УДК 004.43
DOI 10.25205/1818-7900-2021-19-2-29-52

Визуализация результатов анализа языков программирования для их поверхностного сравнения

Л. В. Городняя

Новосибирский государственный университет
Новосибирск, Россия

Институт систем информатики им. А. П. Ершова СО РАН
Новосибирск, Россия

Аннотация

Статья посвящена выбору наглядной и лаконичной формы для представления результатов анаиза и сравнения языков программирования, удобной для оценки выразительной силы языков и трудоемкости реализации систем программирования. Формализация приспособлена к парадигмальному анализу определений языков программирования и выбору практичных критериев декомпозиции программ. В качестве основного подхода выбрана семантическая декомпозиция определений языков в рамках анализа парадигм программирования. Такой выбор позволяет выделять автономно развиваемые типовые компоненты программ, которые могут быть приспособлены к конструированию различных информационных систем. Многие работы по методам разработки программных систем зависят от практичности подходов к декомпозиции программ, отлаживаемых с помощью систем программирования. Решение этой проблемы полезно при изучении методов программирования, исследовании истории языков программирования, для сравнения парадигм программирования, потенциала используемых схем и моделей, оценки уровня новизны создаваемых языков программирования, а также при выборе критериев декомпозиции программ. Кроме того, определенность критериев позволяет формировать методику обучения разработке компонентов информационных систем. Попутно показана дистанция в понятийной сложности между программированием и разработкой систем программирования.

Ключевые слова

определение языков программирования, парадигмы программирования, декомпозиция программ, критерии декомпозиции, семантические системы

Благодарности

Работа выполнена при поддержке РФФИ, проект № 18-07-01048-а

Для цитирования

Городняя Л. В. Визуализация результатов анализа языков программирования для их поверхностного сравнения // Вестник НГУ. Серия: Информационные технологии. 2021. Т. 19, № 2. С. 29–52. DOI 10.25205/1818-7900-2021-19-2-29-52

Visualization of the Results of the Analysis of Programming Languages for Their Superficial Comparison

L. V. Gorodnyaya

Novosibirsk State University
Novosibirsk, Russian Federation

A. P. Ershov Institute of Informatics Systems SB RAS
Novosibirsk, Russian Federation

Abstract

The article is devoted to the choice of a clear and concise form for presenting the results of analise and comparing programming languages and systems, convenient for assessing the expressive power of languages and the complexity of implementing systems. The formalization is adapted to the paradigm analysis of the definitions of programming languages and the selection of practical criteria for decomposition of programs. The semantic decomposition of the definitions of languages and systems as part of the analysis of programming paradigms was chosen as the main ap-

© Л. В. Городняя, 2021

proach. Such a choice makes it possible to single out autonomously developed typical program components that can be adapted to the design of various information systems. Many works on methods for developing software systems depend on the practicality of approaches to decomposition of programs debugged using programming systems. The solution to this problem is useful when studying programming methods, studying the history of programming languages, for comparing programming paradigms, the potential of used circuits and models, assessing the novelty level of created programming languages, and also when choosing criteria for program decomposition. In addition, their existence allows us to form a teaching methodology for developing the components of information systems. Along the way, the distance in conceptual complexity between programming and programming system development is shown.

Keywords

definition of programming languages, programming paradigms, program decomposition, decomposition criteria, semantic systems

Acknowledgements

The study was funded by RFBR according to the research project no. 18-07-01048-a

For citation

xxx. *Vestnik NSU. Series: Information Technologies*, 2021, vol. 19, no. 2, p. 29–52. (in Russ.) DOI 10.25205/1818-7900-2021-19-2-29-52

Введение

В данной статье предлагается ряд решений для визуального представления и формализованного измерения понятийной сложности конструкций, поддержанных в определениях языков и систем программирования (ЯиСП). Такое представление может быть применено при сравнении парадигм программирования (ПП), потенциала схем и моделей, используемых при разработке программ, оценки уровня новизны создаваемых языков программирования (ЯП), а также при выборе критериев декомпозиции программ. Долгоживущие программы, такие как системы программирования (СП), требуют объективных критериев декомпозиции программ, отражающих возможность автономного развития выделенных компонентов СП с целью профилактики повторного программирования при развитии реализуемого ЯП.

1. Математические абстракции

В качестве отправной точки для представления решений по декомпозиции программ можно принять классическое понятие «алгебраическая система» [1]: алгебраическая система – это <A, F>, где A – основное множество значений, F – конечный набор операций над множеством A.

Примерно так можно формализовать библиотечные модули в СП и классы объектов в ООП до тех пор, пока при реализации ЯП не возникает необходимость использовать разные модели вычислений по одним и тем же формулам. С. С. Лавров предложил понятие «семантическая система», расширяющее понятие «алгебраическая система» заданием явного правила применения функций к значениям [2]: семантическая система – это <**V**, **F**, **R**>, где **V** – основное множество значений, **F** – конечный набор функций, возможно принадлежащих основному множеству, **R** – правило применения функций к значениям, возможно входящее в набор функций.

Близкое понятие под названием «монады» введено в язык Haskell.

Большинство ЯП поддерживает разные варианты утончения понятия «семантическая система», например, где **F** – конечный набор функций, **не принадлежащих** основному множеству, **R** – правило применения функций к значениям, **не входящее** в набор функций.

Соответствующее обозначение такого вида семантических систем можно представить как <**V; F; R**>. Вид <V; F; R> характерен для ЯП, обладающих четким разделением данных и программ, а также для языков управления базами данных.

2. Архитектурные модели

Следует отметить, что жаргон современного практического программирования понятие «язык программирования» использует как «входной язык типовой системы программирования». Разница заключается в том, что СП обычно сопровождает реализацию ЯП расширяемым комплектом библиотечных модулей. В результате происходит сглаживание видимых различий между языками программирования и системами программирования. Кроме того, именно доступная СП является истиным документом по языку программирования, в отличие от формальных описания и станддартов, слабо представляющих реализационную прагматику ЯП и часто по объему превышающих разумные размеры. Это приводит к целесобразности при анализе особенностей ЯП и структурировании поддержанных в нем средств учитывать архитектурные модели.

Успехи эелементной базы не исключают актуальность создания эффективных программ. Поэтому выбор параметров декомпозиции программ должен учитывать особенности архитектурных моделей. Наиболее известные компьютерные архитектуры обычно представляют как конструкцию из вычислительного устройства (E), памяти (M), устройства управления процессами (C) и средств коммуникации между устройствами и их элементами (S). Такая конструкция допускает детализациею на уровне более тонких аппаратных решений и расширение на уровне периферийных устройств, что дает первое приближение для выбора основных видов семантических систем в языках программирования [3]. Рассматривая любые семантические системы, важно отметить формальную разницу в характере выполнения функций таких систем. Так, для любого множества значений V реализационно различимы виды функций для методов вычислений FE: (V* → V+), средств доступа к памяти FM: (T : N → V), особенностей управления вычислениями FC: (F → {0, 1})* и обратимой комплексации и структурирования данных FS: (A → K) U (A ← K) = (A ↔ K).

