УДК 004.43 БК 22.18 Г

ФОРМЫ ПРЕДСТАВЛЕНИЯ РЕЗУЛЬТАТОВ ПАРАДИГМАЛЬНОГО АНАЛИЗА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Л. В. Городняя

Институт систем информатики им. А.П. Ершова Сибирского отделения Российской академии наук, пр. Академика Лаврентьева, 6, Новосибирск, 630090; Новосибирский государственный университет, ул. Пирогова, 1, Новосибирск, 630090

lidvas@gmail.com

Аннотация

Цель статьи – выбор представления результатов сравнения языков программирования, удобного для оценки выразительной силы языков и трудоёмкости реализации систем программирования. Формы такого представления должны быть приспособлены к обоснованию практичных критериев декомпозиции программ, что можно рассматривать как подход к решению проблемы факторизации весьма усложнённых определений языков программирования. Актуальность выбора лаконичных и быстро воспринимаемых форм полезен для работы в стремительно развивающемся пространстве новых проблемно-ориентирован-ных языков программирования. Попутно можно показывать дистанцию в понятийной сложности между программированием и разработкой систем программирования.

Ключевые слова*: системы программирования, декомпозиция программ, реализационная прагматика, парадигмы программирования, критерии декомпозиции, семантические системы, определение языков программирования*

ВВЕДЕНИЕ

Исследователи истории утверждают, что сложнее всего описывать историю человеческой мысли. В этом отношении история вычислительной техники и программирования обладает бесспорным преимуществом. Значительная часть истории мысли в этой сфере представлена в виде языков и систем программирования с корпусом программ и информационных систем, созданных на их основе, при возможности обсудить роль и путь идей с очевидцами [1]. Доступны интернет-

ресурсы, содержащие подробные описания языков программирования (ЯП), свободно распространяемые реализации систем программирования (СП), представления программ и комплектов поставки практичных приложений информационных систем (ИС). Имеется ряд систематизированных интегральных представлений взаимосвязей между сотнями и тысячами ЯП, отражающих их взаимовлияние [2] и парадигмальное сходство [3]. Ценность этих обширных материалов не вызывает сомнения. Целесообразно дополнить их средствами для выражения более детальных отношений и оперативного восприятия специфики анализируемых языков и систем.

В статье рассмотрено несколько форм, нацеленных на решение проблемы лаконичного представления особенностей определения языков программирования на уровне семантических систем, примеров программ и численных характеристик. Предполагается, что, используя такие формы, можно выстроить метрическое пространство, полезное при измерении понятийной сложности конструкций, поддержанных в определениях языков и систем программирования (ЯСП). Такое пространство может быть применено при сравнении парадигм программирования (ПП), потенциала схем и моделей, используемых при разработке программ, оценки уровня новизны создаваемых ЯП, а также при выборе критериев декомпозиции программ.

РЕЗУЛЬТАТЫ ПАРАДИГМАЛЬНОГО АНАЛИЗА

Парадигма программирования (ПП) как стиль мышления связана с компромиссом между особенностями решаемых задач и методами их решения с помощью программ, выполняемых автоматом. Задачи существенно различаются по следующим направлениям:

- стабильная постановка задачи, решаемой с помощью конечного автомата;
- развивающаяся постановка задачи, решаемой с помощью расширяемого автомата;
- задача бесперебойного функционирования, для решения часто требующая комплекса взаимодействующих автоматов.

Задачи каждого направления связаны противопоставлением трудно совмещаемых целей решения, таких, как академические, научно-исследователь-ские проблемы или производственные, практично-прикладные вопросы.

Соответствие программы и устройство конечного автомата обычно подразумевают, что имена, введённые в программе, в каждой позиции исполнения программы обладают одним значением. Для расширяемого автомата допускается возможность пополнения значений имени, включая перебор определений, представленных в программе, или их выбор по конкретному механизму. Это позволяет при развитии постановки задачи наследовать ранее отлаженный автомат.

Использование комплекса взаимодействующих автоматов позволяет компонентам декомпозированного представления программы сопоставлять отдельные автоматы, что делает возможным поддержку бесперебойного функционирования, повышения производительности программы, а также представление схем взаимодействия процессов, отвечающих трудно удостоверяемым критериям, таким как надёжность, безопасность, правильность и др. При масштабировании комплексов обычно провозглашают возможность сведения комплекса до одного автомата. Такое сведение для производственных задач может приводить к потере случаев одновременности, важных для реальной практики.

