О представлении результатов анализа языков и систем программирования

Л.В. Городняя

Институт систем информатики СО РАН, НГУ

Аннотация. Статья посвящена выбору формы для представления результатов сравнения языков и систем программирования, удобной для оценки выразительной силы языков и трудоёмкости реализации систем. Формализация приспособлена к парадигмальному анализу определений языков программирования и выбору практичных критериев декомпозиции программ, что можно рассматривать как подход к решению проблемы факторизации определений языков и систем программирования. В качестве основного параметра факторизации выбрана семантическая декомпозиция определений в рамках анализа парадигм программирования. Такой выбор позволяет выделять автономно развиваемые типовые компоненты систем программирования, которые могут быть приспособлены к конструированию различных информационных систем. Многие работы по методам разработки программных систем зависят от практичности подходов к декомпозиции программ, что можно рассматривать как проблему факторизации программ на базе языков программирования, отлаживаемых с помощью систем программирования. Решение этой проблемы полезно для сравнения парадигм программирования, потенциала используемых схем и моделей, оценки уровня новизны создаваемых языков программирования, а также при выборе критериев декомпозиции программ. Кроме того, их существование 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.

On the presentation of the results of the analysis of programming languages and systems

L.V.Gorodnyaya

A.P. Ershov Institute of Informatics Systems (IIS)

Abstract. The purpose of this study is the choice of the form to represent the results of the comparison of languages and programming systems, convenient for

assessing the expressive power of languages and the complexity of the implementation of systems. The formalization is adapted to the paradigm analysis of the definitions of programming languages and the choice of practical criteria for the decomposition of programs, which can be considered as an approach to solving the problem of factorization of the definitions of programming languages and systems. Semantic decomposition of definitions in the framework of the analysis of programming paradigms is chosen as the main parameter of factorization. This choice allows you to allocate autonomously developed standard components of programming systems that can be adapted to the design of various information systems. Many works on methods of development of software systems depend on the practicality of approaches to program decomposition, which can be considered as a problem of factorization of programs based on programming languages debugged with the help of programming systems. The solution to this problem is useful for comparing programming paradigms, the potential of the schemes and models used, assessing the level of novelty of the programming languages created, as well as for selecting the criteria for program decomposition. In addition, their existence allows to form a training methodology for the development of components of information systems. Along the way, the distance between the conceptual complexity of programming and the development of programming systems is shown.

Key words: definition of programming languages, programming paradigm,

Введение

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 языка программирования (ЯП).

В данной статье предлагается ряд решений для формализованного представления метрического пространства, полезного при измерении понятийной сложности конструкций, поддержанных в определениях языков и систем программирования (ЯСП). Такое пространство может быть применено при сравнении парадигм программирования (ПП), потенциала используемых при разработке программ схем и моделей, оценки уровня новизны создаваемых ЯП, а также при выборе критериев декомпозиции программ.

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 чётким разделением данных и программ, а также для языков управления базами данных.

2. 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 [3]. Рассматривая любые семантические системы, важно отметить 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 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 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 упрощения любых построений с учётом разных условий.

Это приводит к представлению о кумулятивной шкале семантических систем на основе классификации видов функций.

Таблица 1

Классы семантических систем

№ 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

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,

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. Различия классов семантических систем на уровне правил применения функций к значениям в ЯВУ выражены таблицами 2-5. Первая строка этой матрицы представляет уровень базовой семантики ЯП, существенные различия видов

[1] Обозначение уточнено А.В.Климовым.

функций семантических систем которого можно представить Таблицей 2. Вторая строка Таблицы 3 представляет уровень макрорасширений, что вместе с первой строкой позволяет характеризовать средства языков низкого уровня.

Таблица 2

Семантическая декомпозиция минимального ядра ЯП

№ 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 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 или т. п.