Следует обратить внимание, что такие виды семантических систем часто обладают кумулятивным эффектом в порядке «VEMCS». Если V – произвольное множество значений, то FE: (V* → V+) – обычные вычисления, заданные формулами над значениями из этого множества, а формула может представлять самоопределимое константное значение из V. Для реализации средств доступа к памяти FM: (T : N → V) характерно выделение понятия «адрес» (N) и подразумевается существование специальной таблицы T, по которой определено соответствие адресов заданным значениям, при задании которых могут использоваться формулы вычислений. Особенности управления вычислениями FC: (F → {0,1})* состоят в разметке запрограммированных действий для выделения выполняемых, обычно используя понятие адресуемой памяти. Обратимая комплексация или структурирование данных в современных архитектурах FS: (A ↔ K) требует определения границы между атомарными (A) и сложными, конструируемыми (K) объектами с возможностью как наращивания сложности, так и урощения любых построений с учетом разных условий.

Это приводит к представлению о кумулятивной шкале кактегорий семантических систем на основе классификации видов функций (табл. 1).

Таблица 1

Классы семантических систем

Table 1

Classes of semantic systems

№ столбца	V	E	M	C	S
Виды функций (F V)	**V**	**FE: V* → V+**	**FM: (T : N → V)** V = N U V	**FC: (F → {0, 1})*** [1] V = {0,1} U V	**FS: (A → K) U (A ← K)** V = A U K

[1] Обозначение уточнено А. В. Климовым.

3. Понятийная матрица

Одним и тем же наборам функций могут соответствовать разные правила **R**, определяющие методы вычислений, влияющие на результативность выполнения программ.

RE – обычные арифметические вычисления, отображающие произвольный ряд **значений** аргументов в не менее чем один результат; механизм вычисления можно не задавать, считать, что происходит чудо выдачи результата средствами аппаратуры или подобно им.

RM – символьные вычисления, подставляющие **представления** аргументов без их предварительного вычисления; механизм сводится к разным схемам копирования данных, обычно доступным в любой архитектуре.

RC – частичные или смешанные вычисления, лавирующие между вычислением и подстановкой в зависимости от разных условий; обычно в числе таких условий имеются прерывания, имеющие аппаратную поддержку.

RS – обобщенные и / или параллельные вычисления, оперирующие организацией процессов как множеством потоков над комплексами из разных устройств или данных; обычно поддерживаются особо эффективные механизмы доступа к соседям.

Представление таких различий можно выразить, дополнив горизонтальную шкалу видов функций вертикальной шкалой моделей вычислений или методов применения функций к значениям (R), что будет выглядеть как матрица семантических систем – понятийная матрица. Можно вспомнить, что ЯП определяется не только на уровне лексики, синтаксиса и семантики, но еще и на уровне **прагматики**. Исторически прагматика представлена как традиция реализации ЯП в наиболее популярных СП в форме правил, уточнывющих особенности категорий семантических систем, поддерживающих разные парадигмы программирования. Следует отметить, что разные парадигмы могут быть реализованы одной и той же прагматикой. Это делает нетривиальным проблему представления уловия для различия парадигм [4]. Различия категорий семантических систем, поддерживающих различимые парадигмы на уровне правил применения функций к значениям в наиболее известных языках высокого уровня (ЯВУ), выражены табл. 2–5. Строка «RE: ядро» этой матрицы представляет уровень базовой семантики ЯП, существенные различия видов функций семантических систем которого можно представить табл. 2. **Базовая семантика** – это подмножество определения ЯП, из которого можно вывести все остальные его средства методом консервативного расширения, т. е. введения функций над базовой семантикой.

Таблица 2

Семантическая декомпозиция минимального ядра ЯП

Table 2

Semantic decomposition of the minimal PL kernel

№ столбца	V	E	M	C	S
Виды функций	V	FE: V* → V+	FM: (T : A → V)	FC: {F → {0,1}}	FS: A ↔ K
RE: Ядро	Значение	Операции	Память	Управление	Вектор

Ядро – семантический базис. Полное определение ЯП можно получать как консервативное расширение ядра. Обычно ядро приспособлено и к неконсервативному расширению пополнением набора библиотечных функций, реализуемых на уровне аппаратуры. Это позволяет в реализации СП для любого ЯП поддерживать разные парадигмы программирования, необходимые для обеспечения полного жизненного цикла программ, чтобы достигать результата независимо от исходных возможностей ЯП.

Значение – минимальное представление объектов из области приложения языка, обычно это самоопределимые константы.

Операции – минимальный комплект функций для обработки значений.

Память – введение адресов для лаконичного и уникального представления значений (указатели, идентификаторы, переменные, метки).

Управление – разметка выполнимости композиции элементов программы из (операций, функций, действий и т. п.) специально выбранными значениями, например, {0|1} или {True | False}, или {Nil | T}.

Вектор – обратимое конструирование одноуровневых комплектов, рассматриваемых как целостность, из которых можно восстанавливать исходные элементы. На уровне ядра достаточно одной структуры, поддерживающей доступ по принципу соседства, – вектора, списка, очереди или т. п.

Строка «RM: Макро» табл. 3 представляет уровень макрорасширений, что вместе с первой строкой достаточно, чтобы характеризовать основные средства большинства языков низкого уровня.

Таблица 3

Семантическая декомпозиция макрорасширения ядра ЯП

Table 3

Semantic decomposition of the macro-extension of the PL core

№ столбца	V	E	M	C	S
Виды функций	V	FE: V* → V+	FM: (T : A → V)	FC: {F → {0,1}}	FS: A ↔ K
RE: Ядро	Значение	Операции	Память	Управление	Вектор
RM: Макро	Данное	Функции	Задание	Блоки	Стек

Макро – пополнение ядра средствами обработки представлений, используемых с целью укрупнения любых конструкций, что позволяет выполнять консервативное расширение ЯП. Кроме того, оно способствует лаконизму текстов программ. Макротехника позволяет наследовать отлаженность фрагментов программ. Бывает важным исключать дубли частей текста, кода и структур данных. Простейший механизм макрогенерации обычно присутствует во многих СП как препроцессор. Так же бывает устроена техника кодогенерации и обработки шаблонов при компиляции программ, нередко проникающая и на уровень ЯП. Реализация укрупнений может быть функционально эквивалентна вызову подпрограмм. Взаимозаменяемость макроподстановки и вызова подпрограмм нередко используется при оптимизации программ.

Данное – хранимое значение или выражение, допускающее уникальность экземпляра, доступного многократно по адресу, возможно, на внешнем устройстве.

Функции – укрупнение операций с возможной параметризацией операндов. Реализационная прагматика может отличаться техникой передачи параметров через стек или специальное поле аргументов или неявно. Последнее позволяет и работу с памятью формально рассматривать как функцию с неявным аргументом, выполняющим роль функции T, задающей соответствия адресов и значений.

Задание – хранимое именованное выражение с возможностью многократного выполнения.

Блоки – хранимое выражение или код программы, представляющий составные действия, ветвления, циклы, вызовы функций, обычно с локализацией переменных, приводящей к понятию «иерархия».

Стек – схема организации данных, дополняющей соседство определенной дисциплиной доступа для поддержки иерархии вычислений, возможно, с защитой независимых блоков.

