УДК 004.43

Парадигмальный подход к факторизации определений языков и систем программирования

Городняя Л.В. (Институт систем информатики СО РАН,

Новосибирский государственный университет)

Статья посвящена проблеме факторизации определений языков и систем программирования. В качестве основного параметра факторизации выбрана семантическая декомпозиция в рамках анализа парадигм программирования. Такой выбор позволяет выделять автономно развиваемые типовые компоненты систем программирования. Типовые компоненты должны быть приспособлены к конструированию различных информационных систем. 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, факторизации определений, декомпозиция программ, критерии декомпозиции, парадигмы программирования, семантические системы, разработка программ, методы обучения программированию.*

1. Введение

Многие работы по методам разработки программных систем зависят от практичности подходов к декомпозиции программ, что можно рассматривать как проблему факторизации программ и средств их представления на базе языков программирования (ЯП), отлаживаемых с помощью систем программирования (СП). Общее понятие факторизации основано на декомпозиции сложных объектов в произведение более простых объектов, из которых перемножение даёт исходный объект при условии, что выбор более простых объектов обусловлен определённым фактором, позволяющим предельно простые объекты отличать от более сложных. Даже при факторизации чисел эта задача, имеющая строгое определение, становится сложной при переходе к большим числам. Такое понятие достаточно естественно может быть перенесено на представления программ, определений ЯП и реализаций СП при условии определения факторов для выделения простых объектов и определения техники произведения для восстановления исходного сложного объекта из результата факторизации.

Если техника произведения может быть сведена к общепринятым методам сборки программ из типовых компонентов, то выбор факторов для выделения простых составляющих в случае программ обладает значительным разнооборазием. Не удивительно, что на весьма представительных конференциях, посвященных обсуждению всех проблем программирования, рассматриваются лишь отдельные штрихи проблемы факторизации программ на материале конкретных наиболее популярных ЯП, таких как Си, Java, Python [7].

В данной статье рассматривается парадигмальный подход к выбору факторов и аналогов произведения для факторизуемых представлений программ, определений ЯП и реализаций СП, использующий семантическую декомпозицию формализованных определений. Проблема факторизации в программировании осложнена разнообразием используемых средств и широким спектром противоречащих друг другу критериев качества программ. Основная цель декомпозиции программ — обеспечение многократности использования отлаженных фрагментов. Именно многократность использования является аргументом доверия программным системам. Конструирование систем из отлаженных компонентов не всегда обладает удобной комбинаторикой. Возможность улучшения конструктива без чрезмерного роста трудозатрат на повторное программирование и отладку обусловлено выбором критериев декомпозиции программ. Нередко критерии декомпозиции отражают структуру декомпозируемого текста программы. Чаще имеет место учёт особенностей квалификации разработчиков программы. При решении сложных задач критерии декомпозиции программ отражают актуальность подзадач по шагам разработки. Долгоживущие программы эксплуатируются дольше времени её авторского сопровождения.

Системы программирования обычно проектируются именно как долгоживущие программы. Это достаточная причина высокой вероятности неавторского улучшения СП. Всё это объясняет, почему нужны объективные критерии декомпозиции ЯП, отражающие возможность автономного развития выделенных компонентов СП. К этой проблеме примыкает слабость методики обучения разработке компонентов программ и программных комплексов. Для разработки СП такая методика усложнена необходимостью полноценного ознакомления с основными и фундаментальными парадигмами программирования (ПП), включая параллельные вычисление [2]. Основные парадигмы имеют производственное значение. Они поддерживают жизненный цикл разработки программ. Фундаментальные парадигмы имеют образовательное значение. Они обеспечивают формирование расширяемой системы понятий, необходимых для освоения экстенсивно развивающегося пространства задач, решаемых с помощью ИТ.

Описание парадигмального подхода к факторизации программ начинается с общего представления о семантических системах, семантике языков разного уровня, особенностях СП и учебного программирования, дающего основания для относительного определения языков и систем программирования (ЯСП) с учётом прагматики и методики парадигмального анализа ЯП. В заключении приведены основные идеи инструментальной поддержки парадигмального подхода к факторизации ЯСП.

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

2. Общее представление

Достаточно объективной основой факторизации СП является семантическая декомпозиция определения ЯП. Результаты такой декомпозиции можно формулировать на базе предложенного С.С. Лавровым понятия "семантическая система". Это понятие расширяет понятие «алгебраическая система» заданием явного правила вычислений [4]. Для точности приведены пояснения к используемым общеизвестным терминам, необходимым для чёткого описания рассматриваемых методов декомпозиции программ. Эти методы в перспективе приводят к парадигмальному анализу определений ЯСП.

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,

Семантически подобные системы в разных ЯП могут обладать разными особенностями реализационной прагматики (РП). Прежде всего это относится к различию систем по схемам пре- или пост-вычислений. Далее реализация ЯП может тяготеть к интерпретации или компиляции программ. При переходе к СП могут различаться методы реализации функций техникой макроподстановки или вызова подпрограмм. Список других штрихов реализационных различий можно продолжать. Это означает существование поливариантности компонентов РП, поддерживающих разные парадигмы программирования. Такие компоненты пригодны для автономного развития.

Практичные критерии декомпозиции, кроме того, требуют учёта и представления специфики определений типовых компонентов в ясной форме, допускающей речевую практику. Помимо формальной определённости класса семантических систем нужна лингвистическая ясность, дающая достаточные основания для лаконичного объяснения

особенностей каждой семантической системы, что можно соотнести с отмеченной в 1970-е А.П. Ершовым проблемой Лексикона программирования, осложнённой свойством естественных языков изменять словарь при смене поколений. Кроме того, выбор основных семантических систем следует обосновывать прецедентом аппаратной реализации, показывающим достижимость эффективной реализации.

Обычно для любого множества значений **V** реализационно различимы следующие виды функций семантических систем:

- методы вычислений (**V* → V+**),
- средства доступа к памяти (**T : A → V**),
- особенности управления вычислениями {**F → {0,1}**},
- обратимая комплексация данных (**E ↔ S**).

Таким видам соответствуют разные правила **R**, определяющие классы семантических систем, особенности реализации которых описаны ниже, в таблицах 1-5. Пояснения к словам, размещённым в клетках таблиц, не являются определениями терминов, слова выполняют лишь роль индексов для размещения в соответствующих клетках неформализованных понятий и средств ЯП, выделяемых при парадигмальном анализе.

3. Базовая семантика

Рассмотрим более детально основные виды эффективно реализуемых функций.

