ИПМ им.М.В.Келдыша РАН

Абрау-2016 • Труды конференции

Л.В. Городняя

Парадигмальная декомпозиция определения языка программирования

Рекомендуемая форма библиографической ссылки

Городняя Л.В. Парадигмальная декомпозиция определения языка программирования // Научный сервис в сети Интернет: труды XVIII Всероссийской научной конференции (19-24 сентября 2016 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2016. — С. 115-127. — URL: http://keldysh.ru/abrau/2016/21.pdf

Размещена также презентация к докладу

Парадигмальная декомпозиция определения языка программирования

Л.В. Городняя

Институт систем информатики СО РАН

Аннотация. Доклад посвящён проблеме совершенствования современных систем программирования и создания новых языков программирования, нацеленных на эффективное решение задач разработки надёжных и удобных информационных систем.

Ключевые слова: парадигмы программирования, декомпозиция программ, реализационная прагматика, определение языков программирования

Введение

Объём описаний новых языков программирования (ЯП) нередко превышает пятьсот страниц (Java, Haskell, C#, Scala, F#) [23,28,29], а разрабатываемые определения новых стандартов на традиционные ЯП приближается к полутора тысячам страниц (C++) с пропорциональным возрастанием количества тестового материала при отладке систем программирования (СП) для таким образом определённого ЯП. Кроме того ряд проблем реализации СП связан со слабой структурированностью таких описаний и определений, не вполне соответствующей логике разработки и конструирования программных систем [31,47,49]. Практика минимизации трудозатрат на создание инструментальной поддержки полного жизненного цикла задач автоматизации программирования препятствует успеху компонентного подхода к разработке СП, требующего более практичных критериев декомпозиции программ [6,16,20,26,36,37]. Ещё Фр.Брукс в своей знаменитой книге отмечал, что трудоёмкость решения задачи в виде компонента программы примерно в три раза выше трудоёмкости разработки автономной программы, а трудоёмкость создания компонента для многопрограммного комплекса ещё выше [4].

Сложившиеся исторически подходы к реализации СП опираются на инструментальную поддержку конструирования лексических и синтаксических анализаторов текста программы, дополненных средствами создания перевода программы на промежуточный язык (фронт-энд) и генератора исполнимого кода программы (бек-энд) [24,46]. Статистика использования таких средств при реализации новых СП показывает, что массово применяются конструкторы анализаторов, достаточно популярны средства создания фронт-энда (Clang-API,

Lex, YACC и т.п.) и существенно реже используются средства создания бек-энда (LLVM). Таким образом, доминирует горизонтальное слоение СП, разработка которых сводится к не слишком трудоёмкой реализации лексики, синтаксиса, семантики отдельно от наследуемой прагматики, дополненной комплексом библиотечных модулей.

Практичность применения инструментов конструирования СП зависит от эволюции определения ЯП и требований к эффективности и надёжности его СП, неизбежно происходящей при сроках разработки, превышающих полгода [45]. Такого рода проблемы призван решать компонентный подход к программированию, зарекомендовавший себя во многих областях приложения информационных систем [15] благодаря возможности локализовать область модификации кода в зависимости от изменения постановки решаемой задачи, что можно назвать требованием независимого развития компонент. Кроме того, долгоживущие и новые ЯП характеризуются тенденцией к мультипарадигмальности, что означает требование поддержки вариантов реализации, используемых в зависимости от условий применения программ. Такие альтернативные подходы к обработке информации, сложившиеся при создании и применении ЯП, принято называть парадигмами программирования (ПП) [48]. Некоторые ПП можно рассматривать как типовые концептуальные подъязыки, но встречаются и ПП, выглядящие как слабо формализованные особенности условий эксплуатации программ и прагматики ЯП, зависящей от квалификационного уровня пользователей. Реализация таких особенностей обычно сказывается на всех уровнях горизонтального слоения СП. Методика вертикального слоения программ, в свое время предложенная А.Л.Фуксманом [31], применительно к СП здесь названа парадигмальной декомпозицией.

Парадигмальная декомпозиция рассматривается как метод выбора критериев для согласованного выделения типовых компонентов в описании ЯП и определении СП, включая спецификацию эксплуатационных аспектов и реализационной прагматики, обеспечивающих варьирование используемых ПП в зависимости от схемы жизненного цикла решаемых задач и требований к эффективности и производительности программируемых решений.