Повышение уровня ЯП обеспечивается не только особым вниманием к средствам укрупнения данных на базе понятия «иерархия» и оперирование блоками программы. Дальнейшее наращивание объемов разрабатываемых программ отчасти достигается автоматизацией контроля некоторых условий корректности применения операций и функций к их операндам в определенных границах. Становятся важными понятия «предикат» и «тип переменных», удобно проверяемые при компиляции, что представлено строкой «RC: Границы» табл. 4. Разновидностью условий являются сигналы о прерываниях или вычисляемых неподходящих ситуациях.

Таблица 4

Семантическая декомпозиция диагностического дополнения ядра ЯП

Table 4

Semantic decomposition of the diagnostic complement of the PL kernel

№ столбца	V	E	M	C	S
Виды функций	V	FE: V* → V+	FM: (T : A → V)	FC: {F → {0,1}}	FS: A ↔ K
RE: Ядро	Значение	Операции	Память	Управление	Вектор
RM: Макро	Данное	Функции	Задание	Блоки	Стек
RC: Границы	Исключения	Предикаты	Типы переменных	Логика	Варианты

Границы – методы проверки вычислимости функций и выполнимости заданий. Цель представления границ – снижение трудоемкости отладки программ упрощением поиска ошибок. При отсутствии ошибок проверка воспринимается как накладные расходы. Встречаются механизмы установки ловушек на непредусмотренные ситуации и программирования обработки исключений с возможностью продолжения вычислений.

Исключения – выбор специальных значений для разметки неожиданных ситуаций. В некоторых ЯП вводят значения типа Error. При переходе к СП происходит добавление текстовых шаблонов для формирования диагностических сообщений об исключительных ситуациях.

Предикаты – специальные функции, позволяющие определять типы значений или сравнивать значения независимо от расположения данных в памяти. Роль предиката может выполнять любая функция при подходящих договоренностях и схеме реагирования на ее результаты.

Типы переменных – связывание типа значения с переменной, хранящей нетипизированный код значения в памяти.

Логика – проверка соответствия типа данных или значений операциям обработки значений или доступа к памяти, возможно, с учетом условий вычислимости.

Варианты – схема организации данных без определенной дисциплины доступа для организации перебора или выбора равноправных элементов или блоков. Полезно при отладке программ как механизм удостоверения принципиальной выполнимости вычислений при частичной постановке задачи.

По мере расширения границ программа может стать достаточно универсальной, способной к разумному поведению на любых входных данных, что выражено строкой «RS: Общность» в табл. 5, в которой добавлена слева колонка с обозначениями строк полученной понятийной матрицы.

Таблица 5

Понятийная матрица:
семантическая декомпозиция практичного обобщения ядра ЯП

Table 5

Concept matrix:
Semantic decomposition of a practical generalization of the PL core

№ строки	№ столбца	V	E	M	C	S
	Виды функций	V	FE: V* → V+	FM: (T : A → V)	FC: {F → {0,1}}	FS: A ↔ K
E	RE: Ядро	Значение	Операции	Память	Управление	Вектор
M	RM: Макро	Данное	Функции	Задание	Блоки	Стек
C	RC: Границы	Исключе-ния	Предикаты	Типы перемен-ных	Логика	Варианты
S	RS: Общность	Неопреде-ленность	Мульти-операции	Внешний мир	Отображения	Ввод-вывод

Общность – дополнительные средства обеспечения отладки и применения программ, поддерживающие возможность разумного продолжения вычислений при любых исходных данных и аварийных ситуациях.

Неопределенность – вводятся специальные дополнительные значения и ловушки (_|_, Error, Future). Такое расширение множества значений позволяет учитывать в текстах программ некоторые отдельные особенности процесса разработки, отладки и схемы жизненного цикла программ.

Мультиоперации – допускается произвольное число операндов операций, аргументов и результатов функций. Возможно просачивание определений на однородные структуры данных, позволяющее определения над элементарными данными автоматически распространять на более сложные данные.

Внешний мир – механизм неявного расширения области действия операций и функций на периферийные устройства, рассматриваемые как обобщение памяти.

Отображение – возможность регулярного применения функции к серии данных благодаря использованию представлений функций или указателей в качестве аргументов функций более высокого порядка.

Ввод-вывод – средства приема данных с внешних устройств и размещения данных на внешних носителях данных, включая средства доступа к устройствам с уровня программы. Стандартно имеется в виду прием данных с клавиатуры и изображение данных на экране. Обычно подразумевается аксиоматика, требующая совместимости форматов ввода-вывода: для всякого вводимого данного существует эквивалентное ему выводимое данное и, обратно, если данное может быть выведено, то его можно ввести без потерь.

Таким образом, разным видам функций относительно схемы применения функций к аргументам на уровне ЯП при реализации СП соответствуют определенные позиции специальной понятийной матрицы, полученной как двумерная кумулятивная шкала. Кумулятивный эффект по второму измерению получается совмещением результатов ячейки [E,S] с [M,V], [M,S] с [C,V], [C,S] с [S,V]. Более подробные пояснения даны в табл. 6. Понятийную матрицу можно равссматривать как форму онтологического определения ЯП.

Таблица 6

Кумулятивные эффекты в понятийной матрице при переходе на очередную позицию (наследование средств предшествующих позиций)

Table 6

Cumulative effects in the concept matrix when moving to the next position (inheritance of the means of previous positions)

Индекс позиции	Пояснение
EV	Самоопределяемые скаляры, их смысл не требует интерпретации, текстовое представление понятно без комментариев
EE	Операции над скалярами, они должны вырабатывать скаляры. При выходе за границы V могут его пополнять или вырабатывать сигнал неуспеха. Множество значений может быть пополнено представлениями формул
EM	Обработка памяти над таблицей адресов с соответствующими им значениями. Адреса и таблица могут быть значениями или не рассматриваться как значения и использоваться неявно как сущности другой природы. Появляется именование значений и формул, что расширяет класс допустимых формул, обеспечивает многократное использование хранимых результатов и приводит к понятию «данное»
EC	Безусловное и условное (без «else») управление процессом вычислений, приоритеты в формулах вычисления и переход к очередному действию или по метке
ES	Конструирование последовательностей по принципу соседства и перебора слева направо дополняются возможностью возвратов или выбора любого элемента ради обратимости. Как правило, это векторы или строки
MV	Хранимые данные могут иметь имена, что позволяет решать проблемы укрупнения используемых единиц, воспринимаемых равноправно со скалярами и операциями. Возникают имена логических значений для представления условий выбора хода процесса
ME	Композиции операций рассматриваются как безымянные функции, равноправные базовым операциям, они могут обрабатывать и вырабатывать не только скаляры, но любые определенные данные, используя упаковку ряда значений в последовательность
MM	Именование функций упрощает их многократное использование в формулах, включая представление рекурсии
MC	Композиции операций и функций можно рассматривать как одно действие, что приводит к понятию «блок», выделенный скобками, влияющими на порядок вычислений и задающими области видимости имен, – иерархия. Появляются системные процедуры, что может приводить к неконсервативным расширениям
MS	Структуры однородных данных и процессов сопровождаются дисциплиной доступа к элементам – FIFO, FILO, взаимоисключение, одновременность или др.
CV	Множество значений пополняется специальными представлениями сигналов «успех-провал» процесса независимо от существования логических значений, а также текстами диагностических сообщений
CE	Появляются понятие «предикат» и специальная функция ERROR для выбора обработчиков диагностических ситуаций и продолжения недоопределенных вычислений
CM	Появляется типизация значений и данных, а также сигнатур операций и функций, используемая для профилактики неудачных вычислений
CC	Вводятся специальные схемы управления вычислениями на основе проверки условий соответствия данных и действий для их обработки