(**V* → V+**) — методы вычислений. Функции таких семантических систем отображают некоторое число однотипных значений, возможно ни одного, в одно или несколько значений этого же типа. В некоторых ЯП такому классу может принадлежать до десятка семантических систем над разными типами значений. Соответствующее правило **R** обычно заключается в предварительном вычислении операндов **V*** с последующей выдачей **V+**.

(**T : A → V**) — средства доступа к памяти. Этот класс семантических систем характеризуется введением множества адресов **A** и специальной таблицы **T.** Соответствующее правило **R** позволяет по определённой (возможно неявно) таблице **T** поддерживать доступ к значениям. Существуют ЯП, в которых такая таблица допускает расширение и полиморфность РП. Обычно правило **R** предполагает предварительное вычисление операнда **V** при заданном **A**, хотя бывает, что и **A** можно вычислять.

{**F → {0,1}**} **—** особенности управления вычислениями. Этот класс семантических систем требует выделения одного или двух значений-переключателей **{0|1}.** Переключатели нужны для выбора ветви вычислений или обхода отдельных действий. В некоторых ЯП такие

значения реализованы как отдельный тип данных. Соответствующее правило **R** может использовать переключатели разного происхождения. Чаще всего - из ранее выполненных вычислений. В таком случае ход вычислений может зависеть от промежуточных результатов. Бывает и учёт независимых регистров. При выполнении функций **F** могут меняться значения переключателей.

(**E ↔ S**) — обратимая комплексация данных. Этот класс семантических систем разделяет множество значений **V** на элементарные (**E**) и составные (**S**). Во многих ЯП используется несколько таких семантических систем - разные структуры данных. Различие структур данных поддерживает разные дисциплины доступа к хранимым в памяти значениям. В некоторых ЯП допускается программирование структур данных. Чаще встречается конструирование из встроенных наиболее удобно реализуемых шаблонов. Соответствующее правило **R** должно быть подчинено определённой аксиоматике, гарантирующей наличие обратных функций в процессе комплексации. Встречаются иногда отсутствие обратных функций, обычно мотивируемое соображениями эффективности.

На уровне базовой семантики ЯП существенные различия видов функций семантических систем можно представить Таблицей 1.

Таблица 1

Семантическая декомпозиция минимального ядра ЯП

Уровень\Класс	***V***	***V* → V+***	***T : A → V***	***{F → {0,1}}***	***E ↔ S***
Ядро	Значение	Операции	Память	Управление	Вектор

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

Значение — минимальное представление объектов из области приложения языка.

Операции — минимальный комплект для обработки значений.

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

Управление — разметка выполнимости последовательности 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.

Различия классов семантических систем на уровне правил применения функций к значениям выражены таблицами 2-5.

Таблица 2

Семантическая декомпозиция макрорасширения ядра ЯП

Уровень\ Класс	***V***	***V* → V+***	***T : A → V***	***{F → {0,1}}***	***E ↔ S***
Ядро	Значение	Операции	Память	Управление	Вектор
Макро	Данное	Функции	Задание	Блоки	Стек

Макро — пополнение ядра средствами обработки представлений выражений, используемых с целью укрупнения любых конструкций. Укрупнение поддерживает консервативное расширение ЯП. Кроме того, оно способствует лаконизму текстов программ. Макротехника позволяет наследовать отлаженность компонентов программ. Бывает важным исключать дубли фрагментов текста, кода и структур данных. Простейший механизм макрогенерации обычно присутствует в СП как препроцессор. Так может быть 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 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 обработку памяти формально рассматривать как функцию с неявным аргументом, выполняющим роль таблицы 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.

Стек — схема организации данных, с определённой дисциплиной доступа для поддержки иерархии, возможно с защитой независимых блоков.

4. Семантика языков высокого уровня

Повышение уровня ЯП обеспечивается средствами укрупнения данных. Это приводит к понятию «иерархия» и оперированию блоками программы. Дальнейшее наращивание объёмов разрабатываемых программ отчасти достигается автоматизацией контроля некоторых условий корректности применения операций и функций к их операндам. Становятся важными понятия «предикат» и «тип переменных», удобно проверяемые при компиляции.

Таблица 3

Семантическая декомпозиция диагностического дополнения ядра ЯП

Уровень\ Класс	***V***	***V* → V+***	***T : A → V***	***{F → {0,1}}***	***E ↔ S***
Ядро	Значение	Операции	Память	Управление	Вектор
Макро	Данное	Функции	Задание	Блоки	Стек
Границы	Исключения	Предикаты	Типы переменных	Проверка логики	Вариант

Границы — методы проверки вычислимости выражений и выполнимости программ. Цель задания границ — снижение трудоёмкости отладки программ упрощением обнаружения ошибок. При отсутствии ошибок проверка воспринимается как накладные расходы. Встречаются механизмы установки ловушек на непредусмотренные ситуации и программирования обработки прерываний с возможностью продолжения вычислений.

Исключения — выбор специальных значений для разметки ненужных 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 его в памяти.

Проверка логики — соответствие типа данных или значений операциям обработки значений или доступа к памяти, возможно с учётом условий вычислимости.

Вариант — схема организации данных без определённой дисциплины доступа для организации перебора равноправных элементов или блоков. Необходимо как механизм удостоверения принципиальной выполнимости вычислений при частичной постановке задачи.

Таблица 4

Семантическая декомпозиция практичного обобщения ядра ЯП

Уровень\Класс	***V***	***V* → V+***	***T : A → V***	***{F → {0,1}}***	***E ↔ S***
Ядро	Значение	Операции	Память	Управление	Вектор
Макро	Данное	Функции	Задание	Блоки	Стек
Границы	Исключения	Предикаты	Типы переменных	Проверка логики	Вариант
Общность	Неопределённость	Мульти	Устройства	Отображение	Ввод-вывод

Общность — дополнительные средства обеспечения отладки и применении 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.

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

Устройства — механизм неявного расширения области действия операций и функций на устройства ввода-вывода, рассматриваемые как обобщение памяти.

Отображение — возможность регулярного применения функции к серии данных благодаря использованию представлений функций 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. 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.

5. Языки сверх высокого уровня

Существуют задачи с повышенными требованиями к качеству программ их решения. В частности, могут быть многочисленные важные ограничения на время и ресурсы, используемые при эксплуатации программ. Всё это обуславливает ряд трудно формализуемых уровней, здесь условно названных термином «надёжность». Часть решений на таких уровнях могут быть представлены средствами ЯП, другие поддерживаются

системой программирования или средой разработки программ, возможно допускающей использование разных ЯП, включая совместное применение библиотек.