Системное программирование связано с решением задач всех таких направлений, причём одновременно и научно-исследовательской, и производственной направленности. Стабильные, желательно математические, постановки задач системного программирования обусловлены требованиями эквивалентности используемых и преобразуемых представлений программ. Развивающиеся постановки задач системного программирования связаны с динамикой совершенствования элементной базы, изменениями определений реализуемых и вновь создаваемых языков программирования, а также с ростом квалификации как разработчиков, так и пользователей программных систем. Задачи бесперебойного функционирования неизбежны для любых систем реагирования на события реального времени и массового применения, например, поддержки пользователей или организации учебного процесса.

Различия в средствах для решения разных направлений постановок задач можно выразить комплектами примеров представления программ. Анализ и сравнение большого числа ЯП разного уровня позволили выделить наиболее существенные характеристики для выражения парадигмальных различий в форме, подобной онтологическим таблицам [4]. Соответствующий пример на материале языка Lisp приведен в таблице 1.

Таблица 1. Парадигмальная характеристика Pure Lisp, чисто функционального подмножества языка Lisp

Параметр	*Описание*
Эксплуатационная прагматика ЯП[1]	Язык учебного назначения, созданный специально для изучения методов функционального программирования на языке Lisp, успешно применяется при решении задач с исследовательским компонентом, требующих быстрой отладки
Особенности системы понятий	Все понятия сведены к разным категориям понятия «функция», унифицированы представления функций и значений, выполнение программы рассматривается как отображение списка аргументов в результат
Перечень понятий, распознаваемых на уровне абстрактного синтаксиса	Символьное выражение (S-выражение), атом, вычислимая форма, переменная, константа, ветвление, элементарные функции, определение функции, именование функции, применение функции, аргумент функции, значение
Базовые средства ЯП	NIL, CONS, CAR, CDR, ATOM, EQ, QUOTE, EVAL, COND, LAMBDA, DEFUN
Семантические расширения[2]	Работа с числами и строками рассматривается как вспомогательная семантика. Ввод и вывод данных считаются псевдо-функциями, обслуживающими отладку. Отдельное расширение языка поддерживает функции с пост-вычислением аргументов – специальные функции
Регистры абстрактной машины	Стеки для хранения результатов, локальных значений переменных, вычисляемой формы и дампа, обеспечивающего защиту контекста вычислений (S E C D)
Категории команд абстрактной машины	Засылки в стек, вычисления над стеком, пересылки из стека, ветвления, применение функции, выходы из ветвлений и функций, восстановление контекста, поддержка рекурсии
Реализационная прагматика СП[3]	Списки из бинарных узлов, содержащих пару тэгированных указателей. Тэг показывает тип данного, адресуемого указателем. Автоматизировано освобождение памяти служебной программой «Сборщик мусора»
Парадигмальная специфика ЯСП [4]	Исторически основополагающий классический язык, поддерживающий функциональное программирование в полном объёме

Аналогичный пример такой же парадигмальной характеристики языка Pascal приведён в Таблице 2.

Таблица 2. Парадигмальная характеристика подмножества языка Pascal, поддерживающего методику структурного программирования

1 Уровень языка, его ниша в полном жизненном цикле программ, соответствующая технология

2 Вспомогательные семантики, их описание относительно концептуальных языков

3 Структуры данных и традиционно подразумеваемые механизмы реализации

4 Роль языка в формировании поддерживаемой им парадигмы и/или перечень поддерживаемых парадигм