Окончание табл. 6

Индекс позиции	Пояснение
CS	Появляются структуры с равноправным доступом к разнородным элементам, возможно, с их разметкой и указанием кратности вхождения
SV	Вводятся мультизначения и специальные значения для представления разного рода неопределенностей, раскрытие или игнорирование которых может быть полезно для продолжения вычислений
SE	Появляются операции над произвольным числом параметров, их распространение на любые структуры данных, проекции формул, отложенные вычисления, формулы доопределения и продолжения вычислений
SM	Понятие присваивания распространяется на обмен данными с переферийными устройствами. Возникает идентификация устройств, абстрактные и конкретные имена, паспорта взаимодействий, сигналы готовности действий, время ожидания отклика от устройства, копии и многое другое, отражающее специфику разного оборудования
SC	Возникают отображения, средства ввода-вывода, сетевое управление, потоки, эстафета обслуживания, итерирование, условия срабатывания, актив-пассив, администрирование, сервер, ОС
SS	Одновременность, синхронизация, взаимоисключения, сигналы и сообщения, пакеты, настройки, сервисы, конфигурации

Примечание: индекс позиции (клетки в понятийной матрице) имеет формат: XY, где X – обозначает строку понятийной матрицы, а Y – столбец, оба имеют значения из E, M, C, S.

4. Семантические системы основных парадигм (паспорт парадигмы)

Рассматривая любые семантические системы, важно отметить разницу в характере выполнения функций таких систем в различных контекстах. Так, для любого множества данных D, представляющих значения V произвольной природы, реализационно различимы схемы функций для методов вычислений E, средств доступа к памяти M через адреса N, особенностей управления вычислениями C и коммуникации или обратимой комплексации и структурирования данных S. Это приводит к представлению об основных категориях семантических систем различно реализуемых схем функций F. Исторически на уровне аппаратуры такие категории систем обладали кумулятивным эффектом в порядке «DEMCS» (см. табл. 5) – представление чисел, арифмометр, калькулятор, дифференциальный вычислитель, компьютер, причем аппаратные подсистемы могут взаимодействовать каждая с каждой. Детализация определений ЯП по семантическим и прагматическим особенностям позволяет достаточно четко различать наиболее известные парадигмы программирования, поддержанные во многих языках [5].

При подготовке программ парадигмальные различия проявляются в отношении порядка между категориями семантических систем. Переход к отладке программ влечет прагматическую разницу в упорядочении слоев, не влияющую на выбор парадигмы: или раскрутка программы от ядра через расширение и уточнение к универсальной программной системе, или сборка из готовых компонентов через настройку и адаптацию к требованиям целевого интегрированного приложения. И то, и другое может быть осуществлено на базе любого ЯиСП.

Таблица 7

Ряд категорий семантических систем аппаратного и реализационного уровня

Table 7

A number of categories of semantic systems at the hardware and implementation level

Подсистема	Примечание
Категории семантических систем	**Аппаратно ориентированная семантика вычислений**
D: данные	Данные из множества D представляют значения из V и шкалу прерываний
E: вычисления	Операции по двум-одному значению производят одно или два значения
M: память	Соответствие между адресами из множества N и хранимыми по этим адресам представлениями из множества D для значений из V допускает разные методы доступа к элементам памяти, включая замену хранимых значений, за исключением адреса 0
C: управление	Сравнение значений с нулем позволяет управлять ходом вычислений наряду с передачами по меткам и обработкой прерываний, не считая перехода по порядку
S: коммуникации	Конструирование сложных данных учитывает возможности команд адресации в памяти и использование внешней памяти
Слои реализационных решений	**Прагматика реализации уточняет методы выислений**
E: ядро	Вычисления над значениями, имеющими представление в виде данных, обрабатываемых непосредственно машинными командами, устройство которых не доступно с уровня языка программирования
M: укрупнение	Данные, результаты и команды их обратотки размещаются в памяти, что позволяет использовать их многократно и к сложным данным, включая процедуры, обращаться по адресу, подобно обращению к команде
C: граница	События, происходящие в процессе вычисслений или во внешней обстановке, могут порождать сигналы или распознаваться по специальным условиям, которые можно учитывать в программе
S: универсальность	Обстановка, в которой выполняются вычисления, может формировать любые входные данные, включая непредусмотренные ранее или неподходящие. Поэтому возникает необходимость расширение области определения программы дополнительными значениями, позволяющими различать подходящие данные и использовать разнообразные устройства

5. Пример представления результатов анализа и сравнения ЯП

Результаты анализа языка Pure Lisp представлены в табл. 8, а языка Pascal – в табл. 9. В клетках таблиц расположены фрагменты текстов программ на анализируемых языках или обозначения фрагментов или понятий языка.

Понятийная матрица позволяет средства языка Pure Lisp характеризовать в рамках семантической системы вида <V, F, R>, отображаемой в абстрактную машину SECD, обладающую независимыми регистрами [6]. Функции такой системы могут принадлежать основному множеству, и правило применения функций может входить в набор функций.

Таблица 8

Понятийная матрица определения языка Pure Lisp

Table 8

The concept matrix of the definition of the Pure Lisp language

№ строки	№ столбца	V	E	M	C	S
	Виды функций	V	F: V* → V	AL : Atom → V	(F → {NIL, T})*	A ↔ K
E	Ядро	a) NIL ; и атомы	a) eq: V V→{NIL,T}	a) AL =(… (NIL . NIL) (T . T))	a) eval:Sexpr AL→ V b) quote	a) cons: V1 V2 → (V1 . V2) b) car: (V1 . V2) →V1 c) cdr: (V1 . V2) →V2
M	Специальные функции	a) Sexpr: (V . V)	a) (lambda (x1 x2 ...) form) b) (label FN F)	a) pairlis X Y→ AL b) assoc: X AL→V	a) ((lambda (x1 x2 ...) form) v1 v2 …) b) ((label FN F) …)	a) (V . . .) ; list ; Список b) (F v1 v2 . . .) ; Форма
C	Границы	a) NIL b) «Нет переменной» c) «Нет функции»	a) atom:V→{NIL,T} b) apply: F V* AL→V c) ERROR	a) null b) Свободные переменные c) Типы значений	a) cond b) FUNCTION	a) Строки b) Clozure = Funarg
S	Общность Практичность	a) Числа b) Строки	a) evlis : (form*) → V* b) evcon : ((form form)*) → V **c) + - * /**	a) Псевдо-функции b) свойства	a) map b) reduce c) lazy	a) PRINT b) READ