Таблица 3

Семантическая декомпозиция макрорасширения ядра ЯП

№ 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 в СП как препроцессор. Также 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 переменных», удобно проверяемые при компиляции, что представлено третьей строкой Таблицы 4. По мере расширения границ программа может стать достаточно универсальной, способной к разумному поведению на любых входных данных, что выражено в Таблице 5, в которой добавлена слева колонка с обозначениями строк полученной понятийной матрицы.

Таблица 4

Семантическая декомпозиция диагностического дополнения ядра ЯП

№ 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 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 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 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.

Таблица 5

Понятийная матрица:

Семантическая декомпозиция практичного обобщения ядра ЯП

№ строк	*№ столбцов*	*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	Исключения	Предикаты	Типы переменных	Логика	Варианты
S	**RS: Общность**	Неопределённость	Мультиоперации	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 видам функций относительно схемы применения функций к аргументам на уровне ЯП при реализации СП соответствуют определённые 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 таблице 6.

Таблица 6

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

№ 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 вектора 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 определённые 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 имён — 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	Появляется понятие «предикат» и специальная функция ERROR для выбора обработчиков диагностических ситуаций и продолжения недоопределённых 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.
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 неопределённостей, 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.

4. Пример представления результатов сравнение ЯП

Результаты анализа языка Pure Lisp представлены в Таблице 7, а языка Pascal в Таблице 8. 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 [4]. Функции такой системы могут принадлежать основному множеству и правило применения функций может входить в набор функций.

Таблица 7

Понятийная матрица определения языка Pure Lisp

№ стр	*№ столбца*	V	**E**	**M**	C	**S**
	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	Специальные функции	•Sexpr: (V . V) •(V . . .) ; список	•(lambda (x1 x2 ...) form) •(label FN F)	•pairlis X Y → AL •assoc: X AL → V	•((lambda (x1 x2 ...) form) v1 v2 …) •((label FN F) …)	•list ; Список •(F v1 v2 . . .) ; Форма
C	Границы	•NIL •«Нет переменной» •«Нет функции»	•atom:V→{NIL,T} •apply: F V* AL → V •ERROR	•null •Свободные переменные •Типы значений	•cond •FUNCTION	•Строки •Clozure = Funarg
S	Общность Практичность	•Числа •Строки	•evlis : (form*) → V* •evcon : ((form form)*) → V •+ - * /	•Псевдо-функции •свойства	•map •reduce •lazy	•READ •PRINT

Понятийная матрица, характеризующая средства языка Pascal, обладает меньшей кумулятивностью. Pascal определён в рамках семантической системы вида <V; F; R>, отображаемой в пи-код [5], 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.

Таблица 8

Понятийная матрица определения языка Pascal

№ стр	*№ 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	•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 •integer •(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* [2] •*NEW* •*DESPOSE, ...*	•while for until	• */Стандартные прцедуры* [3] •*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, что можно представить как в Таблицах 9-11.

Таблица 9

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

Lisp					
	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

Pascal					
	V	E	M	C	S
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**

Таблица 10

Краткие формы представления результатов сравнения сложности ЯП (число пунктов в клетках и строках таблиц 7, 8)

