introduction and tutorial. Preprint UCRL-MA-110620. Lawrence Livermore National Lab., Livermore — California, May, 1992. 128 p.
25. *Касьянов В. Н., Гордеев Д. С., Золотухин Т. А.* Система облачного параллельного программирования CPPS: визуализация и верификация Cloud Sisal программ. Новосибирск: НГУ, 2020, Конструирование и оптимизация программ.
26. *Бурдонов И. Б., Косачев А. С.* Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 2. Условия конечного полного тестирования // Вестник Томского государственного университета. 2011. № 2(15). С. 89–98.
27. *Koopman P., Michels S., Plasmeijer R.* Dynamic Editors for Well-Typed Expressions // Trends in Functional programming/ 22nd International Symposium, TFP 2021, February 17–19, 2021. Springer, LNCS 12834. P. 44–66.

Поступила в редакцию 25.08.2021, окончательный вариант — 23.09.2021.

Городняя Лидия Васильевна, кандидат физико-математических наук, доцент кафедры Программирование Механико-математического факультета НГУ, старший научный сотрудник Института систем информатики СО РАН, ✉ lidvas@gmail.com

Computer tools in education, 2021
№ 3: 57–75
http://cte.eltech.ru
doi:10.32603/2071-2340-2021-3-57-75

On Functional Programming

Gorodnyaya L. V.[1,2], PhD, Associate Professor, ✉ lidvas@gmail.com,
orcid.org/0000-0002-4639-9032

[1]Novosibirsk State University, 1, Pirogova str., 630090, Novosibirsk, Russia
[2]A. P. Ershov Institute of Informatics Systems SB RAS, 6, Acad. Lavrentjev pr., 630090, Novosibirsk, Russia

Abstract

The article is devoted to the description of the features of functional programming, considered as a methodology for solving new and research problems of applied and system programming. The technique of analysis and comparison of programming paradigms is involved, taking into account the priorities of decision-making in the process of developing programs. The methodology for comparing languages and programming paradigms is based on an informal definition of the term “programming paradigm”, according to which, when comparing paradigms, it is necessary to highlight the distinctive testable features that allow verification. As such signs, it turned out to be useful to use priorities in decision-making at different stages of studying the problem statement, and then developing and debugging a program for solving it. Accounting of priority allows you to predict the complexity of the application of programmable solutions, starting with planning, studying and organizing the development of long-lived programs.

The material of the article is somewhat debatable. The article gives a clear formulation of the principles of the functional programming paradigm, which distinguish it from other paradigms. Based on these principles, the derivation of consequences that allow the successful application of functional programming in solving complex problems is carried out, they are concretized to the tasks of organizing parallel computing and improving the performance of programs created within the framework of the functional programming paradigm. The complexity of creating programs for solving new problems is shown on the example of parallel computing and the requirements for a universal multi-paradigm language for parallel computing are described. For functional programming problems, the correctness and completeness of solutions is more important than the efficiency and productivity of the resulting programs. It is this choice of priorities that allows functional programming to be considered as a general technique for preparing prototypes or functional models. We can say that functional programming acts as a design office for production programming.

Keywords: *functional programming, programming language, programming paradigm, programming system, pragmatics, meta-paradigm, new problems, parallel computing.*

Citation: L. V. Gorodnyaya, “On Functional Programming,” *Computer tools in education*, no. 3, pp. 57–75, 2021 (in Russian); doi:10.32603/2071-2340-2021-3-57-75