1. История вопроса

Вскоре после появления раздельной компиляции, воплощённой при разработке языка Fortran, была представлена чёткая концепция решения проблем разработки систем программирования по Венской методике определения языков программирования на базе абстрактного синтаксиса (АС) и абстрактной машины (АМ) [20,37]. Убедительный эксперимент по воплощению этого метода фактически произошёл при реализации СП для языка Lisp, использующей взаимодействие интерпретатора и компилятора при организации памяти со «сборкой мусора» с выделением базовой семантики в виде подъязыка Pure Lisp [38]. Роль абстрактного синтаксиса выполнил язык S-выражений, реализованный в виде списков и позднее послуживший основой для парадигмы

функционального программирования [7,17,19,23,25,27]. В дальнейшем работы по языкам Алмо, Альфа, Сигма и внутреннему языку системы БЕТА показали перспективность многопроходной компиляции и оптимизации программ при ограниченных ресурсах [2,13,14,24]. Появление синтаксических конструкторов YACC и LEX над языком Си определило доминирование решения проблем машинно-зависимого переноса через сведение реализационной прагматики разных языков к ядру языка Си. Объектно-ориентированное программирование на базе классов объектов в C++ дало технику уточнения программных компонентов по мере эволюции постановки задачи [21,22]. Целенаправленная попытка компонентного подхода к проблеме разработки СП на базе специально созданного внутреннего языка предпринята в проекте .Net и в других подобных проектах [26]. По всем этим линиям был достигнут определённый успех, но не образовал достаточную базу программотехники для авторов новых ЯП [40].

2. Синтаксис, семантика и прагматика

Проблема определения ЯП и СП тщательно проработана в Венской методике определения языков программирования [37]. Основная идея — использование *абстрактного синтаксиса (АС)* и *абстрактной машины (АМ)* при определении *семантики* языка программирования и отделения её от реализационной прагматики на конкретной машине (КМ). Конкретный синтаксис (КС) языка отображается в АС, АМ может быть реализована с помощью КМ, причем и отображение, и реализация могут иметь небольшой объем и невысокую сложность.

Диаграмма	*Пояснение*
КС ↔ **АС** ↓ **АМ** → КМ	Существует отображение конкретного синтаксиса (КС) в абстрактный (АС) и обратно. АС отображается в абстрактную машину (АМ). АМ реализуется с помощью конкретной машины (КМ).

Схема 1. Абстрактная декомпозиция определения ЯП по Венской методике.

Если КС удаётся нормализовать, то его перевод в АС и обратно (КС ↔ **АС**) можно построить автоматически [3], что можно сделать синтаксическими конструкторами типа YACC, LEX. Сущность определения языка концентрируется в виде так называемой универсальной семантической функции языка (УФ : **АС → АМ**), выполняющей переход от *абстрактного синтаксиса к абстрактной машине — трансляцию* (интерпретацию и/или компиляцию) [20]. УФ и АМ образуют определение функциональной и операционной семантики ЯП. Такая абстрактная декомпозиция достаточна для установления первичной границы между фронт-эндом и бек-эндом [11,46]. Для производственных ЯП определение УФ достаточно сложно, что приводит к ошибкам в оценке трудоёмкости не менее чем в два раза. По этой же причине

нередко определение семантики ЯП сводят к определению АМ, реализация которой чувствительна к навыкам низкоуровневого программирования и потому её реализационная прагматика часто остаётся без формализации. Выбор АМ существенно влияет на пространство допустимых КМ [11,24]. Следует отметить, что РП обычно связана с парадигмами ЯП.

3. Базовая семантика

Граница между АС и АМ может быть уточнена введением уровня базовых средств (БС), что гарантирует при тривиальности перехода БС ↔ АМ, профилактику усложнённости АМ и взаимозаменяемость АМ и БС на первых шагах раскрутки реализации СП.

Диаграмма	*Пояснение*
КС ↔ **АС** ↓ **БС** ↘ ↕ КМ **АМ** ↗	При отображении АС в АМ выделяется уровень базовых средств (БС), просто отображаемых на АМ и обратно. Переход к КМ может быть выполнен и как реализация БС.