Таблица 5

Семантическая декомпозиция высокопроизводительного расширения ЯП.

Уровень\Класс	***V***	***V* → V+***	***T : A → V***	***{F → {0,1}}***	***E ↔ S***
Ядро	Значение	Операции	Память	Управление	Вектор
Макро	Данное	Функции	Задание	Блоки	Стек
Границы	Исключения	Предикаты	Типы переменных	Проверка логики	Вариант
Общность	Неопределённость	Мульти	Устройства	Отображение	Ввод-вывод
Надёжность	Параметры обстановки и сети устройств	Отношения	Передача данных	Дисциплина обслуживания	Комплексы

Надёжность — дополнительные средства обеспечения качества программ при отладке и применении программ. Часть требований к качеству программ связано с применением программ на сетевом и многопроцессорном оборудовании. Чаще всего это поддержка параллельных вычислений, что весьма по разному влияет на особенности правил применения функций к значениям.

Параметры обстановки и сети устройств — вводятся специальные дополнительные значения и формы для представления конфигураций устройств. Такое расширение множества значений позволяет учитывать некоторые отдельные особенности процесса разработки отлаживаемых программ и практики их применения. Кроме того, появляется возможность учёта в программах динамики распределённых информационных систем.

Отношения — декларативное объявление контролируемых зависимостей. Существуют ЯП, защищающие сохранение таких объявленных условий.

Передача данных — механизм неявного расширения области действия операций над памятью на периферийные устройства. Такой механизм необходим для ЯП, нацеленных на поддержку параллельных вычислений и создание распределённых информационных систем. Обычно возникают средства работы с копиями данных, их репликации и клонирования.

Дисциплина обслуживания — возникают механизмы формирования пространств итераций и программирования методов доступа к разнородной памяти. Возможно запараллеливание тела цикла или витков рекурсивной функции на разные устройства - процессоры. Такие пространства поддерживают многопоточность или многопроцессорность в случае слабых зависимостей между соседними итерациями. При оптимизации программ возможно сведение рекурсии к циклу, а для верификации часто требуется обратное.

Комплексы — структура данных, допускающая разметку и кратность вхождения составляющих. Такая структура используется как средство повышения эксплуатационных

характеристик программы. Комплексы позволяют представлять тиражирование готового или подобного конструктива без избыточного копирования.

6. Переход к системам программирования

Реализация каждой из семантических систем, выделенных из определения ЯП, может быть выполнена разными методами. От методов требуется сохранение достаточной автономии систем и их соответствие спецификации. Спецификация в свою очередь должна быть согласована с правилом R, определяющим класс семантической системы. Кроме того, значения при разработке СП обретают поливариантность используемых представлений. По меньшей мере используются одновременно тексты, структуры и коды с полным спектром переходов от одного представления к другому. В результате удаётся смягчать избыточную сложность отладки программ ценой повышения трудоёмкости разработки СП. Возможна верифицируемая комбинаторика семантических систем по мере улучшении СП. При создании СП число реализуемых промежуточных семантических систем резко возрастает. Это зависит от сложности задач системной поддержки внутренних форматов данных. Дополнительные сложности вносит оптимизация программ. Ещё сложнее решение проблем производительности вычислений. Для долгоживущих программ свой вклад даёт и факторизации компонентов, приспособленных к многократному применению в разных СП. Возникает ряд дополнительных уровней, характеризуемых своими наборами понятий:

Окружение — операционная система, поддерживающая реальные процессы, оперируя понятиями Объекты, Процессор и Устройства, Действия, Файлы, Процесс, Конфигурация.

Реализация — процесс или результат создания исполнимого кода программы, оперируя понятиями Шаблоны кода, Генерация программы, Атрибуты, Линейные участки (SSA-формы[1]), Размеченное множество.

Оптимизация — приведение программы к форме, обладающей заданными свойствами, оперируя понятиями Критерии, Преобразования, Эквиваленты, Маршрут, Орграф.

Факторизация — приведение программы к декомпозированной форме, части которой обладают определённой автономностью при улучшении, исполнении или применении программ. Оперирует понятиями Комплекты, Процессоры, Библиотеки, Потоки, Сеть.

Производительность — приведение программы к форме, дающей более высокую скорость исполнения. Оперирует понятиями Время, Синхронизация, Сложное действие, Барьеры, Синхросеть.

1 SSA-формы - линейные участки с однократным присваиванием. Играют важную роль в методах оптимизации программ.

Каждый такой уровень включает в себя полный образ реализуемого ЯП. Это пропорционально увеличивает понятийную сложность разработки СП. Сравнение систем понятий уровня ЯП и СП показывает значительное наращивание дистанции между программированием и системным программированием, что более подробно рассмотрено в [2].

7. Классификатор парадигм и стилей программирования

Как правило под парадигмами понимают особенности мышления, характерные для выбора способа решения некоторого класса задач. О парадигмах программирования начали говорить с середины 1970-ых годов при осознании кризиса технологии программирования. Вскоре внимание от этой темы отвлеклось на задачи освоения новой элементной базы. Понятие «парадигма программирования» пока не имеет строгого определения, хотя термин стал модным и активно используется при объявлении новых способов работы с программами.

Возникает вопрос: в какой мере анонсы различных ПП реально отражают разные особенности мышления? Далее: как оценить, насколько новая парадигма отличается от уже имеющихся? Ответы на эти вопросы выведены из результатов анализа и сравнения наиболее популярных основных и фундаментальных парадигм программирования, поддержанных в большом числе ЯП. Рассмотрены особенности и многих производных парадигм [10,11], сводимых к композиции основных и фундаментальных. Кроме того, показана зависимость успешного применения ПП от фазы интервальной схемы процесса разработки программ. Схема успешной разработки отражает уровень изученности решаемых задач. Практичность результата разработки обуславливается прагматикой традиционных решений реализации базовой семантики. Эффективность таких решений зависит от подразумеваемых аппаратных средств.

Программирование решения любой непростой задачи происходит в определённой схеме поддержки жизненного цикла программ. Эксплуатация программы нередко ориентирована на развитие широкого спектра аппаратуры и расширение категорий пользователей. По этой причине многие долгоживущие и новые ЯП поддерживают сразу несколько различных парадигм. Мультипарадигмальность может быть представлена разными пропорциями в поддержке отдельных парадигм. Для определения таких пропорций нужна парадигмальная декомпозиция определений ЯСП. Прежде всего пропорции связаны с оценкой уровня изученности решаемых задач. К оценке примыкают требования ограничения новизны в программистских проектах для прогнозирования времени жизни проекта. Эти два параметра позволяют корректно обосновывать выбор средств и методов разработки программ, включая

выбор ЯСП. Материал для такого обоснования дают известные прагматические аспекты истории ЯСП. Они представлены фактами развития парадигм программирования.