Параметр	*Описание*
Эксплуатационная прагматика ЯП	Язык учебного назначения, созданный для обучения студентов методам программирования решений задач, готовых для эффективной реализации при поддержке автоматным моделированием
Особенности системы понятий	Программа и данные – раздельные сущности. Выполнение программы сводится к шагам изменения состояний памяти, хранящей данные. Используемые в программе идентификаторы подчинены иерархии областей видимости, задаваемых вложенностью определений процедур и функций. Процедуры и типы данных не являются значениями. Возможно конструирование новых типов данных и приведение типа данных к заданному
Перечень понятий, распознаваемых на уровне абстрактного синтаксиса	Скаляр, вектор, значение, ключевые слова, константа, переменная, тип данных, операция, выражение, операнд, элемент вектора, сравнение, действие, последовательность действий, ветвление, цикл, определение процедуры/функции, вызов процедуры/функции
Базовые средства ЯП	TRUE, FALSE, перечислимые значения, :=, a[i], IF, WHILE, PROCEDURE, st1 ; st2 .
Семантические расширения	Разные виды чисел, записи, множества, указатели, метки, передачи управления, функции, переключатели, конструирование типов данных, ввод-вывод, файлы, строки, библиотеки
Регистры абстрактной машины	Стеки промежуточных результатов, локальных переменных и параметров, выполняемой процедуры и вектор памяти (S E C M)
Категории команд абстрактной машины	Засылки в стек, вычисления над стеком и памятью, пересылки из стека и локальных переменных, ветвления, передачи управления, вызовы процедур
Реализационная прагматика СП	Память распределяется статически по блокам заданного размера, обработка векторов использует вычисление смещений от базового адреса и подразумевает контроль границ отведенной под вектор памяти, при необходимости привлекаются библиотеки, поддерживающие ввод-вывод, работу с файлами и динамической памятью
Парадигмальная специфика ЯСП	Прецедент строго определения языка программирования, обладающего не слишком высокой сложностью в реализации, поддерживающего методику результативного структурного программирования. Может выполнять роль эталонного монопарадигматического языка при сравнительном анализе языков и определении их парадигматической характеристики

Такие таблицы могут иметь прикладное и образовательное значение, но по оперативности восприятия они заметно уступают обычным средствам деловой графики, успешно применяемым для представления сложных данных, традиционно сводимых к числовым соотношениям. Следует отметить зависимость понимания смысла от владения оттенками профессиональной терминологии и

вариациями определения понятий в разных ЯП. Современная практика программирования имеет тенденцию к использованию примеров текстов или шаблонов программ без выделения понятийной структуры. Важно, чтобы такие тексты на разных ЯП допускали ясное сопоставление и сведение к общим интуитивным понятиям. Наиболее объективные понятия такого рода связаны с архитектурными моделями, методами реализации СП и классификацией решаемых задач.

АРХИТЕКТУРНЫЕ МОДЕЛИ

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 детализацию на уровне более тонких аппаратных решений и расширение на уровне периферийных устройств, что даёт первое приближение для выбора основных видов семантических систем в языках программирования [5, 6]. Рассматривая любые семантические системы, важно отметить разницу в характере выполнения функций таких систем. Так, для любого множества значений V реализационно различимы категории функций F для методов вычислений FE: (V* → V+)[5], средств доступа к памяти FM: (T : N → V)[6], особенностей управления вычислениями FC: (F → {0, 1})* и обратимой комплексации и структурирования данных FS: (A → K) U (A ← K) = (A ↔ K)[7].

Интересно, что такие виды семантических систем 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})* используют разметку

5 V* – произвольное число значений, возможно ни одного, V+ – хотя бы одно значение

6 Здесь N – множество адресов, T – таблица связей между адресами и значениями

7 Здесь A – атомарные неделимые данные, K – конструируемые структурированные данные

запрограммированных действий для выделения выполняемых, а также обычно понятие адресуемой памяти. Обратимая комплексация или структурирование данных в современных архитектурах FS: (A ↔ K) требуют определения границы между атомарными (A) и сложными, конструируемыми (K) объектами с возможностью как наращивания сложности, так и упрощения любых построений с учётом разных условий. Это приводит к представлению о кумулятивной шкале видов семантических систем на основе классификации архитектурно-подобных видов функций, представленных в Таблице 3.

Таблица 3. Виды семантических систем

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

ПОНЯТИЙНАЯ МАТРИЦА

Одним и тем же наборам и категориям функций могут соответствовать разные правила **R**, определяющие методы реализации СП и, следовательно, дополнительный фактор для выбора средств, иллюстрируемых с помощью текстов программ в качестве примеров отдельных понятий ЯП.

RE – обычные арифметические вычисления, отображающие произвольный ряд **значений** аргументов в не менее чем один результат,

RM – символьные вычисления, подставляющие **представления** аргументов без их предварительного вычисления,

RC – частичные или смешанные вычисления, лавирующие между вычислением и подстановкой в зависимости от разных условий,

RS – обобщённые и параллельные вычисления, оперирующие организацией процессов как множеством потоков над комплексами из разных устройств.

Представление таких различий можно выразить, дополнив горизонтальную шкалу видов функций вертикальной кумулятивной шкалой моделей вычислений или методов применения функций к значениям (R), что будет выглядеть как матрица семантических систем – понятийная матрица со строками RE, RM, RC, RS.

Строка **RE** этой матрицы представляет уровень базовой семантики ЯП (см. Таблицу 4).