Схема 2. Предварительная декомпозиция определения ЯП с выделением уровня базовых средств (БС).

В состав БС включают операции ЯП и средства управления вычислениями, несводимые к более простым средствам ЯП. Выделение БС гарантирует чёткое отделение языково ориентированного фронт-энда от машинно-ориентированного бек-энда, что даёт минимизацию трудоёмкости при переносе СП на новую аппаратуру. Такая декомпозиция была выполнена в первых реализациях языков Lisp и C [20,21,38]. Можно ограничить БС одной областью данных, рассчитывая, что операции над другими областями данных несложно включить как дополнительные команды, расширяющие АМ [38]. Тем не менее, даже для не слишком сложных ЯП число одноуровневых функций, образующих фронт-енд, легко превосходит тысячу, дополненную рядом библиотек и пакетов, нередко обладающих общим функциональным назначением при различных механизмах реализации [23,28,29]. Попытки улучшения СП обычно требуют повторного программирования заметного числа взаимосвязанных функций над общими данными или дополнения СП новой библиотекой, нацеленной на повышение производительности программ [5]. В последнем случае программист получает рекомендацию отредактировать ранее отлаженную программу [5,8].

4. Прикладные семантики

Сложность целостной реализации универсальной функции **УФ: АС →БС → АМ** преодолевается декомпозицией её на прикладные семантические системы (ПС) — семантическая декомпозиция [16]. Прикладные семантические системы {ПС1,…,ПСn} представляют отдельные механизмы ЯП, в принципе эффективно сводимые к аппаратным решениям при реализации и развитии автомата АМ → КМ.

Диаграмма	*Пояснение*
КС ↔ АС = **{ПС1,…, ПСn}** **–----------** ↓ **БС** ↘ ↕ **КМ** **АМ** ↗	При анализе АС выделяются прикладные семантики (ПСi), реализация которых возможна с помощью БС. Предварительная реализация ПС может быть выполнена средствами реализуемого ЯП.

Схема 3. Семантическая декомпозиция определения ЯП с разложением на прикладные семантики (ПСi).

Это позволяет реорганизовать УФ в комплекс из более простых **УФi: ПСi →БС → АМ**. Примерно так устроено описание языка Basic, представляющее собой набор подъязыков, обеспечивающих доступ к отдельным средствам аппаратуры, пополнение состава которых выполняется добавлением соответствующего подъязыка, слабо связанного с остальными [21].

Комплект основных видов ПСi в качестве типовых компонент ЯП обычно предоставлен средствами работы с памятью, организации вычислений, обработки структур данных и управления вычислениями, реализация которых обладает общей спецификой. Возможно выделение средств диагностики, типового контроля, средств укрупнения действий и т. д. Такая семантическая декомпозиция ЯП имеет шансы достичь независимости развития соответствующих ей компонентов СП. Обычно ЯП содержит около 20-ти видов ПС, причём число альтернативных вариантов ПС одного назначения может приближаться к десятку как в ЯП Planner. Выделение таких семантических систем обусловлено различиями в схеме применения операций к операндам, связанными с реализационной прагматикой СП и наличием их аппаратной поддержки. Понятие «семантическая система» выделяет конкретный тип данных (ТД), набор базовых операций (БО) над ним и правило применения операций (ПО) к данным, возможно допуская пересмотр состава системы [16].

Интеграция новых компонентов ЯП в ранее реализованную версию СП требует включения ряда согласованных определений на всех горизонтальных уровнях определения ЯП. На уровне АМ это означает дополнение её регистрами и специальными командами.

5. Прагматика

Для перехода от определения ЯП к реализации СП ключевое значение имеет семантика, но для уяснения парадигм программирования (ПП), поддерживаемых ЯП, требуется понимание не только условий эксплуатации программ, но и реализационной прагматики (РП), которая может быть не представлена в определении или стандарте ЯП, но подразумеваться традиционно или воспроизводится по прототипу. Выбор РП существенно влияет на пространство процессов выполнения программ.

Вычисления характеризуются объявлением 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 мощности множества допустимых значений, набору встроенных операций над ними, возможности конструировать новые операции и правила применения операций к данным.

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, приёма сообщений, синхронизации действий и взаимодействия потоков [18].