Особое место среди парадигм занимает функциональное программирование (ФП). Прежде всего с этой парадигмой связан лаконичный стиль представления программ. Процесс ФП нацелен на предельно общие методы решения задач. Общность даёт представление универсальных функций, допускающих полный контроль правильности вычислений.

В результате формируется потенциал развиваемых средств информационной обработки. При пошаговой технологии разработки программ ФП может использоваться как инструмент прототипирования других парадигм. Это образует основу функционального подхода к описанию парадигм программирования и ЯСП. Реально ФП способно выполнять роль парадигмального моделирования новых информационных систем. Это образует полноценный полигон для языкотворчества. Первое применение такого полигона находится в области проблемно-ориентированных ЯСП. Основы ФП исторически привлекают внимание специалистов при каждой смене элементной базы. Другая линия - неожиданное расширение сферы применения ИТ.

Очередной этап надежд на ФП связан с актуальностью проблем параллельных вычислений (ПВ). Многие попытки классификации парадигм программирования не рискуют признать ПВ самостоятельной парадигмой. Чаще встречается включение ПВ в общеизвестные парадигмы в виде расширений, дополнительных библиотек, указаний компилятору или специальных механизмов. Обычно встраиваются в ЯП отдельные средства без претензий на общее решение проблем ПВ. Многообразие моделей взаимодействующих процессов ждёт своего решения уже более 70-ти лет. Ряд проблем проявилось ещё в докомпьютерную эпоху. Всё это говорит о сложности проблемы. Не исключено, что решение проблем ПВ слишком чувствительно к методам обучения, зависит от образовательных механизмов. Это повышает требования к определению парадигмы параллельного программирования и поддерживающих её ЯСП. Предстоит дальнейшее исследование парадигм параллельного программирования. Важно достичь возможности привлечения разных моделей взаимодействия программируемых процессов. Нужна опора на языки лаконичного представления программ. Растёт актуальность создания СП, обеспечивающих не только тестирование, но и расширяемый спектр средств отладки и верификации программ. Это может дать техническую основу для подготовки производительных программных систем.

В этом плане поучителен опыт разработки и развития долгоживущих ЯСП. Характерно внимание проблемам отладки и расширения программ. Долгоживущи СП необходимо обладают мультипарадигмальностью даже для изночально монопарадигмальных ЯП (Haskell,

F#). Такое расширение позволяет успешно применять СП на протяжении полного жизненный цикла программ. В этом плане жизненный цикл программ для ПВ обладает большей длительностью, выходящей за пределы основных ПП.

Обычно парадигмальные различия проявляются на всех уровнях определения ЯСП (лексика, синтаксис, семантика, прагматика). Особенно чётко видна парадигмальная характеристика на уровне прагматики. Именно эта разница служит основанием для классификации парадигм программирования и отделения их от подходов, стилей, методов и т. п.

8. Представление относительной семантики ЯСП

Подход к сравнению и классификации ЯСП осложнён возрастанием объёмов их описаний и сложности реализаций. Первое формальное описание языка Pure Lisp занимало примерно полторы страницы. На них было представлено около десяти понятий. Для их освоения было достаточно около 250-ти упражнений. В сравнении с этим, описание нового стандарта C++ занимает примерно 1300 страниц. Формальное представление грамматики – 32 страницы. Используется определение 280-ти понятий. Для отладки компилятора C++ предлагается более 6000 тестов.

Реализация Lisp-интерпретатора поддерживала встроенные средства компиляции программируемых функций. Это обеспечивало гладкий переход от отладки программ к их эффективному применению. Некоторые диалекты Lisp-а и сферы их приложения выполнили роль прототипов для создания новых ИТ не только в области искусственного интеллекта. Язык использовался как средство проектирования, прототипирования, символьной обработки, исследования методов оптимизации и преобразования программ, гипертекстов и многого другого. Первые реализации языка Lisp в СССР показали высокую моделирующую силу ФП. Она оказалась достаточной для решения сложных задач в области верификации программ. На языке Lisp были проведены исследования доказательных построений (и проводятся в настоящее время [12]) и решения задач химии, биологии и языкознания. До сих пор многие из таких задач рассматриваются как весьма сложные. Дальнейшее усложнение многих классов задач связано с переходом к большеобъёмным массивам. Доминирование парадигмы императивного программирования успешно поддержало эффективные решения хорошо изученных задач по ранее отлаженным алгоритмам. При переходе к большеобъёмным массивам эта парадигма слабо приспособлена к разработке новых методов, требующих заметного объёма отладки на фоне распределённых систем. Это провоцирует массовое языкотворчество в сфере новых и экспериментальных постановок задач. Такие

задачи неустранимо обладают исследовательским компонентом, требующим более мощных и более общих парадигм.

Создание новых проблемно-ориентированных ЯП выполняет работу по компенсации недостаточности основных ПП для новых задач. Такую работу пытались и до сих пор пытаются выполнить на базе парадигмы объектно-ориентированного программирования. Большие надежды возлагались на разработку и стандартизацию проблемно-ориентированных интерфейсов. Приходится констатировать, что формально полезные абстракции на практике редко получают удачное конкретное наполнение и применение. Наследование отлаженных программистских решений обычно происходит на уровне библиотечных модулей или инструментов в СП. Строгой формализации чаще предпочитают успешное применение редактируемых образцов программ и многократно используемых шаблонов языковых конструкций. Это позволяет определения новых ЯП формулировать относительно известных ЯСП. Для этого нужны средства определений в стиле описания границ сходства и различия. Такие границы можно описывать в форме сопоставления парадигмальных моделей. Парадигмальные модели позволяют таким образом выделять автономно развиваемые модули. Взаимонезависимость парадигм удобна для синтеза СП из типовых программных компонентов, поддерживающих отдельную парадигму. Модули, соответствующие отдельным семантическим системам, могут иметь разные конкретизации для различных парадигм. Сгруппированные для поддержки разных ПП модули при такой декомпозиции могут обладать взаимозаменяемостью при смене парадигмы. Накопление таких модулей образует технологическую основу для экспериментальной разработки новых ЯП, определения которых должны быть декомпозированы для пошагового определения СП.