Таблица 4. Семантическая декомпозиция минимального ядра ЯП

№ столбца	***V***	***E***	***M***	***C***	***S***
Категории 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 поддержки 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 – вектора, списка, очереди и т. п.

Появление конструируемых данных является достаточным основанием для укрупнения любых видов семантических систем, что представлено строкой RM в таблице 5.

Таблица 5. Семантическая декомпозиция макрорасширения ядра ЯП

№ столбца	***V***	E	***M***	***C***	***S***
Категории 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 в СП как препроцессор. Так же бывает устроена техника кодогенерации и обработки шаблонов при компиляции программ. 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 с определённой дисциплиной доступа для поддержки иерархии, 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 объёмов разрабатываемых программ

отчасти достигается автоматизацией контроля некоторых условий корректности применения операций и функций к их операндам в определённых границах. Становятся важными понятия «предикат» и «тип данных», что выражено строкой RC таблицы 6. По мере расширения границ программа может стать более универсальной, способной к разумному поведению на типовых входных данных, что можно представить таблицей 6.

Таблица 6. Семантическая декомпозиция диагностического дополнения ядра ЯП

№ столбца	***V***	E	***M***	***C***	***S***
Категории 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 данных	Логика	Варианты

Границы – методы проверки вычислимости функций и выполнимости заданий. Цель представления границ – снижение трудоёмкости 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 договорённостях и схеме реагирования на её результаты.

Типы данных – связывание типа значения с указателем или переменной, хранящей нетипизированный код значения в памяти.

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

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

Зависимость от учёта границ применимости программных компонент при сборке из них более сложных программ обычно преодолевается обобщением до универсальных функций, что представлено в строке RS таблицы 7. Введена дополнительная нумерация строк для более очевидного сопоставления строк и столбцов в таблицах, начиная с номера 8.

Таблица 7. Семантическая декомпозиция практичного обобщения ядра ЯП

№ строки	*№ столбцов*	***V***	**E**	***M***	***C***	***S***
	Категории 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	Исключения	Предикаты	Типы данных	Логика	Варианты
S	**RS: Общность**	Неопределённость	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.

Неопределённость – 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 приёма данных с внешних устройств и размещения данных на внешних носителях данных, включая средства доступа к устройствам с уровня программы. Стандартно имеется в виду приём 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 категориям функций относительно схемы применения функций к аргументам на уровне ЯП при реализации СП соответствуют определённые позиции специальной понятийной матрицы, полученной как двумерная кумулятивная шкала. При анализе ЯП примеры минимальных средств его применения можно расположить по клеткам понятийной матрицы, рассматривая число или объём таких примеров как одну из численных характеристик сложности.

ПРИМЕРЫ ФОРМ ПРЕДСТАВЛЕНИЯ РЕЗУЛЬТАТОВ СРАВНЕНИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Результаты анализа языка Pure Lisp представлены в таблице 8, а подмножества языка Pascal – в таблице 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, отображаемой в абстрактную машину SECD, обладающую независимыми регистрами [7]. Функции такой системы могут принадлежать основному множеству, и правило применения функций может входить в набор функций.

Таблица 8. Понятийная матрица определения языка Pure Lisp

№ стр	*№ столбца*	**V**	**E**	**M**	**C**	**S**

оки	*Виды функций*	V	F: V* → V	AL: Atom → V	(F → {NIL, T})*	A ↔ K
E	Ядро	• NIL; и атомы	• eq: V V→{NIL,T}	▪AL = (… (NIL . NIL) (T . T))	• Eval: Sexpr AL→ V • quote	▪ cons: V1 V2 → (V1 . V2) ▪ car: (V1 . V2) →V1 ▪ cdr: (V1 . V2) →V2
M	Специаль-ные функции	• Sexpr: (V . V) •(V . . .) ; список	• (lambda (x1 x2 ...) form) • (label FN F)	▪ pairlis: XL YL → AL ▪assoc: X AL → Y	• ((lambda (x1 x2 ...) form) v1 v2 …) •((label FN F) v1 v2 …)	▪ list ; Список ▪ (F v1 v2 . . .) ; Форма
C	Границы	• NIL • «Нет определения переменной» • «Нет определения функции»	• atom: V→{NIL,T} • apply: F V* AL → V • ERROR	▪ null ▪ Свободные переменные ▪ Типы **значений**	• cond • FUNCTION	▪ Строки ▪ Clozure; = Funarg; = замыкание
S	Общность (Практичность)	•Числа •Строки	•evlis: (form*)→V* •evcon: ((form form)*) → V • **+ - * /**	▪ Псевдо-функции ▪ Свойства атома	• map • reduce • lazy	▪ READ ▪ PRINT