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, счётчиковые и программные методы повторного использования памяти для структур данных [30].

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).

Реализационная прагматика, затрагивая все уровни определения ЯП, в основном предоставляет решения в области конкретной организации

вычислений, уточняющей решения и принципы, провозглашенные в определении АМ. В первую очередь это относится к вопросам защиты областей памяти и их конечности, т.е. реагирования на дефицит памяти. Следует сразу особо отметить, что основные традиционные ПП апеллируют к операционной памяти, время жизни состояний в которой ограничено пределами времени исполнения программы. Обработка внешней и многоуровневой памяти представляется как побочные эффекты. Это создает проблемы перехода к большеобъёмным и распределённым данным.

Для основных ПП определение АМ сводится к системе команд (СК) над небольшим числом регистров, обычно от двух до шести. СК может быть получена как отображение части БС, а состав регистров и особенности их функционирования отражают реализационную прагматику обработки данных.

Диаграмма	*Пояснение*
КС ↔ АС = { **ПС1**,…, ПСn } ↓ ↓ ↓ БС = { **БО**1,…, БОn } ↕ ↕ ↕ **АМ** ~ < **СК1,…,** СКn > → КМ ~ ---------- ~ **регистры**	**ПС = <ТД, БО, ПО>** При анализе АМ для эффективной реализации ПСi определяются системы команд (СКi) над конкретными регистрами .

Схема 4. Прагматическая декомпозиция определения ЯП
по интерфейсу семантики с реализационной прагматикой.

Компоненты **УФi: ПСi →БОi → СКi** обретают более чёткую форму, но следует отметить, что разные СКi могут совпадать, обладать пересечениями или при реализации сводиться к различным парадигмальным вариантам, что требует дальнейшей детализации.

6. Основные парадигмы

К основным парадигмам программирования обычно относят императивное, функциональное, логическое и объектно-ориентированное программирования [8]. При сравнении императивного и функционального подходов к программированию, П. Лэндин (P.J. Landin) предложил специальную абстрактную машину SECD (подробное описание можно найти в [27]) в качестве АМ для ЯП функционального программирования. Машина SECD работает над четырьмя независимыми регистрами: стек для результатов, контекст для именованных значений, управляющая программа, резервная память (Stack, Environment, Control list, Dump). И аналогичная машина SECM для ЯП императивного программирования работает над четырьмя регистрами: стек для значений, контекст для аргументов функций, локальных переменных и регистра возврата, управляющая программа, общая память (Stack, Environment,

Control program, Memory). Для ЯП логического программирования определяется машина SECDR, дополняющее определение SECD регистром R для хранения не опробованных вариантов. При переходе от императивного программирования к ООП машина SECDM строится как включение резервной памяти D в машину SECM.

Множество прикладных семантических систем можно разбить на подмножества, реализуемые в рамках одинаковых парадигм. Обычно РП может поддерживать одну ПП, но встречаются ПСi, требующие совместного применения ряда парадигм. Например, в языке F# реализация ленивых вычислений выполнена на стыке функционального и объектно-ориентированного программирований [23].

Диаграмма	*Пояснение*
КС ↔ АС = { **ПС1**,…, ПСn } ↓ ↓ ↓ БС = { **БО**1,…, БОn } ↕ ↕ ↕ **АМ** ~ < **СК1**,…, СКn > ↕ ↕ < **РП1**,…, РПn> → КМ РПi = {РПij, ... }	Определение АМ дополняется отображением её СКi на ряд парадигмальных вариантов реализационной прагматики (РПi), представляющих собой подмножества комплекта поддержанных в СП парадигм.

Схема 5. Парадигмальная декомпозиция определения ЯП
с альтернативными вариантами реализационной прагматики.

Типовой компонент обретает вид:

<КСi ↔ПСi → БОi → СКi → РПij >

где i - обозначение прикладной семантики,

j – обозначение альтернативного варианта её реализации.

Практика определения ЯП на уровне фронт-энда порождает иллюзию, что одни парадигмы могут рассматриваться как уточнение других парадигм, например, императивное программирования дополняется до объектно-ориентированного. На уровне бек-энда или реализационной прагматики можно видеть разницу в организации обработки данных. Для поддержки ООП хранимые данные сопровождаются кодом сигнатуры, позволяющим в динамике контролировать применимость действий.