Программные инструменты для создания новых ЯСП могут использовать относительное определение семантики со ссылками на известные ЯП. Достаточно констатировать, что вычисления организованы как в языке Pascal, работа с памятью как в языке Lisp, управление вычислениями как в языке C#, структуры данных как в языке APL. Формально это напоминает технику теоретических работ на уровне свободно интерпретируемых схем программ. Плодотворность такого подхода хорошо видна на разнообразии схем, подобных сетям Петри. Чёткая формулировка относительной семантики позволит использовать банк улучшаемых верифицируемых компонентов информационных систем. Учитывая сложность решения вопросов верификации для обеспечения надёжности и безопасности программ, именно такой подход может обеспечить многократность использования результатов, оправдывающую трудозатраты. Ряд методов конструирования учебных систем программирования также может повысить эффективность экспериментального

программирования на базе банка типовых компонентов ЯСП. Возможность апеллировать к известным ЯСП может гарантировать быстрое ознакомление с новыми средствами создаваемых ЯП.

Проблема автоматизации ПВ — одно из важнейших направлений современных исследований в области ИТ. Стремительный прогресс современного оборудования опережает развитие методов организации высокопроизводительной информационной обработки данных. Развитие моделей параллелизма в языках высокого уровня характеризуется значительным разнообразием. Такое разнообразие трудно поддержать в рамках одного ЯП. Поэтому от современной системы ПВ требуется не только лаконизм понятного представления программ. Нужна ещё и конструктивность высокоуровневых определений, допускающих совмещение разноязыковых средств. Кроме того, заметные проблемы связаны с расширяемостью спектра используемых архитектур. В целом, на пути к автоматизации ПВ возникают существенные затруднения. Часть затруднений мало зависит от результатов фундаментальных исследований. Многое обусловлено формированием технологий и человеческим фактором.

Часть трудностей может быть смягчена активизацией применения программируемых методов преобразования программ. Условно это можно назвать суперпрограммированием. Имеются прецеденты работ такого направления. Многое может дать подход на основе трансформационно-операционной семантики языков параллельного программирования. В целом всё это приводит к актуальности задач изменения подходов к организации СП. Нужны информационно-инструментальные средства не только встроенного в компиляторы, но и программируемого анализа и преобразования программ. Такие средства требуют сопровождения и конструирования распределенной измерительной среды для экспериментальных исследований производительности информационных систем. Измерительная среда даёт аргументы для пополнения корпуса отлаженных программ и тестовой базы. К этому примыкает задача конструирования уточняемых функций при разработке программ. Технически это похоже на перегрузку операций в ООП. Полезна возможность уточнение начинать макетированием программ с помощью тестов, спецификаций и описаний, что несколько перекликается с идеями обработки недоопределённой информации.

Анализ парадигмальных особенностей и тенденций развития ЯСП приводит к ряду важных выводов. Прежде всего, возрастание объёма определений ЯП вызвано в значительной мере попытками поддержать в рамках одного языка все средства, необходимые в полном жизненном цикле программ. Рост объёмов определений можно нормализовать

методикой относительных описаний, используя предварительные знания основных ПП. Пропорции между разными парадигмами допускают отладку средствами ФП. Достаточно развиты средства синтаксически управляемого синтеза определений, позволяющие от средств ФП автоматически переходить к любому конкретному синтаксису. Гибкость таких средств может быть повышена техникой сопоставления с образцами и другими методами. Достаточно важно, что решение проблем ПВ требует поддержки своих парадигм, выходящих за пределы удобного применения основных. В любом случае, независимо от ПП, обеспечение надёжности и безопасности ПО требует пересмотра традиционных решений о возможностях СП. Решение таких, трудно формализуемых, проблем связано с образовательными аспектами гуманитарной составляющей информатики. Особого внимания требует зависимость практики программирования от имплицитного научения. Здесь имеет место проявление интуиции, инсайта и неявного знания. Это также даёт предпосылки для пересмотра структуры СП на современном оборудовании и мощных возможностях ИТ.

9. Эксплуатационная прагматика базовых ПП

Рассматривая полный жизненный цикл программ как ряд схем решения задач с помощью определения, разработки и отладки программируемых решений, можно обратить внимание на методы пошаговой или непрерывной разработки экспериментальных и развивающихся программных систем, позволяющие ряд начальных версий создавать техникой макетирования. Само понятие макета допускает постепенное уточнение прототипов программ. Определение функционирования таких уточняемых прототипов можно выполнять методом вертикального слоения программ А.Л. Фуксмана [5,6]. Концептуально этот метод является предшественником аспектно-ориентированного программирования. Отдельные слои могут быть подвергнуты семантической и парадигмальной декомпозиции. Это обеспечивает использование резервов синтаксически управляемого конструирования обработчиков символьной информации [2]. Такие резервы позволяют поддержать непрерывную улучшаемость системы по мере её отладки и внедрения. В процессе улучшения появляется возможность выделять типовые компоненты, пригодные для включения в разные системы. Компоненты систем обладают общими семантическими подсистемами, зависящими от реализационной и эксплуатационной прагматик.

Основные ПП обладают на уровне эксплуатационной прагматики достаточно чёткими различиями. Такие различия как схема программы, формат элементов программы, особенности процесса выполнения программы, граница между правильностью программы и отказом в продолжении выполнения. Для решения задач, обладающих стабильной

постановкой, характерны парадигмы функционального и императивно-процедурного программирования. Решение подверженных развитию задач естественнее выражается в парадигме логического и объектно-ориентированно программирования. При переходе к особо сложным постановкам задач, решение которых требует коллективных усилий и многопроцессорных комплексов, возникает необходимость в парадигмах параллельного программирования. Постановка задачи обуславливает выбор решений в таком трёхмерном парадигмальном пространстве.

Функциональное программирование - программа представляется деревом, вершины которого функции. При обходе дерева функции поочерёдно взаимодействуют с аргументами. Число и типы аргументов должны соответствовать определению функции. Аргументами могут быть фрагменты дерева. Это позволяет организовывать локальное изменение дерева. Результатом программы считается результат последней вычисленной функции. Функция может быть применена к аргументам или дать отказ. В последнем случае обход дерева прерывается и программа признаётся невыполнимой. Программисту при отладке следует изменить программу так, чтобы обход дерева мог быть завершён.