Понятийная матрица, характеризующая средства языка Pascal, обладает меньшей кумулятивностью. Pascal определён в рамках семантической системы, отображаемой в пи-код [8], в котором, в отличие от SECD, регистры SEC размещены в общей памяти. Составляющие такой системы строго разделены. Множество V фиксировано по составу, неявное определение правил организации вычислений R не допускает программируемых модификаций, функции могут выполнять роль данных в терминах указателей, связь с внешним миром определяется вне языка через библиотечные модули.

Таблица 9. Понятийная матрица определения подмножества языка Pascal

№	*№ столб.*	**V**	**E**	**M**	**C**	**S**

строки	*Виды функций*	V	F: V* → V	M : Id→ V+F	(F → {True, False})*	V N ↔ Array [N] of V
E	Ядро	•type = (a, b, ...)	• = <> • pred succ	▪Id ▪var X : int; ▪X := 1; ▪label L; ▪L: a := 1;	• (1*(2+X)) • S1 ; S2 •ключевые слова • goto L;	▪s: array [1 .. 9] of int; ▪s [5] := x; ▪x := s [5]
M	Расширение	•false true • maxint • intege •(1,3,6,9, ...) • const •A=1;B=2; •type ta = 1 .. 10;	• + - * / div mod • F (1, x, ...)	▪var va : ta; ▪begin . ▪ begin ... end .. end ▪func F subr S ▪end.	•if X then S1 else S2 •begin S1 ; S2... end •S (X, Y) •forward;	▪s: array [1 .. 9] of array [1 .. 3] of int; ▪s [5,2] := x; ▪x := s [5,2]
C	Границы	•«Нет определения» •“не соотв. ТД :=” “не соотв. ТД операнда” “не соотв. ТД параметра” •“выход из диапазона”	• = < > >= <= •F: V → {True, False} •in •not or and	▪Типы **данных**: ▪int ▪char ▪array ...	•case •Соответствие ТД	▪record ▪set ▪record … case
S	Практичность	•Указатель: •Nil •X^	•Функция-параметр	▪Процедура-параметр, ▪*/Стандартные процедуры* [8] ▪*NEW* ▪*DESPOSE, ...*	•while for until	▪*/Стандартные прцедуры* [9] ▪*READ* ▪*WRITE*

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. Наполнение таких матриц можно выполнять по мере изучения отдельных ЯП. Полученные оценки можно представлять, как это сделано в Таблицах 10–12, допускающих автоматическое производство.

Таблица 10. Представление результатов сравнения понятийной сложности определений языков Pure Lisp и Pascal (число пунктов в клетках Таблиц 8, 9)

8 Библиотечное дополнение

9 Библиотечное дополнение

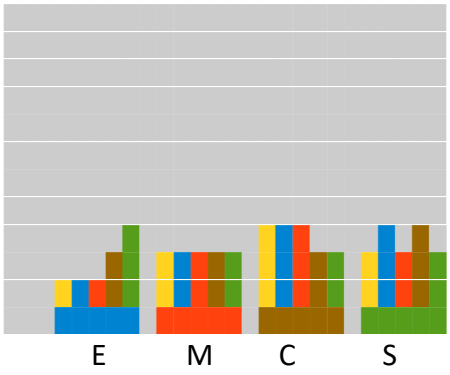
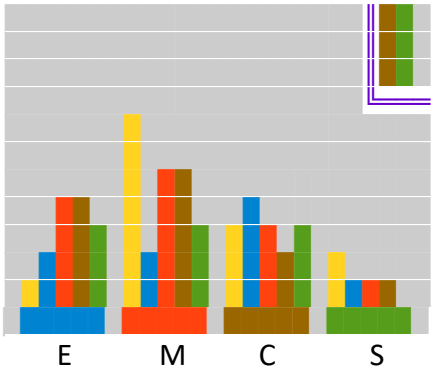
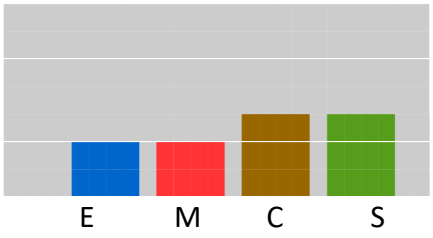
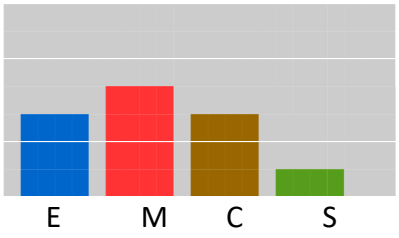
Lisp						**Pascal**					
	V	E	M	C	S		V	E	M	C	S
E	1	1	1	2	3	E	1	2	5	5	3
M	2	2	2	2	2	M	7	2	5	4	3
C	3+	3	3	2	2	C	3+	4	3	2	3
S	2	3	2	3	2	S	2	1	1 /**3***	1	0 /**3***