7. Новые парадигмы компьютерных языков

Поиск методов снижения трудоёмкости разработки массово востребованных программных приложений идёт через попытки исключить повторное программирование и отладку решений многократно решённых задач.

В своё время идея пакетов прикладных программ [15] потребовала резкого расширения программистского корпуса и создания инструментария машинно-зависимого переноса программ. Затем популярность объектно-ориентированного подхода в форме CORBA-технологий привела к проблеме формализации экспертного знания для разных областей приложения информационных систем [51].

Экстенсивное порождение предметно-ориентированных языков в XXI веке знаменует переход к методике конструирования компьютерных языков в качестве средства, позволяющего пользователю, умеющему работать с текстами, таблицами и визуальными оболочками на уровне Word-Excel и Visual-Studio, формировать грамматики, накрывающие пространство понятных решений текущих задач [32,33,39,41,42,44,52]. Происходящее в этом процессе уточнение такого пространства приводит к проблеме минимизации объёма изменений, возникающих в ранее сложившихся парадигмах программирования, нацеленных на борьбу за эффективность однопроцессорных программ над оперативной памятью, рассчитанных на надёжность хранения данных на носителях, допускающих защиту и контроль [1].

Теперь пафос системного программирования переходит к обработке большеобъёмных слабо структурированных данных, не вполне защищённых от искажений, допускающих распределённую и многопроцессорную обработку. Организация параллельных вычислений, конструирование проблемно-ориентированных языков (DSL), обработка big data, обеспечение надёжности Web-сервисов, распределённых баз данных и многое другое [5,12,21,34,35], пока программируемое как побочные эффекты, требует исследования и формализации специальных парадигм компьютерных языков, для которых возможно выделение типовых компонент, встраиваемых в различные информационные комплексы.

Заключение

Трудоёмкость реализации и жизнеспособность компонентов СП для долгоживущих, развивающихся и мультипарадигмальных ЯП зависит от того, удаётся ли декомпозировать реализационную прагматику ЯП на парадигма-зависимые варианты реализации отдельных прикладных семантик. Парадигмы обычно не расширяют друг друга, а противопоставляются, представляя разные точки зрения, допускающие интеграцию в общем пространстве.

Относящиеся к одной парадигме подъязыки или семантические системы разных ЯП легко сравнить на уровне структурированной таким образом операционной семантики, анализируя определения их абстрактных машин. Это позволяет с каждой парадигмой связать варианты реализационных особенностей, уточняющих типовые семантические системы и правила их взаимодействия, что можно назвать парадигмальной декомпозицией определения ЯП.

Следует отдельно отметить, что по мнению молодого поколения системных программистов наиболее серьёзные труды по созданию систем программирования были выполнены в 1970-е годы [40]. Сложившаяся в те годы полемика и противопоставление статики-динамики отражает уровень эксплуатационных характеристик того времени [36]. Теперь предстоит исследование новых эксплуатационных аспектов обработки данных и определение абстрактных машин, полезных при конструировании реализационной прагматики компьютерных языков на уровне бек-энда.

Благодарности

Автор выражает благодарность организаторам сайтов любителей программирования http://compiler.su и https://habrahabr.ru/, создавшим Интернет-сервис для изучения современных проблем программирования и распространения профессиональных знаний среди энтузиастов разработки языков программирования.

Литература

1. Андреева Т.А., Ануреев И.С., Бодин Е.В., Городняя Л.В., Марчук А.Г., Мурзин Ф.А., Шилов Н.В.. Компьютерные языки как форма и средство представления, порождения и анализа научных и профессиональных знаний. // Труды XV Всероссийской научно-методической конференции "Телематика-2008" − Санкт-Петербург, 2008. − С. 77-78.
2. Андрей Петрович Ершов - ученый и человек / Отв. ред. А.Г. Марчук. – Новосибирск: Изд-во СО РАН, 2006. – 504 с. (Наука Сибири в лицах)
3. Ахо А.В., Хопкрофт Дж.Э., Ульман Дж.Д. Структуры данных и алгоритмы. − М.: Вильямс, 2000. − 384 с
4. Фредерик Брукс. Мифический человеко-месяц, или Как создаются программные системы = The mythical Man-Month: Essays on Software Engineering. — Символ-Плюс, 2010. — 304 с. — (Профессионально). — 1500 экз. — ISBN 5-93286-005-7.
5. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.
6. Городняя Л.В. Об одном подходе к синтезу транслятора на примере языка Литтл. // Теория и практика системного программирования. − Новосибирск, 1977. − С. 60-71.
7. Городняя Л.В. Основы функционального программирования. – М.: Интернет-Университет Информационных технологий. − - URL: http://www.intuit.ru, 2004. − 272 с.
8. Городняя Л.В. Парадигмы параллельного программирования в университетских образовательных программах и специализации //