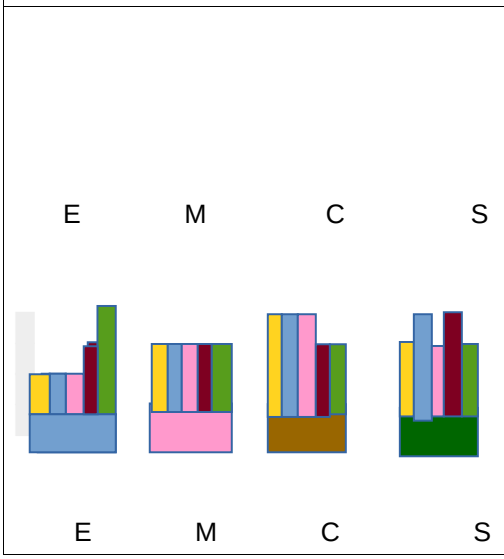
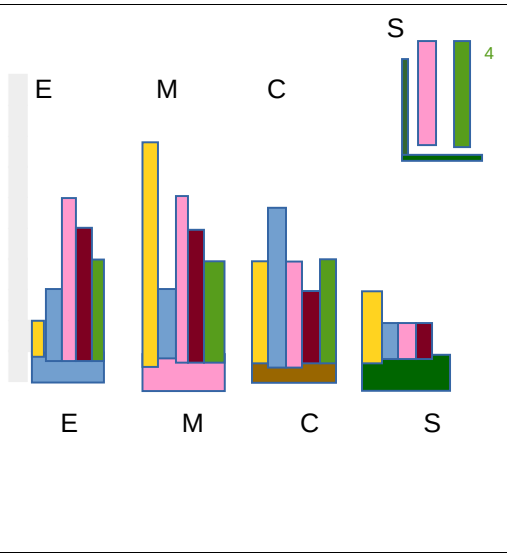
Lisp — 43+	**Pascal — 56+ / 6***
(E (1 1 1 2 3) M (2 2 2 2 2) C (3+ 3 3 2 2) S (2 3 2 3 2))	(E (1 2 5 4 3) M (7 2 5 4 3) C (3+ 4 3 2 3) S (2 1 **1+3*** 1 **0+3***))
((E 8) (M 10) (C 13+) (S 12))	((E 15) (M 21) (C 15+) (S 5 **/6***))

[2] Библиотечное дополнение

[3] Библиотечное дополнение

Таблица 11

Визуальная форма представления результатов сравнения сложности ЯП

Lisp — 40+	Pascal — 82+ /6*
	

5. Переход к СП

Для оценки сложности реализации СП требуется рассмотреть более мощное пространство решений, обусловленное с одной стороны развитием аппаратуры, особенно периферийных устройств, с другой стороны проблемами выбора практичных реализационных решений из некоторого, не всегда заранее определённого, множества [6]. Кроме того, не исключены разные ПП для внешне одинаковых конструкций, что можно пронаблюдать на внешнем сходстве присваиваний в языках Pascal, Val и Sisal. Учитывая, что за время реализации ЯП его определение обычно претерпевает развитие, возникает необходимость системного обобщения, позволяющего сглаживать рост трудоёмкости варьирования реализации СП, что схематично представлено в Таблице 12 как реализационный куб, состоящий из четырёх слоёв, представленных понятийными матрицами с номерами E, M, C, S.

4 Отделено минимальное число библиотечных процедур

Таблица 12

Реализационный куб понятий СП

Название слоя				***№ слоя***	*Спецификация семантических систем слоя*		
Универсальный слой 4				**S**	**{РП}↔{КЯ↔АС↔АМ↔КМ}**		
Операционный слой 3			**C**	**АМ→ КМ**			
Изобразительный слой 2		**M**	КЯ ↔ АС				
Абстрактный слой 1	**E**	**АС → АМ**					
№ стр	*№ столбца*	*V*	***E***	***M:***	***C:***	***S:***	
	Виды функций	***V***	***V* → V+***	***T : N → V***	*(F → {0, 1})**	***A ↔ K***	
E	**Вычисления (арифметика)**	Значение	Операции	Память	Управление	Структуры + цепочки	
M	**Подстановки (макро)**	Данное	Функции	Задание	Блоки	Иерархия + Стек	
C	**Границы (частичность)**	Диагностика Исключения	Предикаты + Error	Типы переменных	Логика	Варианты Множества. Комплексы	
S	**Общность. Практичность**	+ Error Мультизначения	Мультиоперации. Просачивание операций	**Внешний мир.(свойства)**	Отображения. Пространства итерирования	Ввод-вывод	

Такие реализационные кубы достаточны для представления результатов парадигмального анализа ЯСП, а также оценки сложности их разработки и реализации, например, по числу различимых средств на каждом слое куба в отличие от сложности применения [6]. Это значит, что от рассмотрения куба возможен переход к матрице, а от матриц можно перейти к векторам, подобно решениям по обработке многомерных структур в языках Fortran и APL.