Таблица 11. Более краткие формы представления результатов сравнения сложности ЯП (число пунктов в клетках, строках и столбцах Таблиц 8, 9)

Lisp – 43+	**Pascal – 57+ / 6***	Комментарий
(E (v1 e1 m1 c2 s3) M (v2 e2 m2 c2 s2) C (v3+ e3 m3 c2 s2) S (v2 e3 m2 c3 s2))	(E (v1 e2 m5 c4 s3) M (v7 e2 m5 c5 s3) C (v3+ e4 m3 c2 s3) S (v2 e1 m**1+3*** c1 s**0+3***))	Число примеров в клетках матрицы с сохранением координат клеток
(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 5 3) C (3+ 4 3 2 3) S (2 1 **1+3*** 1 **0+3***))	Число примеров в клетках матрицы с сохранением координат строк.
((E 8) (M 10) (C 13+) (S 12))	((E 15) (M 22) (C 15+) (S 5 **/6***))	Число примеров в строках матрицы с сохранением координат строк
V E M C S E 1 1 1 2 3 M 2 2 2 2 2 C 3 3 3 2 2 S 2 3 2 3 2	*V E M C S* E 1 2 5 4 3 M 7 2 5 5 3 C 3 4 3 2 3 S 2 1 1 1 **0**	Число примеров в клетках матрицы с сохранением координат клеток
Столбцы: 8 9 8 9 9	Столбцы: 13 9 14 16 9	Число примеров по столбцам
Строки: 8 10 13 12	Строки: 15 22 15 5	Число примеров по строкам

Полученные формы позволяют ускорить сопоставление характеристик разных ЯП, с точностью до учёта соответствия координат ячеек матрицы и их наполнения. Более яркая и быстрая ассоциация получается привлечением цвета и диаграмм, подобно средствам деловой графики, используемым для представления схемно-интегральных характеристик сложных явлений.

Таблица 12. Визуальные формы представления результатов сравнения сложности ЯП

Lisp – 40+	**Pascal – 82+ /6***	Комментарий
		Высоты столбцов пропорциональны числам из Таблицы 10. Сверху справа – библиотечные модули
		Площади прямоугольников пропорциональны числам примеров по строкам из табл. 11

ДИСТАНЦИЯ В ПОНЯТИЙНОЙ СЛОЖНОСТИ СЕМАНТИЧЕСКИХ СИСТЕМ

Согласно Венской методике, при определении ЯП различимы уровни абстрактной семантики, конкретного языка и конкретной машины.

1) (АС → АМ) – консервативное расширение АМ до АС или монотонный спуск от АС к АМ, декомпозированный в соответствии с выбором вспомогательных и прикладных семантик, что делает их функционально равнозначными [9].

2) КЯ ↔ АС – отображение КЯ в АС и обратно, что делает их семантически эквивалентными.

3) АМ → КМ – эмуляция АМ на подмножестве КМ, пригодном для реализации АМ, что делает их прагматически равносильными.

Это значит, что при реализации СП понятийная матрица ЯП расслаивается на три слоя, каждый из которых наследует сложность ЯП. Учитывая, что за время реализации ЯП его определение обычно претерпевает развитие, возникает необходимость системного обобщения, позволяющего сглаживать рост трудоёмкости варьирования реализации СП.

4) Системное обобщение для поддержки независимого развития разных уровней реализации СП.

E: { АС ↔ АМ } – множество консервативных расширений АМ до АС и монотонных спусков от АС до АМ, декомпозированных в соответствии с выбором вспомогательных и прикладных семантик [9].

M: {КЯ} ↔ АС – множество ЯП, эквивалентных АС.

C: АМ ↔ {КМ} – множество машин, пригодных для реализации АМ.