Императивно-процедурное программирование - программа представляется размеченной последовательностью необходимых команд над общей памятью. Порядок выполнения команд подчинён императивной логике управления, задающей последовательность, начинающуюся с известной стартовой метки. Исполнение команды заключается в изменении значения некоторых регистров памяти. Очередная команда исполняется немедленно. Отсутствие учёта и проверки каких-либо соответствий или условий готовности даёт экономию времени исполнения. Результатом программы считаются значения ряда регистров, получившиеся по завершении программы. Неуспешное исполнение команды приводит к передаче управления обработчику прерываний. Обработчик может продолжить выполнение программы. Как правило до исполнения программы, на этапе компиляции, выполняется анализ исполнимости команд. Возможно некоторое изменение их последовательности из соображений оптимизации. Это приводит к определённому расширению пространства допустимых решений, возможно не известных пользователю. Выполнение программы завершает исполнение специальной команды или отсутствие очередной команды. Программист при отладке обычно доводит программу к форме, не требующей обработки прерываний и завершаемой для определённости специальной командой.

Логическое программирование - программа представляется рядом "равноправных" вариантов. Любой вариант может выполниться успешно или дать отказ, влекущий выполнение другого варианта. Программа признаётся невыполнимой только если нет ни

одного успешно выполняющегося варианта. В случае неуспеха программисту при отладке следует добавить варианты, чтобы хотя бы один был успешным. Результатом программы считается результат любого из успешных вариантов или их объединение. Пространство допустимых процессов расширяется выполнением неуспешных вариантов, результаты которых не сказываются на окончательном результате.

Объектно-ориентированное программирование - программа представляется иерархией классов объектов. Класс содержит описания методов обработки объектов и спецификацию прав доступа методов к элементам объектов. Выполнение программы начинается с головной функции. Вызов функции сводится к применению методов к объектам при соответствии прав доступа. Пространство допустимых процессов включает кодирование специальных сигнатур для проверки соответствия метода и объекта. Это позволяет исключить перебор при выборе одного из определений метода, функции или операции, обладающих внешне одинаковыми представлениями. При несоответствии программист может при отладке изменить спецификацию прав доступа, перегрузить некоторые определения, дополнить или перестроить иерархию классов, чтобы нужный метод мог быть применён к объекту.

Переход к параллельным вычислениям заставляет программы представлять сетью, узлы которой - действия или данные. Действия выполнимы при определённых условиях готовности. При обходе сети выполняются готовые действия, возможно без ожидания завершения смежных действий. По существу на практике различаются два подхода к параллелизму. Асинхронный подход ориентирован на независимые потоки действий, возможно обменивающиеся сообщениями. Другой подход ориентирован на организацию взаимодействия процессов над общей памятью, допускающих императивную синхронизацию на уровне аппаратуры.

При асинхронном подходе отказ при выполнении некоторого действия не препятствует выполнению других действий, кроме него самого, что расширяет пространство допустимых процессов в сравнении с ФП. При выполнении одних действий могут меняться условия готовности других. Результатом программы считаются результаты ряда необходимых действий. Обход сети прерывается при отсутствии готовых к исполнению действий. Программа признаётся невыполнимой, если не выполнены отдельные необходимые действия. Программисту при отладке следует изменить программу так, чтобы обход сети обеспечивал выполнение необходимых действий при любом порядке обхода сети.

Взаимодействие процессов над общей памятью допускает императивную синхронизацию на уровне аппаратуры. Это может требовать более жёстких границ выполнимости очередного действия. Для императивной синхронизации процессов программа представляется сетью,

узлы которой — вилкообразные разветвления линейных участков, выполнимых при достижении разветвления. При обходе сети действия линейного участка выполняются неимперативно, т. е., возможно без непрерывности и ожидания завершения смежных действий. Отказ при выполнении некоторого действия не препятствует выполнению других действий и других участков, но текущий участок может быть помечен как дефектный, что расширяет пространство допустимых процессов. Программа завершается по мере выполнения всех бездефектных участков или признаётся невыполнимой, если нет успешно продолжаемых участков. Результатом программы считаются равноправные результаты всех участков. Программисту при отладке следует изменять программу так, чтобы обход сети обеспечивал выполнение необходимых участков при любых дефектах обхода сети.

10. Методика учебного программирования

Последние годы характеризуются всплеском интереса к созданию новых проблемно-ориентированных языков программирования. Это порождают заметно расширяющийся спектр проблем образовательного и реализационного характера. Парадигмальные модели языков и систем программирования могут быть полезны при создании классификатора для парадигмальной характеристики языков программирования. Такая классификация нацелена на стратификацию представления особенностей семантики языков и систем программирования на автономно развиваемые компоненты. Выделение компонентов в процессе пошаговой разработки экспериментальных систем программирования можно связать с формированием схем изучения и преподавания системного программирования. Это поспособствует снижению трудоёмкости разработки новых языков и систем программирования.

В разных источниках упоминается от 20-ти до 70-ти парадигм программирования. Список парадигм видоизменяется и расширяется в зависимости от актуальности тех или иных проблем программирования. Влияет и мода на особенности популярных ИТ. Бум языкотворчества в области проблемно-ориентированных ЯП показывает важность парадигмальных различий в определениях языков при выборе инструментов для практических работ. Успех выбора зависит от выработки рекомендаций по предпочтению инструментальных средств при создании новых программных систем и развитии ИТ. Разработка методов анализа определений ЯСП с целью установления их парадигмальной характеристики может дать подходящую основу для таких рекомендаций.

Практичность целей требует уточнить определение термина «парадигмы программирования». Необходимо отличать парадигмы от подходов, стилей, дисциплин,

техник, технологий, методов, методик, методологий программирования. Затем следует выразить отношения между базовыми и производными парадигмами. При выборе типовых компонентов для конструирования новых ЯП можно учесть историю появления наиболее популярных парадигм. Это даёт возможность уяснить причины и особенности их успешного применения на разных категориях задач разработки программных систем. Работа со сложными конструкциями требует методики лаконичного представления определений ЯСП. Лаконизм позволяет видеть парадигмальные характеристики с общих позиций. Всё это даёт возможность вывода системы обучения программированию. Такая система должна включать методы преподавания программирования на базе парадигмальных моделей. Ключевое значение имеют парадигмальные модели, представимые средствами функционального программирования. Это парадигма обладает высокой моделирующей силой. Лаконизм и понимаемость парадигмальных моделей может быть достаточной для описания учебных языков при обучении параллельным вычислениям. Такие модели пригодны для выработки ясных рекомендаций по технологической схеме разработки учебных материалов по программированию [8,9].