Таблица 9

Понятийная матрица определения языка Pascal

Table 9

The concept matrix of the definition of the Pascal language

№ строки	№ столбца	V	E	M	C	S
	Виды функций	V	F: V* → V	M : Id→ V+F	(F → {True, False})*	V N ↔ Array [N] of V

E	Ядро	a) type = (a,b, . . .)	a) = <>
			b) pred succ
				a) Id
				b) var X : int;
				c) X := 1;
				d) label L;
				e) L: a := 1;
					a) (1*(2+X))
					b) S1 ; S2
					c) ключевые слова
					d) goto L;
						a) s: array [1 . . 9] of int;
						b) s [5] := x;
						c) x := s [5]

M	Расширение	a) false true
		b) maxint
		c) integer
		d) (1,3,6,9, . . .)
		e) const A=1;B=2;
		f) type
		g) ta = 1 .. 10;
			a) + - * / div mod
			b) F (1, x, ...)
				a) var va : ta;
				b) begin .
				c) begin . . .
				end. . end
				d) func F subr S
				e) end.
					a)if X then S1 else S2
					b) begin S1 ; S2... end
					c) S (X, Y)
					d) forward;
						a) s: array [1 . . 9]
						of array [1 . . 3] of int;
						b) s [5,2] := x;
						c) x := s [5,2]

C	Границы	a) «Нет определения» b) “не соотв. ТД :=” “не соотв. ТД операнда” “не соотв. ТД параметра” c) “выход из диапазона”	a) = < > >= <= b) F: V → {True, False} c) in d) not or and	Типы данных: a) int b) char c) array ...	a) case b) Соответствие ТД	a) record b) set c) record … case
S	Практичность	Указатель: a) Nil b) X^	a) Функция-параметр	a) Процедура-параметр, 1) С*тандартные процедуры* *2) NEW* *3) DESPOSE, ...*	a) while for until	*1) Стандартные библиотеки* *2) WRITE* *3) READ*

Понятийная матрица, характеризующая средства языка Pascal, обладает меньшей кумулятивностью. Pascal определен в рамках семантической системы вида <V; F; R>, отображаемой в пи-код [7], в котором, в отличие от SECD, регистры SEC размещены в общей памяти. Составляющие такой системы строго разделены. Множество V фиксировано по составу, неявное определение R не допускает модификаций, функции могут выполнять роль данных в терминах указателей, связь с внешним миром определяется вне языка через библиотечные модули.

Такие понятийные матрицы достаточны для представления результатов сравнительного анализа ЯП и оценки сложности их применения, например, по числу различимых конструкций в каждой клетке понятийной матрицы. Тем не менее, сравнение таких матриц требует определенных усилий. Более простое, хотя и грубое, сравнение можно выразить как в табл. 10–12.

Таблица 10

Представление результатов сравнения понятийной сложности определений языков Pure Lisp и Pascal (число пунктов в клетках табл. 7, 8)

Table 10

Presentation of the results of comparing the conceptual complexity of the definitions of the languages Perl Lisp and Pascal (the number of points in the cells of Tables 7, 8)

Lisp						Pascal					
	V	E	M	C	S		V	E	M	C	S
E	1	1	1	2	3	E	1	2	5	4	3
M	2	2	2	2	2	M	7	2	5	4	3
C	3+	3	3	2	2	C	3+	4	3	2	3
S	2	3	2	3	2	S	2	1	1/**3**	1	0/**3**

Трудоемкость сравнения таких матриц много меньше, сразу видно соотношение числа семантических сестем в разных категориях. Варианты кратких формы представления тех же самых результатов сравнения сложности ЯП (число пунктов в клетках и строках табл. 8, 9) показаны в табл. 11.

Таблица 11

Краткие формы представления результатов сравнения сложности ЯП

Table 11

Brief forms of presentation of the results of comparing the complexity of the PL

Lisp – 43+	Pascal – 56+ / 6*
(E (1 1 1 2 3) M (2 2 2 2 2) C (3+ 3 3 2 2) S (2 3 2 3 2))	(E (1 2 5 4 3) M (7 2 5 4 3) C (3+ 4 3 2 3) S (2 1 **1+3*** 1 **0+3***))
((E 8) (M 10) (C 13+) (S 12))	((E 15) (M 21) (C 15+) (S 5 /**6***))

Если то же самое представить визуальной диаграммой в стиле деловой графики, то происходит значчительное снижение трудоемкости грубого сравнения характеристик (см. табл. 12), мгновенно с одного взгляда видны различия в потенциале категорий семантических систем. На диаграмме цвет символизирует категорию семантической системы, высота столбика – число семантических систем, входящих в данную категорию. Горизонтальный прямоуголь-

ник символизирует прагматический слой реализационных решений, а размещенные сверху столбики – необходимое библиотечное сопровождение языка, обычно вызванное отсутсвием средств отладки программ. Каждой позиции визуальной диаграммы соответствует клетка понятийной матрицы.

Таблица 12

Визуальная форма представления результатов сравнения сложности ЯП

Table 12

Visual representation of the results of comparing the complexity of the PL

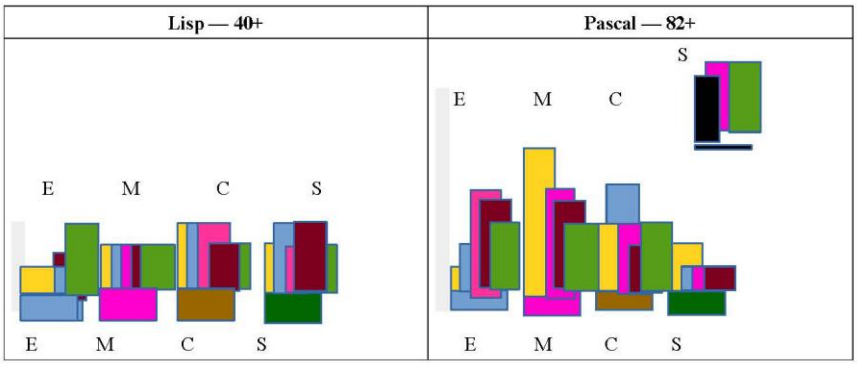


Таблица 13

Понятийная матрица определения языка Pure Lisp

Table 13

Pure Lisp Language Definition Concept Matrix

№ стр		V	E	M	C	S
		V	F: V* → V	AL : Atom → V	(F → {NIL, T})*	A ↔ K
E	b a	a) NIL ; и атомы	a) eq: V V→{NIL,T}	a) AL =(… (NIL . NIL) (T . T))	a) eval:Sexpr AL→ V b) quote	a) cons: V1 V2 → (V1 . V2) b) car: (V1 . V2) →V1 c) cdr: (V1 . V2) →V2
M	b a	a) Sexpr: (V . V)	a) (lambda (x1 x2 ...) form) b) (label FN F)	a) pairlis X Y→ AL b) assoc: X AL→V	a) ((lambda (x1 x2 ...) form) v1 v2 …) b) ((label FN F) …)	a) (V . . .) ; list ; Список b) (F v1 v2 . . .) ; Форма
C		a) NIL b) «Нет переменной» c) «Нет функции»	a) atom:V→{NIL,T} b) apply: F V* AL→V c) ERROR	a) null b) Свободные переменные c) Типы значений	a) cond b) FUNCTION	a) Строки b) Clozure = Funarg
S		a) Числа b) Строки	a) evlis : (form*) → V* b) evcon : ((form form)*) → V c) + - * /	a) Псевдо-функции b) свойства	a) map b) reduce c) lazy	a) PRINT b) READ

Появляется возможность с каждой понятийной матрицей сопоставить ее визуальную диаграмму, позволяющую мгновенно видеть и оценвать разнообразие семантических систем в языках и мощность отдельных категорий семантических систем по числу входящих в них элементов (табл. 13, 14).