S: E+M+C+{РП} – системное обобщение можно рассматривать как концептуальное замыкание пространства конкретных ЯП, обладающих общими семантическими системами, и машин, пригодных для нетрудоёмкого переноса на них реализованных СП. Это измерение обобщает разнообразие абстрактных, изобразительных и операционных средств. Таким образом возникает прагматический реализационный куб, выглядящий как кубик Рубика с четырьмя клетками по каждому из трёх измерений (категории функций, методы вычисления, реализационные слои – уровни реализации).

Дальнейшие усложнения пространства решений, принимаемых при создании СП, направлены на инструментальную поддержку процесса разработки программ и схем, используемых при проектно-модельной подготовке этого процесса, выборе основных технологий программирования и развитии сферы приложения ЯП.

5) Технологическая поддержка (преобразования программ, факторизация, масштабирование относительно аппаратуры, вывод моделей и верификация, реорганизация и настраиваемость).

6) Человеческий фактор (индивидуальная разработка, группы, команды, фирмы-проекты).

7) Практический каркас (изученность задачи, полный жизненный цикл программ, отладка, работоспособность программы, тесты, улучшаемость, уточнение, утончение).

8) **Эксплуатационная карта** СП – уровень пользователя (интерфейс, образцы сценариев и данных, учебные **примеры** и задачи с решениями в виде отлаженных программ, лексикон и руководства).

9) Инструментальное окружение (рабочее место системного программиста – БНФ, навигация-декомпозиция, интеграция-библиотеки компонент, систематизированных по критериям парадигмальной декомпозиции).

Можно обратить внимание, что слой 8 может содержать примеры из понятийной матрицы ЯП.

ЗАКЛЮЧЕНИЕ

Описанные формы можно рассматривать как конкретизации понятийной сложности по Колмогорову [10]. Близкие работы начаты ещё в середине 1960-х годов, когда возникла задача определения ЯП, поддерживающих процессы разработки СП при создании новых ЯСП. В конце 1970-х годов в рамках работ по проекту МАРС [6] был выполнен обзор динамики развития компьютерных конфигураций, что обосновывает выбор основных семантических систем [5]. В те годы был выполнен ряд серьёзных попыток создания языков системного программирования, позволяющих конструировать системы программирования [5, 11–14]. Наиболее продвинутые из них, такие, как Венская методика, не учитывали оценок трудоёмкости и понятийной сложности системного программирования, но составили базу Си-ориентированным LEX-YACC, идеи которых унаследовали Clang-LLVM.

Предложенную формализацию при оценке сложности и трудоёмкости программирования можно дополнить разделением требований к постановкам задач по сферам применения на академические и производственные, а по уровню изученности – на чёткие, развиваемые и усложнённо трудоёмкие.

Благодарности

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований, проекты №№ 15-07-06345, 18-07-01048.

СПИСОК ЛИТЕРАТУРЫ

1. http://http://www.sorucom.org/ – сайт Международной конференции «Развитие вычислительной техники и ее программного обеспечения в России и странах бывшего СССР: история и перспективы».

2. https://www.levenez.com/lang/ – сайт "Computer Languages History".

3. http://progopedia.ru/ – сайт «Энциклопедия языков программирования».

4. *Городняя Л.В.* Парадигмы программирования: анализ и сравнение. Сиб. Отделение Рос. Акад. наук, Ин-т систем информатики им. А.П. Ершова. Новосибирск: Изд-во СО РАН, 2017. 232 с.

5. Лавров С.С. Методы задания семантики языков программирования// Программирование. 1978. № 6. С. 3–10.

6. *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.

7. *Хендерсон П.* Функциональное программирование. М.: Мир, 1983. 349 с.

8. *Вирт Н.* От Модулы к Оберону // Системная информатика. Вып 1. Проблемы современного программирования. Новосибирск: Наука. Сиб. отделение, 1991. С. 63–75.

9. Городняя Л.В. Резервы синтаксически ориентированного конструирования систем программирования. //Научный сервис в сети Интернет: труды XIX Всероссийской научной конференции (18–23 сентября 2017 г., г. Новороссийск). М.: ИПМ им. М.В. Келдыша, 2017. С. 120–129. URL: http://keldysh.ru/abrau/2017/ proc.pdf

10. Колмогоров А.Н. Три подхода к определению понятия «количество информации»// Проблемы передачи информации. 1965. № 1 (1). С. 3–11.

11. *Фуксман А.Л.* Технические аспекты создания программных систем. М.: Статистика, 1979. 180 с.