Всероссийская научная конференция "Научный сервис в сети Интернет: решение больших задач − Новороссийск-Москва, 2008. − С. 180-184.
9. Городняя Л.В. Парадигма программирования: курс лекций // Новосиб. Гос. Ун-т.- Новосибирск : РИЦ НГУ, 2015. - 206 с.
10. Городняя Л.В. Первые реализации языка *Lisp в СССР*. - URL: *www.computer-museum.ru/histsoft/**lisp**_sorucom_2011.htm*
11. Гуревич Ю. Последовательные машины абстрактных состояний охватывают последовательные алгоритмы. - В сб. Системная информатика. Вып 9. Формальные модели и модели информатики − Новосибирск: Изд. СО РАН, 2004. − С. 7-50.
12. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. − С-Пб.: ЕХП-Петербург, 2003, − 1104 с.
13. Крайнева И.А., Марчук А.Г. Игорь Васильевич Поттосин. Из истории новосибирской школы программирования (к 80-летию со дня рождения). Вестник НГУ, Серия: математика, механика, информатика. 2013, №1. С. 2-12.
14. Ирина Крайнева, Наталья Черемных. Альфа-язык и транслятор // Открытые системы. — 2014. — № 6. - URL: http://www.novsu.ru/file/867726 - Открытые системы. СУБД 2016 № 01
15. Лаврищева Е.М. Развитие отечественной технологии программирования. Кибернетика и системный анализ. 2014, том 50, № 3. - С.1-16. - URL: http://www.ispras.ru/publications/razvitie_otechestvennoy_tekhnologii_programmirovaniya.pdf
16. Лавров С.С. Методы задания семантики языков программирования. − Программирование, N 6, 1978. − С. 3-10.
17. Лавров С.С., Городняя Л.В. Функциональное программирование. Интерпретатор языка Лисп. //Компьютерные инструменты в образовании. С-Пб. 2002, N5.
18.Ломазова И.А. Вложенные сети Петри. – М.: Научный мир. 2004 − 207 с
19. Непейвода Н.Н. Стили и методы программирования. – М.: Интернет-Университет Информационных технологий. - - URL: http://www.intuit.ru/department/se/progstyles/, 2004
20. Оллонгрен А. Определение языков программирования интерпретирующими автоматами. − М.: Мир, 1977. − 288 с.
21. Пратт Т., Зелковиц М. Языки программирования. Разработка и реализация / Под общей редакцией А.Матросова. − СПб.: Питер, 2002. – 688 с.
22. Прехельт Л. Эмпирическое сравнение семи языков программирования. – М.: Открытые системы, 12(56), 2000. − С. 45-52.
23. Сошников Д. В. Программирование на F#. – М.: ДМК Пресс, 2011. – 192 с.
24. Степанов Г.Г. Пути обеспечения переносимости программ и опыт использования системы СИГМА // Трансляция и преобразование программ. – Новосибирск: ВЦ СО АН СССР, 1984. – 9 с.