References

1. J. McCarthy, *LISP 1.5 Programming Manual*, Cambridge, UK: The MIT Press, 1963.
2. J. Backus, “Can programming be liberated from the von Neumann style? A functional stile and its algebra of programs,” *Commun. ACM*, vol. 21, no. 8, pp. 613–641, 1978; doi: 10.1145/359576.359579
3. P. Henderson, *Functional programming*, Moscow: Mir, 1983 (in Russian).
4. S. S. Lavrov, “Functional programming,” *Computer Tools in Education*, no. 3–4, pp. 81–93, 2002 (in Russian).
5. S. S. Lavrov and L. V. Gorodnyaya, “Functional programming. Principles of Lisp Implementation,” *Computer Tools in Education*, no. 5, pp. 49–58, 2002 (in Russian).
6. L. V. Gorodnyaya, “Functional Programming Fundamentals,” in *NOU «INTUIT»*, [Online Course], 2004 (in Russian). Available: http://www.intuit.ru/studies/courses/29/29/info
7. L. V. Gorodnyaya, “The first implementations of the Lisp language in the USSR,” in *Proc. of Materialy vtoroi Mezhdunarodnoi konferentsii Razvitie vychislitel’noi tekhniki i ee programmnogo obespecheniya v Rossii i stranakh byvshego SSSR (SoRuCom–2011), Veliky Novgorod, Russia, 2011*, pp. 95–100, 2011 (in Russian).
8. S. S. Lavrov, “Methods for defining the semantics of programming languages,” *Programming and Computer Software*, no. 6, pp. 3–10, 1978 (in Russian).
9. L. Gorodnyaya, “On the presentation of the results of the analysis of programming languages and systems,” in *Proc. of the 20th Conf. Scientific Services & Internet (SSI-2018) Novorossiysk-Abrau, Russia, September 17–22, 2018*, pp. 152–166, 2018 (in Russian).
10. P. Wegner, “Concepts and paradigms of object-oriented programming,” *ACM Sigplan Oops Messenger*, vol. 1, no. 1, pp. 7–87, 1990; doi: 10.1145/382192.383004
11. L. Gorodnyaya, “Strategic Paradigms Of Programming, Which Was Initiated And Supported By Academician Andrey Petrovich Ershov,” in *Proc. of 2020 5th Int. Conf. On The History Of Computers And Informatics In The Soviet Union, Russian Federation And In The Countries, SoRuCom 2020*, pp. 1–11, 2020 (in Russian); doi: 10.1109/SORUCOM51654.2020.9464972
12. V. V. Voevodin, “Parallel’nye Vychisleniya,” St. Petersburg, Russia: BKhV-Peterburg, 2002 (in Russian).
13. L. V. Gorodnyaya, “On implicit multiparadigmality of parallel programming,” in *Proc. of 23rd Scientific Conference “Scientific Services & Internet – 2021”, Online*, Moscow.: IPM im. M.V.Keldysha, pp. 104–116, 2021 (in Russian); doi.org/10.20948/abrau-2021-6
14. R. Dushkin, *Funktsional’noe programmirovanie na yazyke Haskel*, Moscow: DMK-Press, 2016 (in Russian).
15. D. V. Soshnikov, *Funktsional’noe programmirovanie na yazyke F#*, Moscow: DMK-Press, 2011 (in Russian).
16. A. J. Field and P. G. Harrison, *Functional Programming*, V. A. Gorbatova, ed., Moscow: Mir, 1993 (in Russian).
17. A. van Wijngaarden et al., eds., *Revised Report of the Algorithmic Language ALGOL 68*, Moscow: Mir, 1979 (in Russian).
18. S. S. Lavrov, “Tak chto zhe takoe informatika?,” *Computer Tools in Education*, no. 1, pp. 22–25, 2000 (in Russian).
19. G. S. Tseitin, “An associative calculus with an insoluble problem of equivalence,” *Problems of the constructive direction in mathematics. Part 1, Trudy Mat. Inst. Steklov*, vol. 52, Moscow, Leningrad, Russia: Acad. Sci. USSR, pp. 172–189, 1958 (in Russian).
20. A. I. Maltsev, *Algorithms and recursive functions*, Moscow: Nauka, 1965 (in Russian).
21. G. M. Fichtenholz, *The Fundamentals of Mathematical Analysis*, vol. 1, Moscow: Nauka, 1968 (in Russian).
22. A. L. Lastovetskii, “Predvaritel’noe soobshchenie o yazyke programmirovaniya mpC,” *Voprosy kibernetiki: Prilozheniya sistemnogo programmirovaniya, Nauchnyi sovet po kompleksnoi probleme "Kibernetika"*, Moscow: RAN, pp. 20–39, 1995 (in Russian).

23. V. E. Kotov, “MARS: arkhitektura i yazyki dlya realizatsii parallelizma, Sistemnaya informatika,” *No. 1: Problemy sovremennogo programmirovaniya*, Novosibirsk, Russia: Nauka. Sib. dep., pp. 174–194, 1991 (in Russian).
24. D. C. Cann, *SISAL 1.2: A Brief Introduction and tutorial. Preprint UCRL-MA-110620*, Livermore, California: Lawrence Livermore National Lab., 1992.
25. V. N. Kas’yanov, D. S. Gordeev, and T. A. Zolotukhin, *Sistema oblachnogo parallel’nogo programmirovaniya CPPS: vizualizatsiya i verifikatsiya Cloud Sisal programm*, Novosibirsk: NGU, Konstruirovanie i optimizatsiya programm, 2020 (in Russian).
26. I. B. Burdonov and A. S. Kosachev, “Semantiki vzaimodeistviya s otkazami, divergentsiei i razrusheniem. 2. Usloviya konechnogo polnogo testirovaniya,” *Vestnik Tomskogo Gosudarstvennogo Universiteta*, no. 2 (15), pp. 89–98, 2011 (in Russian).
27. P. Koopman, S. Michels, and R. Plasmeijer, “Dynamic Editors for Well-Typed Expressions,” in *Trends in Functional Programming. TFP 2021. Lecture Notes in Computer Science*, vol. 12834, Springer, Cham., pp. 44–66, 2021; doi: 10.1007/978-3-030-83978-9_3

Received 25-08-2021, the final version — 23-09-2021.

Lidia Gorodnyaya, PhD, Associate Professor of the Department of Programming of the Faculty of Mechanics and Mathematics of Novosibirsk State University, Senior Researcher at the A. P. Ershov Institute of Informatics Systems SB RAS, ✉ lidvas@gmail.com