Таблица 14

Понятийная матрица определения языка Pascal

Table 14

The concept matrix of the definition of the Pascal language

№ стр		V	E	M	C	S
		V	F: V* → V	M : Id→ V+F	(F → {True, False})*	V N ↔ Array [N] of V

E	d
	c
	b
	a
		a) type = (a,b, . . .)
		b) pred succ
			a) = <>
			b) pred succ
				a) Id
				b) var X : int;
				c) X := 1;
				d) label L;
				e) L: a := 1;
					a) (1*(2+X))
					b) S1 ; S2
					c) ключевые слова
					d) goto L;
						a) s: array [1 . . 9] of int;
						b) s [5] := x;
						c) x := s [5]

M	f
	e
	d
	c
	b
	a
		a) false true
		b) maxint
		c) integer
		d) (1,3,6,9, . . .)
		e) const A=1;B=2;
		f) type
		g) ta = 1 .. 10;
			a) + - * / div mod
			b) F (1, x, ...)
				a) var va : ta;
				b) begin .
				c) begin . . . end. . end
				d) func F subr S
				e) end.
					a)if X then S1 else S2
					b) begin S1 ; S2... end
					c) S (X, Y)
					d) forward;
						a) s: array [1 . . 9] of array [1 . . 3] of int;
						b) s [5,2] := x;
						c) x := s [5,2]

C	d c b a	a) «Нет определения» b) “не соотв. ТД :=” “не соотв. ТД операнда” “не соотв. ТД параметра” c) “выход из диапазона”	a) = < > >= <= b) F: V → {True, False} c) in d) not or and	Типы данных: a) int b) char c) array ...	a) case b) Соответствие ТД	a) record b) set c) record … case
S	3 2 1 b a	Указатель: a) Nil b) X^	a) Функция-параметр	a) Процедура-параметр, 1) *Стандартные процедуры* *2) NEW* *3) DESPOSE,* ...	a) while for until	*1) Стандартные библиотеки* *2) WRITE* *3) READ*

Можно создать гипертекст, в котором от каждой позиции диаграммы возможен переход к соотвествующей клетке понятийной матрицы. Теперь можно понятийную матрицу линеаризовать, и каждому элементу категории семантических систем дать учебные примеры. Число необходимых примеров дает понимание сложности понятий языка (табл. 15).

Таблица 15

Начало ряда учебных примеров программ
(ФП: Pure Lisp – чистые вычисления)

Table 15

The beginning of a series of training program examples
(FP: Pure Lisp-pure computing)

Индекс семантической системы	Понятие	Примеры	Примечания
	Любую информацию можно представить символами. Информационная обработка может быть сведена к выражениям из функций с использованием небольшого числа операций над символами		Исходная гипотеза
EVa	Минимальные значения	NIL	= ( ) = пустой список и «ложь»
		T и другие атомы	Атомы без начального значения
EEa	EQ (равенство)	(EQ NIL NIL)	«истина»
		(EQ T T)	Если T имеет значение, то «истина»
		(EQ NIL T)	«ложь» или «неопределено»
ESa	CONS	(CONS NIL NIL)	Пара (NIL . NIL) = (NIL)
		(CONS x y)	Пара (x . y), **если** x и y имеют значение
ESb	CAR	(CAR (CONS x y))	x – Левый элемент пары (x . y)
ESc	CDR	(CDR (CONS x y))	y – Правый элемент пары (x . y)
EMa	Ассоциативный список	((T . T) (NIL . NIL))	Содержимое памяти, показывающее, что NIL и T означают сами себя, так как связаны сами с собой
ECa	EVAL (Вычисление)	(EVAL NIL) (EVAL T) (EVAL (CONS T NIL)) (EVAL A)	NIL T (T . NIL) = (T) **Если** A вычислимо, то его значение, **иначе** «неопределено»
ECb	QUOTE (Блокировка) ' – эквивалент	(QUOTE A) (QUOTE (EVAL A)) (EVAL (QUOTE A))	A (EVAL A) A
		'A ' (EVAL A) (EVAL ' A)	; эквивалент (QUOTE A) ; эквивалент (QUOTE (EVAL A)) ; эквивалент (EVAL (QUOTE A))

Примечание: индекс семантической системы имеет формат: XYz, где z – номер пункта в понятийной матрице, X – обозначает строку понятийной матрицы, а Y – столбец, X и Y имеют значения из E, M, C, S.

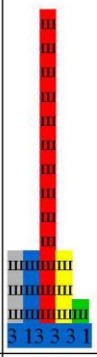
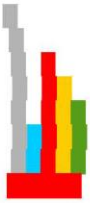
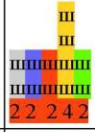
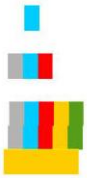
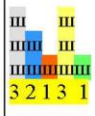
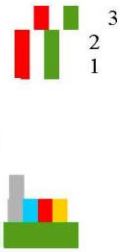
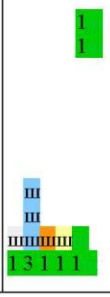
Можно создать гипертекст, в котором от каждой семантической системы из соответствующей клетки понятийной матрицы возможен переход к комплекту учебных примеров, показывающих особенности ее проявления при разработке программ. Диаграммы, характеризующие понятийные матрицы разных языков, можно использовать при их грубом сравнении, чтобы сразу видеть потенциал отдельных категорий семантических систем и быстро отбраковывать заведомо неподходящие. Известно, что в информационном поиске эффективность определяется скоростью отбраковки неподходящего (табл. 16).

Таблица 16

Сравнение ЯП

Table 16

PL comparison

	Лисп	Паскаль	Си - Имп - 52	Примечание
Ядро				Компактное ядро Лиспа нацелено на минимум используемых абстракций. Несколько более объёмное ядро языка Паскаль отражает внимание к понятности. Си опирается на большое число машинных команд.
Расширение				Паскаль поддерживает два формата расширений – отдельно для выражений и для операторов над разнообразными типами данных
Границы				Разное понимание границ вычислимости в различных каетгориях семантических систем
Контекст				Лисп обладает полным спекром средств для отлаки программ и связи с внешним миром. Паскаль и Си для связи с внешним миром используют библиотечные модули, без которых отладка программ проблематична.

6. Парадигмальная декомпозиция мультипарадигмальных ЯП

Для мультипарадигмальных ЯП семантическому анализу желательно предпослать парадигмальную декомпозицию, потому что поддержка средств для разных парадигм может обладать существенными прагматическими различиями, что делает их неудобными для сравнения. И наоборот, отдельные парадигмы могут быть поддержаны общими прагматическими механизмами. Пример такой декомпозиции приведен в табл. 17–20.