Общепринята практика экономить время подготовки специалистов погружением их в современный производственный инструментарий. Такая практика, минуя основы ради быстрого выхода в производство, закономерно приводит к провалам при быстрой смене технологий. Учебные языки программирования позволяют в лаконичной форме понять базовые ПП. В перспективе можно показать пути их развития в ближайшем будущем. Именно учебные языки и системы программирования призваны смягчать образовательные проблемы развития ИТ. В конце 1950-ых годов прецедент такого учебного языка был создан как введение в программирование на языке Lisp через изучение его подмножества Pure Lisp. Так была решена задача привлечения энтузиастов ради развёртывания исследований в области искусственного интеллекта. И сейчас основные идеи этого уровня используются в развитии теории автоматизации доказательств [12]. Есть основания решать трудные проблемы организации параллельных вычислений начиная с проблемы начального ознакомления с миром параллелизма. Это можно бы сделать на материале робототехники в стиле Lego-Logo, смягчив зависимость от физического материала использованием современных средств компьютерной графики.

Разработка учебных СП и игровых тренажёров – прекрасный полигон для экспериментов по совершенствованию и изучению техники компиляции программ. Материал для постановки учебных задач даёт создание новых проблемно-ориентированных ЯП. В этом плане перспективна разработка факультативного курса «Учебная информатика». Такой курс

непременно должен включать в себя ознакомление с основными явлениями параллелизма. Введение в параллелизм для школьников может дать курс «Начала параллелизма» с практикумом на базе языка начального обучения. Важно как можно раньше проявить и помочь осознать интуитивные способности понимать и прогнозировать взаимодействия процессов. Не трудно видеть, что с такими задачами в обычной жизни реально справляется большинство.

На базе опыта Новосибирских Школ юных программистов специально разработан язык СИНХРО для раннего ознакомления с миром параллелизма [3]. Этот язык даёт основные представления о технике программирования взаимодействующих процессов и методах конструирования программ. Представление данных обеспечивает возможность чёткого разделения процессов хранения данных и вычисления хранимых значений. Язык содержит средства макрогенерации, обогащённой управляемым контролем синтаксической и семантической правильности текстов программ. Ядро языка содержит механизм синхронизации процессов, включая согласование итерирования и рекурсии в терминах барьеров. Предполагается, что полноценное овладение современным миром ИТ более надёжно в форме игр на базе учебных языков и СП и проблемно-ориентированных игровых тренажёров. Созрела необходимость пересмотра методов организации и производственных СП с целью рационального использования возможностей ИТ, таких как видео, цвет, геометрия, шрифты и игры.

Эти вопросы проявляются на фоне быстрой эволюции технических средств и расширения сферы их применения. Решение ряда проблем осложнено обилием и разнообразием доступных весьма противоречивых учебных материалов. Это обостряет проблему достоверности представленного знания. К счастью теперь массово доступны используемые для самообучения и взаимообучения механизмы Интернет-поисковиков. Это даёт возможность выявления наиболее значимых позиций в своде информационных материалов.

11. Анализ схем изучения и преподавания ЯСП

Инвариант обучения программированию подразумевает некоторую математическую преамбулу. Обычно это даётся как обзор основных разделов классической дискретной математики. Многие понятия математики стали общепринятым аппаратом описания компьютерного мира, формальным языком постановки задач и методов их алгоритмического решения. За редким исключением не придано значение моделям непрерывной и неклассической математики. Разнообразие аксиоматических теорий и многозначных логик слабо представлены в программах обучения программистов. Тем не менее, абстрактные

механизмы имеющие место в реальной практике программирования и реализации ЯСП, находят своё воплощение в форме ПП. Есть достаточные основания для формирования более широких учебных программ по прикладной математике информационных систем, особенно для специализации по системному программированию. Классификация фундаментальных парадигм программирования отражает именно их образовательное значение. При анализе ПП с этой точки зрения отмечается важность разделённости ПП в определениях разных ЯП.

Не менее важна гуманитарная составляющая процессов разработки программ. Роль человеческого фактора имеет яркие проявления в истории информатики. Начиная с первых идей автоматизации вычислений и до наших дней можно привести многочисленные примеры индивидуального вклада отдельных личностей и коллективов в становление информатики и программирования. Движущие силы информатики тесно связаны с яркими лидерами, их способностями, индивидуальностью, характерами и образованием. Это всё в свою очередь базируется на природных механизмах имплицитного научения. Таким образом, результаты исследований в области этологии человека могут рассматриваться как исходный ориентир образовательной информатики. Именно такой ориентир, обуславливающий гуманитарные аспекты программирования и программистского образования, является важнейшим фактором успешной практики в программировании. Возможно, так устроена любая ноаая отрасль науки и производства. Молодость программирования как науки и технологии ещё не позволила переварить значительный объём имплицитного знания, не перевела его на удобный вербальный уровень. Потому программирование сохраняет зависимость от латентного обучения и организации специальных учебных практикумов. Общедоступность мощных информационных ресурсов резко повышает актуальность внимания к гуманитарным аспектам программистского образования. Раннее введение в учебный мир программирования на материале робототехники способно быстро довести до решения проблем достоверности знаний, представленных в сетях даже в форме большеобъёмных массивов.

Упорядочение учебно-методических материалов наиболее перспективно в рамках функционального подхода к системному представлению прикладных программ учебного назначения. Такой подход даёт методику лаконичного представления результатов парадигмального анализа схем изучения ЯСП. Особенно важно отследить схему практических занятий по подготовке и отладке программ. Существенно наладить работу в разных контекстах или на разных комплексах. Практика на базе производственных СП обычно отвлекает от понимания базовых возможностей ядра ЯСП. Именно специальный компьютерный практикум призван сформировать и проявить интуитивную базу программирования на примере работы с учебными макетами и учебными ЯП. Требуется

комплект учебных ЯП полноценного освоения разных парадигм программирования. Комплект ЯП должен быть достаточным для понимания типовых проблем и методов разработки программ.

Практика обучения обычно дополняется системой предвузовской подготовки талантливой молодежи для сознательного выбора профессии. Специализация в области программирования и информационных технологий в этом не исключение. Проблемой является акцент на активизацию творческих способностей, противоречащий массовому убеждению в доступности решений почти всех задач. Включение осознания инсайтов в процессе получения новых знаний затруднено чрезмерным богатством доступных знаний, уровень достоверности которых требует проверки. В принципе, такое положение дел может давать пищу для микроотркытий.