25. Стивен Р. Палмер, Джон М.Фелсинг. Практическое руководство по функционально-ориентированной разработке ПО. − М.: Вильямс, 2002. − 299 с.
26. Уоткинс Д., Хаммонд М., Эйбрамз Б. – Программирование на платформе .Net. − М. Вильямс, 2003. − С. 367.
27. Хендерсон П. Функциональное программирование. – М.: Мир, 1983. – 349 с.
28. Хорстман К. Scala для нетерпеливых. – ДМК пресс, 2013. – 408 с. – ISBN 978-5-94074-920-2, 978-0-321-77409-5.
29. Хорстманн К. С. Java SE 8. Вводный курс = Java SE 8 for the Really Impatient. – М.: «Вильямс», 2014. – 208 с. – ISBN 978-5-8459-1900-7.
30. А.Ю. Филатов, В.В. Михеев // Препринт 179. Стратегии внутрипотоковой сборки мусора и оценка их эффективности. - URL: http://www.iis.nsk.su/files/preprints/179.pdf
31. Фуксман А.П. Технические аспекты создания программных систем. – М.: Статистика, 1979. – 180 с.
32. Baar T. A DSL and a SPIN-frontend for river-crossing problems defined with Xtext. Bulletin of Novosibirsk Computing Center, Computer Science subseries. 2015, n.38, p.29-36.
33. Baar T. Verification Support for a State-Transition-DSL Defined with Xtext. Perspectives of System Informatics - 10th International Andrei Ershov Informatics Conference, PSI 2015, in Memory of Helmut Veith, Kazan and Innopolis, Russia, August 24-27, 2015, Revised Selected Papers. Lecture Notes in Computer Science 9609, Springer 2016, p.50-60.
34. Danial Berezin, Neil D. Jones Compiling ULC to Lower-level Code by Game Semantics and Partial Evaluations. - Fifth International Valentin Turchin Workshop on Matacomputation. p. 11-23
35. Dimitur Krustev. A Supercompiler Assisting Its Own Formal Verification // Russia, Pereslavl-Zalessky: Publishing House «University of Pereslavl» - Fifth International Valentin Turchin Workshop on Matacomputation. Juny 27- Jily 1, 2016, p. 105-125
36. Jens Knoop Compiler Construction. 20th International Conference, CC 2011. Held as Part of the Joint European Conferences on Theory and |Practice of Software, Lecture Notes in Computer Sciences, 6601. ETAPS 2011 Saarbrcken, Germany, March 26 – April 3, 2011. Springer. 330 p.
37. Lucas P., Lauer P., Stigleitner H. Method and Notation for the Formal Definition of Programming Languges. IBM Laboratory – Venna, TR 25.087, 1968.
38. McCarthy J. LISP 1.5 Programming Mannual. – The MIT Press., Cambridge, 1963. – 106p.
39. Marjan Mernik Formal and Practical Aspects of Domain-Specific Languages. — IGI Global, 2012. — ISBN 978-1-4666-2092-6.
40. Сайт энтузиастов разработки новых языков программирования- URL: http://compiler.su

41. Материалы по созданию DSL - URL: https://eclipse.org/Xtext/documentation/301_grammarlanguage.html
42. Markus Voelter, DSL Engineering: Designing, Implementing and Using Domain-Specific Languages, 2013 - URL: http://voelter.de/dslbook/markusvoelter-dslengineering-1.0.pdf, http://dslbook.org/
43. Сайт с материалами по особо эффективной реализации Lisp-а – CMUCL. - URL: http://www.cons.org/cmucl/ -
44. Walid Taha Domain-Specific Languages. Houston. 2009. - URL:
45. Зуев Е. История разработки компилятора Си+ по заказу иностранной фирмы в ранне постсоветское время. - URL: http://www.gramotey.com/?open_file=1269097005
46. Статья Руслана Хайрова про особенности LLVM - URL: https://habrahabr.ru/post/47878
47. Разработка компиляторов: Дмитрий Булычев, Наталья Вояковская, Антон Москаль, Андрей Терехов - URL: http://www.intuit.ru/studies/courses/26/26/info
48. Городняя Л.В. Парадигмы программирования. – М.: Интернет-Университет Информационных технологий. − 2006 - URL: http://www.intuit.ru/studies/courses/
49. Теория и реализация языков программирования. Галочкин М., Гончар Д., Серебряков В., Фурутян М.- URL: http://www.intuit.ru/studies/courses/1157/173/info
50. JetBrains Metaprogramming System MPS - URL: https://www.jetbrains.com/mps/
51. Сайт с материалами по основам CORBA. - URL: http://www.optim.ru/cs/1998/4/corba/corba.asp
52. Сайт с материалами по инструментарию конструирования DSL/- URL: http://xsemantics.sourceforge.net/xsemantics-documentation/XsemanticsSyntax.html