Таблица 17

Понятийная матрица определения языка Си (ядро – скаляр)

Table 17

The concept matrix of the definition of the C language (the kernel is a scalar)

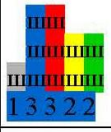
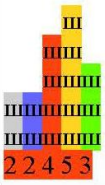
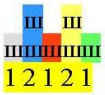
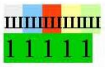
№ стр	№ столбца	V: value	E: eval	M: memory	C: control	S: structure
	Виды функций	V	F: V* → V	AL : Atom → V	(F → {NIL, T})*	A ↔ K
S		•	•	•	•	•
E		• 1 23 Const	• != Не равно == • + - • &v	• int [описатель] [имя]; • X = y • & взятие адреса	• Справа налево • i = j + 2	• [ ] • x [4]
M		• Переменные • скаляр-ные функции	• A + 34 выражения • F (23)	• (x = y) + 1 • x [4] = 5 • i = j + 2; • int F (x) {return x = 2;}	• (y*z)/w изнутри наружу Расставлять скобки, • *Return (x);* • **F (2) + 3** • **While** • **main (argc, aigv, envp)** старт	• стек операндов • таблица переменных • таблица функций
C		• сообщения	• < > <= >= • «выход результата за пределы диапазона используемого типа данных»	• «Нет определения»	• ae ? e1 : e2 или • pe ? e1 : e2	• “библиотека недоступна”
S		• Library	• Include препроцессор ***#include <math.h>*** ***из стандартного каталога***	• Scanf (& Var)	• Printf	• <stdio>

Таблица 18

Понятийная матрица определения языка Си (расширение – процедуры)

Table 18

The conceptual matrix of the definition of the C language (extension – procedures)

№ стр	№ столбца	V: value	E: eval	M: memory	C: control	S: structure
	Виды функций	V	F: V* → V	AL : Atom → V	(F → {NIL, T})*	A ↔ K
E	1 1 1 1 1	• *Список аргумен-тов*	• *type*	• *void*	• *el, e2*	• *(el, e2, . .) список параметров*
M	1 2 1 2 4	• Указа-тель на функцию int (*f)();	• *(F (a, b, ...) + 2) имя_функ-ции (e1, e2,. .., eN);* • *type приведение ТД*	• *void ermesg(s) char * s; { printf("***%s\n", s); }*	• *Return 0;* • *F(выр , ...); G (3); fclose (file);*	• *Стек параметров* • *стек локалов* • *список аргументов* • *struct*
C	1 4 1 2 1	• *не совпада-ют ТД Типы значений*	• *Совме-стимость типов +* • *=* • *Ф (23)*	*Внешние переменные* • *extern long linfunc(); int y; y = linfunc (3.05, 4.0, 1e-3);*	• *typedef старый_тип новый_тип* • *Функция main() выполняется первой.*	• *union bigword { long bg_long; char*bg_char[4]; };*
S	1 1 2 5 1	• *Текст програм-мы*	• ***#include "ABC"***	• *#define идентификатор1 (идентификатор2, ...) строка* *#define abs(A) (((A) > 0) ? (A) : -(A))* • *#undef*	• if ABC + 3 • #ifdef ABC • #ifndef ABC • #else . . . • #endif	• *Описания формальных параметров main*

Таблица 19

Понятийная матрица определения языка Си (контроль – императив)

Table 19

The conceptual matrix of the definition of the C language (control – imperative)

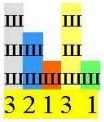
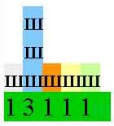
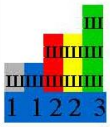
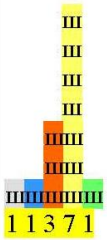
№ стр	№ столбца	V: value	E: eval	M: memory	C: control	S: structure
	Виды функций	V	F: V* → V	AL : Atom → V	(F → {NIL, T})*	A ↔ K
Линейные участки						
E		• float, 4037e-5 double • char *p; char **t; char m[7][50]; Строки "This is a character string" • указатели	• * / • *av += ae* • & • * • iv++ • iv-- • ++pv • --pv • intptr = &n; • *Использование: *pe • Адресные операции • &Использование: &v	[описатель] [имя] ([список]); Инициализация переменных *[описатель] [имя] = [инициализатор]* • *int 1 = 1;* • *int a[] = {1,4,9,16,25,36};* • {x = 1; y = 2; z = 3;}	• ({Последовательность инструкций} • (следующий оператор) • можно задать желаемый приоритет операци	• [N] 12
M		• Метки • Null	• Любое выражение, заканчивающееся точкой с запятой (;), • (указатель _на_функцию) (e1, e2,..., eN)	Глобальные переменные Автоматические переменные • register int y; • static	• Goto ABC2; • ABC2: x = 3; • break, • continue	• Struct • Таблица меток
C		• «Нет метки» • Не существует указанного поля • «значение вместо указателя»	• n = sizeof(arname) / sizeof(int) • *Логические операции*	• переменные, помеченные ключевыми словами	if((условие)) (оператор) • else (альтернативный оператор) • if (i == 0) break; • if (x < 0) printf("negative");	• Union
S		• Адреса из других модулей	• *#define ABC 100* • *#undef ABC* • ***#line 20 "ABC"***	• *Поименованные константы*	• for	*Описания внешних переменных*

Таблица 20

Понятийная матрица определения языка Си (универс – структуры)

Table 20

The conceptual matrix of the definition of the C language (the universe – structure)

№ стр		V: value	E: eval	M: memory	C: control	S: structure
		V	F: V* → V	AL : Atom → V	(F → {NIL, T})*	A ↔ K
E		• Указатель	• *ptr = c;	• описания • {x = 1; y = 2; z = 3;}	• F (g (, , ,)) + 1 • for { for While , do while }	• Typedef • [ ] [ ] [ ] • [ ] { [ ] struct Union}
M		• Null	• *Использование: *fpe • fpe = funcname; (*fpe) (argl,arg2);	• Type F () { G () ;} • *ptr = c; • *Использование: *fpe • Значением выражения является функция, адресуемая указателем fpe. П р и м е р • fpe = funcname; (*fpe) (argl,arg2);	• F (G ( )) ; • While { for While , • do while } • do while { for While , • do while } • if((условие)) (оператор • else (альтернативный оператор) • if (i == 0) break; • if (x < 0) printf("negative") – вложенные ветвления	• Struct { [ ] struct Union} • [ ] . →
C		• «указатель – знач»	• Sizeof	• Typedef • extern • extern int Global_var; extern char *Name; extern int func();	• Switch case • case 3 + 4: Неправильно: case X + Y: • Вариант default не обязательно должен быть последним. • switch (x) { case 'A': printf("CASE A\n"); break; • case 'B': case 'C': printf("CASE B or C\n"); break; • default • вложенные переключатели	• Union { [ ] struct Union} [ ] . ->
S		• *Включаемые файлы*	• коды других модулей	• malloc(), • calloc(), • realloc()	• *extern в начале текущего файла*	• **ФАЙЛЫ**

Теперь можно увидеть достаточно краткую сводку общей картины категорий семантических систем языка (табл. 21).