Дальнейшие усложнения пространства решений при реализации ЯСП направлены на инструментальную поддержку процесса разработки СП и схем, используемых при проектно-модельной подготовке этого процесса, выборе основных технологий программирования и развитии сферы приложения. Нередко такие работы имеют ранг внутреннего заказа.

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 упражнения 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).

Заключение

Изложенный формализм можно рассматривать как конкретизацию понятийной сложности по Колмогорову [7]. Близкие работы начаты ещё в середине 1960-ых годов, когда возникла задача создания ЯП, поддерживающих процессы разработки СП при создании новых ЯСП. В конце 1970-ых в рамках работ по проекту МАРС [3] был выполнен обзор динамики развития компьютерных конфигураций, что обосновывает выбор основных семантических систем, допускающих эффективную реализацию. В те годы был выполнен ряд серьёзных попыток создания языков системного программирования, позволяющих конструировать системы программирования [2,8-11]. Наиболее продвинутые из них, такие как Венская методика, не учитывали оценок трудоёмкости и понятийной сложности системного программирования, но они составили базу Си-ориентированным LEX-YACC, идеи которых унаследовали Clang-LLVM.

Современные работы по проверке доказательств теорем используют понятие «λ-куб», близкое понятию «семантическая система» вида <V; F; R> без кумулятивного эффекта по горизонтали, дополненному учётом полиморфизма основных множеств [12]. В таких работах декларируется эквивалентность понятий «доказательство» и «программа», но в рамках чисто функционального программирования, как в языке Haskell [13].

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.

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 18-07-01048-а.

Литература

1. Мальцев А.И. Алгоритмы и рекурсивные функции. − М.: Наука. 1965 г., 392 с.
2. Лавров С.С. Методы задания семантики языков программирования // Программирование, 1978. N 6. С. 3-10.
3. Котов В.Е. МАРС: архитектуры и языки для реализации параллелизма. // Системная информатика. Вып 1. Проблемы современого программирования. − Новосибирск: Наука. Сиб. отд-ние, 1991. − с.174-194.
4. Хендерсон П. Функциональное программирование. – М.: Мир, 1983. – 349 с.

5. Вирт Н. От Модулы к Оберону. // Системная информатика. Вып 1. Проблемы современного программирования. − Новосибирск: Наука. Сиб. отд-ние, 1991. − С. 63-75.
6. 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
7. Колмогоров А.Н. Три подхода к определению понятия «количество информации». − Проблемы передачи информации, 1965. № 1 (1): с. 3–11.
8. Фуксман А.Л. Технические аспекты создания программных систем. – М.: Статистика, 1979. – 180 с.
9. Koster, Cornelis H.A. Compiler Description Language. CDL3 manual. − The Netherlands, August 18, 2004 http://www.cs.ru.nl/cdl3/cdl3.pdf
10. Wulf, W.A.; Russell, D.B.; Habermann, A. N. (1971). BLISS: A Language for Systems Programming. − CACM 14(12):780-790, Dec 1971
11. Гололобов В.И., Чеблаков Б.Г., Чинин Г.Д. Описание языка ЯРМО. // Новосибирск. Препринты No 247, 248 ВЦ АН СССР Сибирское отделение.
12. Barendregt, Henk Lambda Calculi with Types. − Handbook of Logic in Computer Science, Volume II, Oxford University Press.
13. Voevodsky, V. “A1-homotopy theory”, *Doc. Math.*, Proceedings of the International Congress of Mathematicians, Vol. I (Berlin, 1998), no. Extra Vol. I, 1998, 579–604 p.