УДК 004.4

Н.С. НИКИТЧЕНКО

КОМПОЗИЦИОННО-НОМИНАТИВНЫЕ АСПЕКТЫ АДРЕСНОГО ПРОГРАММИРОВАНИЯ

Ключевые слова: *теория программирования, адресное программирование, композиционное программирование, композиционно-номинативное программирование, принцип адресности, принцип программного управления, семантика программ, композиции, структуры данных, номинативные данные.*

ВВЕДЕНИЕ

Екатерина Логвиновна Ющенко известна во всем мире как «Леди Лавлейс Советского Союза». Эта высокая характеристика утверждает статус Екатерины Логвиновны как одного из основателей программирования на просторах бывшей сверхдержавы. Автор статьи считает себя прямым и опосредованным учеником Екатерины Логвиновны. Прямым — потому что во время учебы в Киевском государственном университете имени Тараса Шевченко слушал ее лекции, а потом — ее выступления на конференциях и семинарах, опосредованным — потому что был приобщен к научным идеям Екатерины Логвиновны также благодаря Владимиру Никифоровичу Редько, который работал с Екатериной Логвиновной в ее отделе Института кибернетики АН УССР. Все, кто знал Екатерину Логвиновну, ощущал ее научное и личностное влияние.

Научные идеи, заложенные Екатериной Логвиновной более чем полвека назад, не только не устарели, но актуальны и сейчас, более того, приобретают новое звучание в современных исследованиях проблем программирования. Цель статьи — проанализировать принципы адресного программирования (АП) [1–5], одним из авторов которого была Екатерина Логвиновна, и их дальнейшее развитие.

Адресное программирование достаточно детально изложено в [1]. Фактически это было первое пособие по программированию для студентов ВУЗов. Сам перечень авторов вызывает искреннее восхищение и глубокое уважение: Б.В. Гнеденко, В.С. Королюк, Е.Л. Ющенко!

Идеи АП оказались продуктивными для развития программирования в СССР. Как отмечал основатель киевской школы кибернетики В.М. Глушков: «В 1955–56 гг. в Киеве начал работать семинар, на котором был предложен ряд способов записи алгоритмов и методов программирования. Очень плодотворными оказались идеи языка адресного программирования (ЯАП), широко использовавшиеся при развитии теории программирования. С этого языка уже в начальный период развития кибернетики в АН УССР были созданы трансляторы, облегчившие процесс программирования для имевшихся в то время в Вычислительном центре ЭВМ «Киев», «Урал-1» и М-20. Впоследствии такими трансляторами были снабжены и другие отечественные ЭВМ» (цитируем по http://www.iprinet.kiev.ua/gf/nau_pp.htm).

Рассмотрим лишь методологические аспекты АП, выделив композиционный и номинативный аспекты. Эти аспекты являются центральными для композиционного (КП) и композиционно-номинативного программирования (КНП), которые развиваются на кафедре теории и технологии программирования Киевского национального университета имени Тараса Шевченко и которые можно рассматривать как развитие АП.

Указанная цель определяет и структуру статьи. В последующих разделах рассмотрены основы АП, КП и КНП. Здесь каждый из подходов проанализирован лишь в семантическом аспекте. Это значит, что программа трактуется как функция над некоторыми данными, а средства конструирования программ уточняются как композиции (операторы) над функциями. Программные системы рассмотрены как формализмы представления классов программ, они фактически задают языки программирования, используемые в подходе. Поэтому каждый подход описывается по такой схеме:

1) основные принципы подхода;
2) структуры данных;
3) функции;
4) композиции;
5) программные системы;
6) применение подхода.

1. ОСНОВЫ АП

Основные принципы АП — принцип адресности и принцип программного управления. Их формулировку цитируем по [1], где аббревиатура ЦАМ означает «цифровая автоматическая машина», ЗУ — «запоминающее устройство».

«Общим принципом построения программ для всех ЦАМ является принцип адресности. Информация о задаче кодируется и размещается в определенном порядке в ячейках ЗУ машины, имеющих адреса. Размещение исходных данных в ячейках ЗУ является начальным этапом программирования. Программа вычислительного процесса записывается в адресном виде, т.е. в программе указываются не сами числа, над которыми необходимо производить операции, а адреса ячеек ЗУ, содержащих эти числа. Выполнение программы на ЦАМ приводит к решению конкретной задачи при заданном заполнении ячеек ЗУ. Благодаря описанию программы в адресном виде есть возможность по одной и той же программе решать разные конкретные задачи, меняя заполнение ячеек ЗУ (параметры задач). Таким образом обеспечивается массовость алгоритма, описанного в виде программы.

Этот же принцип адресности лежит в основе и других методов описания вычислительных процессов. Например, при описании вычислительного процесса в буквенном виде подразумевается, что при конкретном осуществлении этого процесса вместо букв будут подставлены соответствующие им числа. Буквы являются обозначениями, метками, адресами соответствующих им чисел.

Второй общей для всех ЦАМ особенностью программного описания процессов вычислений является принцип программного управления. Порядок выполнения отдельных этапов вычислений строго определен в программе порядком команд и командами передачи управления, учитывающими результаты промежуточных вычислений.