Таблица 21

Сравнение парадигм Си

Table 21

Comparison of C paradigms

	Скалярное – 39 систем	Процедурное – 34 системы	Императивное – 52 системы	Структурное – 48 систем
Ядро	1 3 3 2 2	1 1 1 1 1	3 13 3 3 1	1 1 2 2 3
Расширение	2 2 4 5 3	1 2 1 2 4	2 2 2 4 2	1 2 5 9 2
Границы	1 2 1 2 1	1 4 1 2 1	3 2 1 3 1	1 1 3 7 1
Контекст	1 1 1 1 1	1 1 2 5 1	1 3 1 1 1	1 1 3 1 1

Визуальные диаграммы в табл. 21 позволяют мгновенно оценить разницу в потенциале парадигм, поддержанных языком Си, подтверждающую общее мнение о его возможностях. Больше всего в языке средств для императивного программирования – это ведущая парадигма. Далее по мощности идет структурное программирование, играющее заметную роль. Скалярное и процедурное программирование имеют поддержку, но рассматриваются как второстепенные парадигмы, меньше влияющие на практическое прешение задач, для решения

которых предназначен Си [8]. Таким образом, получается нечто вроде визитной карточки ЯиСП, позволяющей быстро выполнять их грубое сравнение.

Заключение

Изложенные представления можно рассматривать как конкретизацию понятийной сложности по Колмогорову [9]. Близкие работы начаты еще в середине 1960-х гг., когда возникла задача создания ЯП, поддерживающих процессы разработки СП при создании новых ЯиСП. В конце 1970-х гг. в рамках работ по проекту МАРС [10] был выполнен обзор динамики развития компьютерных конфигураций, что обосновывает выбор основных семантических систем уровня аппаратуры, допускающих эффективную реализацию. В те годы был выполнен ряд серьезных попыток создания языков системного программирования, позволяющих конструировать системы программирования. Наиболее продвинутые из них, такие как Венская методика, не учитывали оценок трудоемкости и понятийной сложности системного программирования, но они составили базу Си-ориентированным LEX-YACC, идеи которых унаследовали Clang-LLVM.

В настоящее время число новых языков программирования стремительно возростает [1], что требует специальной разработки методов лаконичного представления их особенностей для практичной оценки их возможностей. Предложенную визуализацию при оценке сложности и трудоемкости программирования на базе ЯиСП можно дополнить разделением требований к постановкам задач по сферам применения на академические и производственные, а также по уровню изученности на четкие, развиваемые, усложненно трудоемкие и новые.

Список литературы

1. **Болски М. И.** Язык программирования Си. Справочник: Пер. с англ. М.: Радио и связь, 1988. 96 с.: ил. ISBN 5-256-00171-X
2. **Вирт Н.** От Модулы к Оберону // Системная информатика. Новосибирск: Наука, 1991. Вып. 1: Проблемы современого программирования. С. 63–75.
3. **Городняя Л. В.** О представлении результатов анализа языков и систем программирования // Научный сервис в сети Интернет: Тр. XX Всерос. науч. конф. (17–22 сентября 2018 г., Новороссийск). М.: ИПМ им. М. В. Келдыша, 2018.
4. **Городняя Л. В.** Систематизация парадигм программирования по приоритетам принятия решений // Электронные библиотеки. 2020. Т. 23, № 4: Тематический выпуск «Научный сервис в сети Интернет», ч. 2. С. 666–696. URL: https://elbib.ru/article/view/616/711
5. **Колмогоров А. Н**. Три подхода к определению понятия «количество информации» // Проблемы передачи информации. 1965. № 1 (1). С. 3–11.
6. **Котов В. Е.** МАРС: архитектуры и языки для реализации параллелизма // Системная информатика. Новосибирск: Наука, 1991. Вып. 1: Проблемы современого программирования. С. 174–194.
7. **Лавров С. С.** Методы задания семантики языков программирования // Программирование. 1978. № 6. С. 3–10.
8. **Мальцев А. И.** Алгоритмы и рекурсивные функции. М.: Наука, 1965. 392 с.
9. **Хендерсон П.** Функциональное программирование. М.: Мир, 1983. 349 с.
10. **Wegner P.** Concepts and paradigms of object-oriented programming. *SIGPLAN OOPS Mess*., 1990, vol. 1, no. 1, p. 7–87. DOI 10.1145

[1] См. сайт с описаниями 171 языка и 31 парадигмы: http://progopedia.ru/.

References

1. **Bolsky M. I.** The C programming language. Reference book: Trans. from English. Moscow, Radio and Communication, 1988, 96 p.: ill. (in Russ.) ISBN 5-256-00171-X
2. **Wirth N.** From Modula to Oberon. In: System Informatics. Novosibirsk, Nauka, 1991, iss. 1: Problems of modern programming, p. 63–75. (in Russ.)
3. **Gorodnya L. V.** On the presentation of the results of the analysis of programming languages and systems. In: Scientific service on the Internet: proceedings of the XX All-Russian Scientific Conference (September 17–22, 2018, Novorossiysk). Moscow, IPM named after M. V. Keldysh, 2018. (in Russ.)
4. **Gorodnyaya L. V.** Cystematizatsiz paradigms of programming by priorities of decision-making. *Electronic libraries*, 2020, vol. 23, no. 4: Thematic issue “Scientific service on the Internet”, part 2, p. 666–696. (in Russ.) URL: https://elbib.ru/article/view/616/711
5. **Kolmogorov A. N.** Three approaches to the definition of the concept of “quantity of information”. *Problems of information transmission*, 1965, no. 1 (1), p. 3–11. (in Russ.)
6. **Kotov V. E.** MARS: Architectures and languages for implementing concurrency. In: System Informatics. Novosibirsk, Nauka, 1991, iss. 1: Problems of modern programming, p. 174–194. (in Russ.)
7. **Lavrov S. S**. Metody zadachi semantiki yazykov programmirovaniya [Methods of setting the semantics of programming languages]. *Programmirovanie*, 1978, no. 6, p. 3–10. (in Russ.)
8. **Maltsev A. I.** Algorithms and recursive functions. Moscow, Nauka, 1965, 392 p. (in Russ.)
9. **Henderson P.** Functional programming. Moscow, Mir, 1983, 349 p. (in Russ.)
10. **Wegner P.** Concepts and paradigms of object-oriented programming. *SIGPLAN OOPS Mess*., 1990, vol. 1, no. 1, p. 7–87. DOI 10.1145

Материал поступил в редколлегию
Received
21.02.2021

Сведения об авторе

Городняя Лидия Васильевна, кандидат физико-математических наук, доцент ММФ, Новосибирский государственный университет (Новосибирск, Россия), Институт систем информатики им. А. П. Ершова СО РАН (Новосибирск, Россия)

lidvas@gmail.com

Information about the Author

Lydia V. Gorodnyaya, Candidate of Sciences (Physics and Mathematics), Associate Professor of the MMF, Novosibirsk State University (Novosibirsk, Russian Federation), A. P. Ershov Institute of Informatics Systems SB RAS (Novosibirsk, Russian Federation)

lidvas@gmail.com