Результативность такого подхода была объективно показана историей Новосибирских Школ юных программистов. Выпускники этой Школы создали первый комплект прикладных программ автоматизации школьного учебного процесса «Школьница». Примерно такая дополнительная образовательная линия теперь фактически существует в форме Интернет-доступа к информационным материалам, дистанционных школ, консультаций, конференций и конкурсов. Причём, не только по программированию, но и по другим видам творчества.

Преподавание парадигм параллельного программирования в университетских образовательных программах и специализации по ИТ уже получило признание в мировой практике. Осознано её образовательное значение, но методические и образовательные проблемы ПВ ещё не преодолены. Так же не вполне осознан образовательный аспект фундаментальных и основных ПП. Слабо представлен полный спектр ПП в профессиональной подготовке информатиков. Для специализации информатиков изучение ПП должно быть чем-то вроде таблицы Менделеева в химии или алгебры в математике. На уровне высшего образования уже накоплен опыт преподавания ПП бакалаврам и магистрантам. Пора учесть, что результативность обучения практическому программированию может быть повышена активизацией парадигмального подхода к формированию программ обучения. ФП даёт лаконичные формы для достаточно глубокого понимания парадигмальных различий изучаемых ЯСП. Оно полезно и для формирования интуиции, при условии, что инвариант обучения программированию содержит гуманитарный аспект. Функциональные модели дают систематизированное представление основных и фундаментальных ПП, включая имплицитное знание. Важный материал даёт история программирования, информатики и вычислительной техники, особенно рассказы участников и очевидцев, дополненные результатами исследований по истории науки.

12. Заключение

Парадигмы программирования по существу различаются на уровне прагматики и интуитивных моделей понимания решаемых задач. Они определяют компромисс между трудоёмкостью разработки программ и производительностью их применения. Выбор парадигмы зависит от ряда характеристик постановки решаемой задачи. Такие параметры как уровень изученности задач, ранг абстрагирования понятий, степень организованности структур данных, полнота отлаженности алгоритмов и программ позволяют систематизировать свод исторически сложившихся парадигм программирования. Парадигмальный анализ постановки задачи позволяет представить проект программы в виде композиции из парадигмальных моделей. Такие модели допускают реализацию в виде автономно развиваемых программных модулей. Именно это способствует их многократному использованию в разных программных комплексах.

Изучение парадигм программирования начинается с проявления интуитивных представлений о взаимодействии процессов. Отладка такого взаимодействия может сопровождаться образцами методов разработки программ и стилей программирования. Разнообразие стилей и методов доступно как корпус свободно распространяемых программ и свод учебно-методических материалов. Основное препятствие - трудоёмкость систематизации этого обширного пространства, что делает актуальной задачу автоматизации парадигмального анализа программ.

Представление результатов парадигмального анализа возможно в рамках экспертной системы при условии выбора подходящих средств визуализации схем понятий, некоторые из возможных приёмов визуализации уже намечены и рассмотрены [2]. Инструментарий автоматизации парадигмального анализа можно создать как взаимодействие экспертной системы с гибким комплексом средств сопоставления программ с разными моделями, подобными системам верификации программ или проверки доказательств теорем. Такие системы уже пригодны для анализа нормализованных форм программ в стиле функционального программирования на языке Haskell.

Список литературы

1. Городняя, Л.В. Парадигмы программирования: анализ и сравнение / Из-во СО РАН, РФФИ 17-11-00042. 216 с.
2. Городняя Л.В. Пространство решений в языках и системах программирования. //Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17-22 сентября 2018

г., г. Новороссийск). -- М.: ИПМ им. М.В.Келдыша, 2018. - [Электронный ресурс]. URL: http://keldysh.ru/abrau/2018/theses/46.pdf (дата обращения 06.09.2018).

3. Городняя Л.В. Язык параллельного программирования СИНХРО, предназначенный для обучения. Новосибирск, ИСИ СО РАН, 30 с. (Препринт / ИСИ СО РАН; № 180) [Электронный ресурс]. URL: https://www.iis.nsk.su/files/preprints/gorodnyaya_180.pdf (дата обращения 06.09.2018).
4. Лавров С.С. Методы задания семантики языков программирования / Программирование, 1978. N 6. С. 3-10.
5. Михалкович С.С., Налбандян Ю.С. Адольф Львович Фуксман – математик и программист. [Электронный ресурс]. URL: http://www.sorucom.org/pdf/SORUCOM_2017.pdf (дата обращения 06.09.2018), [Электронный ресурс]. URL: http://plc.sfedu.ru/files/slides/day-1/Mikhalkovich.pdf (дата обращения 06.09.2018).
6. Фуксман А.Л. Технологические аспекты создания программных систем - М. : Статистика, 1979. – 183 с.
7. ACM SIGPLAN conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH) [Электронный ресурс]. URL: *https://2017.splashcon.org/home* (дата обращения 06.09.2018).
8. Gorodnyaya L.V., Andreyeva T.A. Programming paradigms in higher education. // Bulletin of the Novosibirsk Computing center. Series: Computer science. – № 38. – 2015. – pp. 67-90.
9. Gorodniaia L., Andreyeva T.. Study of Programming Paradigms. // 10th International Technology, Education and Development Conference. 7-9 March, 2016, Valencia, Spain. //INTED2016 Proceedings. – 2016. – pp. 7482-7491. ISBN: 978-84-608-5617-7 ISSN: 2340-1079 doi: 10.21125/inted.2016.0768
10. Peter Van Roy (2009-05-12). "Programming Paradigms for Dummies: What Every Programmer Should Know" (PDF). info.ucl.ac.be. Retrieved 2018-01-22.
11. Peter Van Roy. Сайт с материалами по системе Mozart, поддерживающей учебный мульти-парадигмальный язык программирования Oz. [Электронный ресурс]. URL: http://sourceforge.net/projects/mozart-oz/ (дата обращения 06.09.2018)
12. 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.

Приложение 1

Аббревиатуры и обозначения

ПВ — параллельные вычисления.

ПП — парадигмы программирования.

РП — реализационная прагматика.

СП — система программирования.

ФП — функциональное программирование.

ЯП — язык программирования.

ЯСП — языки и системы программирования.

A — адреса в памяти.

E — элементарные значения.

F — функции или операции.

R — правило применения операций к значениям.

S — составные или конструируемые значения.

T — таблица соответствия адресов хранимым значениям.

V — значения.

V* — любое число значений, возможно ни одного.

V+ — любое число значений, хотя бы одно должно быть.УДК 004