12. *Koster Cornelis H.A.* Compiler Description Language. CDL3 manual. The Netherlands, August 18, 2004. http://www.cs.ru.nl/cdl3/cdl3.pdf

13. *Wulf W.A., Russel D.B., Habermann A.N.* BLISS: A Language for Systems Programming// CACM. 1971. V. 14. No 12. P. 780–790.

14. *Гололобов В.И., Чеблаков Б.Г., Чинин Г.Д.* Описание языка ЯРМО. Новосибирск. Препринты No 247, 248 ВЦ АН СССР, Сибирское отделение.

FORMS OF REPRESENTATION OF RESULTS OF PARADIGMAL ANALYSIS OF PROGRAMMING LANGUAGES

L. V. Gorodnyaya

A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Novosibirsk State University, Novosibirsk

lidvas@gmail.com

Abstract

The purpose of the article is the choice of presenting the results of a comparison of programming languages, convenient for assessing the expressive power of languages and the complexity of the implementation of programming systems. Forms of such a presentation should be adapted to substantiate practical criteria for program decomposition, which can be viewed as an approach to solving the problem of factorization of very sophisticated definitions of programming languages.

The relevance of choosing concise and quickly perceived forms is useful for working in the rapidly developing space of new problem-oriented programming languages. Along the way, you can show the distance in the conceptual complexity between programming and the development of programming systems.

Keywords*: programming systems, program decomposition, decomposition criteria, semantic systems, implementation pragmatics, programming paradigms, definition of programming languages*

REFERENCES

1. http://http://www.sorucom.org/ – Website of the International Conference "The development of computing equipment and its software in Russia and the countries of the former USSR: history and prospects"

2. https://www.levenez.com/lang/ – Website of the "Computer Languages History"

3. http://progopedia.ru/ – Website of the "Encyclopedia of programming languages"

4. *Gorodnyaya L.V.* Paradigms of programming: analysis and comparison / Rus. Acad. of Sciences, Sib. Branch, A.P. Ershov Institute of Informatics Systems. Novosibirsk: Izdat. SB RAS, 2017. 232 p.

5. *Lavrov S.S.* Methods for specifying the semantics of programming languages. // Programming. 1978. No 6. P. 3–10.

6. *Kotov V.E.* MARS: architecture and languages for realization of parallelism // System Informatics. Issue. 1. Problems of modern programming. Novosibirsk: Science. Sib. Branch, 1991. P. 174–194.

7. *Henderson P.* Functional programming. Moscow: Mir, 1983. 349 p.

8. Wirth N. From Modula to Oberon // System informatics. Issue 1. Problems of modern programming. Novosibirsk: Science. Sib. Separation, 1991. P. 63–75.

9. *Gorodnaya L.V.* Reserves syntactically oriented design of programming systems // Scientific service on the Internet: works of the XIX All-Russian Scientific Conference (September 18–23, 2017, Novorossiysk). M.: IPM them. M.V. Keldysh, 2017. P. 120–129. URL: http://keldysh.ru/abrau/2017/proc.pdf

10. *Kolmogorov A.N.* Three approaches to the definition of "amount of information"// Problems of Information Transfer. 1965. No 1 (1). P. 3–11.

11. *Fuksman A.L.* Technical aspects of creating software systems. Moscow: Statistics, 1979. 180 p.

12. *Koster Cornelis H.A.* Compiler Description Language. CDL3 manual. The Netherlands, August 18, 2004. http://www.cs.ru.nl/cdl3/cdl3.pdf

13. *Wulf W.A., Russel D.B., Habermann A.N.* (1971). BLISS: A Language for Systems Programming// CACM. 1971. V. 14. No 12. P. 780–790.

14. *Gololobov V.I., Cheblakov B.G., Chinin G.D.* Description language YARMO. Novosibirsk. Preprints No 247, 248 EC of the USSR Academy of Sciences, Siberian Branch.

СВЕДЕНИЯ ОБ АВТОРЕ

ГОРОДНЯЯ Лидия Васильевна – старший научный сотрудник Института систем информатики им. А.П. Ершова СО РАН, доцент Новосибирского государственного университета, специалист в области системного программирования и образовательной информатики.

Lidia Vasiljevna GORODNYAYA – Senior Researcher of A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Associate Professor of Novosibirsk State University, a specialist in system programming and educational informatics.

email: lidvas@gmail.com

Материал поступил в редакцию 12 ноября 2018 года