Эти принципы программирования сохраним в определении адресной программы, отказавшись в описании процессов вычислений от других ограничений, связанных с техническими особенностями конструкций ЦАМ. При этом язык АП по возможности сохраняется машинным, в связи с чем перевод адресных программ для конкретных ЦАМ можно легко автоматизировать.» [1, с. 241].

Итак, авторы АП сформулировали лишь два принципа: адресности и программного управления. Для автора статьи важен именно этот выбор принципов, ведь их развитие приводит к КП и КНП. А именно, развитие принципа программного управления ведет к принципу композиционности, а принципа адресности — к принципу номинативности, которые и являются основными специальными принципами для КП и КНП.

Структуры данных АП. Данные в АП толкуются абстрактно. Это означает, что рассматривается произвольная система объектов S, между которыми установлены некоторые соотношения. Отдельные объекты из S называют кодами. Примерами систем объектов являются множество выделенных символов (алфавит) с операцией равенства, множество натуральных чисел с операциями константа 0 и следование и тому подобное. Также в системе S рассмотрены операция выделения содержимого адреса $'a$, где код a назван адресом кода $'a$. Далее введено понятие состояния системы кодов S (распределения адресов), определяющееся операцией выделения содержимого.

Типы функций АП. В АП операции определяются над системой S, но фактически они являются операциями над состояниями системы. Операция выделения содержимого может применяться несколько раз, давая содержимое адреса k-ранга. Эта операция определяется индуктивно: ${}^1a = 'a$, ${}^k a = '({}^{k-1}a)$, $k > 0$. Важной операцией является операция переноса a в b (обозначается $a \Rightarrow b$), которая изменяет содержимое адреса b на a. Кроме того, используются традиционные арифметические операции, а также предикатные функции, которые принимают лишь два значения (1 или 0).

Композиции АП. Понятие композиции в АП еще четко не сформулировано. Используются понятия порядка вычислений и команд передачи управления.

Выделяются операторы преобразования, или действия (сейчас употребляем термин «оператор присваивания») и операторы распознавания («условный оператор», «оператор разветвления» и тому подобное).

Для определения оператора преобразования операции выделения содержимого и переноса обобщаются на случай, когда их параметрами являются не коды, а функции над кодами. А именно, если f и g — функции над кодами, то $'f$ трактуется как оператор (композиция) выделения содержимого и который вычисляется таким образом: вычисляется значение f, полученное значение трактуется как адрес, а результатом вычисления будет содержимое этого адреса. Тогда оператор преобразования $g \Rightarrow f$ (в современной записи $f := g$) засылает значение функции g по адресу, который является значением функции f, т.е. в результате $'f$ будет равняться значению g.

Следует отметить четкое определение оператора распознавания $P\alpha\beta$ (в современной записи *if P then* α *else* β), где P — предикатная функция в S, а α, β — адресные формулы.

Оператор распознавания может использоваться в качестве оператора управления. Так, запись $P'\alpha^2\beta$ означает [1], что когда значением P есть 1, то нужно вычислять формулу $'\alpha$, т.е. формулу, обозначение которой является содержимым адреса α; если же значением P есть 0, то действует формула, обозначением которой является содержимое адреса второго ранга β. Отметим, что такое толкование оператора распознавания как оператора управления требует внесение адресных формул в систему кодов S.

Программные системы АП. Принципы АП определяют основные его аспекты: адресность и управляемость. Конкретное воплощение этих принципов происходит в ЯАП, называемого также адресным языком. Следует отметить, что этот язык описывается неформально и синкретически, без четкого выделения семантического и синтаксического аспектов. Вызвано это тем, что ЯАП был одним из первых языков программирования, в частности, рукопись пособия [1] подготовлена еще до появления официального описания языка Алгол-60, который фактически определил БНФ как стандарт для представления синтаксиса языков программирования. Соответствующие исследования семантики были лишь в начальной фазе. Вместе с тем высокая математическая культура авторов позволила достаточно точно определить семантику языка в терминах множеств, функций, отношений и т.д.

В ЯАП «адресная программа задается исходным распределением адресов в S (состоянием системы кодов S) и последовательностью адресных формул с указанием порядка их применения» [1]. Порядок действий задается или операторами распознавания или традиционным методом, когда после выполнения операции выполняется следующая за ней. Допускается также, что некоторые операции могут выполняться в произвольном порядке, тем самым возможен определенный недетерминизм выполнения программы. Эта идея также была новаторской в то время.

ЯАП является фактически специализацией отношения *адрес* $\mapsto$ *код*: в ячейке с определенным адресом сохраняется код. Одной из важных особенностей языка есть то, что разрешается непрямая адресация, т.е. в ячейке может находиться новый адрес. Это существенно повышает выразительную мощность ЯАП. Без непрямой адресации невозможно было бы обрабатывать последовательности чисел разной длины.

Применение АП. Адресный язык был применен для разработки ряда алгоритмов для решения как арифметических, так и не арифметических задач. ЯАП широко использовался для построения интерпретаторов и трансляторов для первых отечественных ЭВМ.

Наличие операторов и композиций, опирающихся на непрямую адресацию, делают ЯАП весьма мощным, вместе с тем эта мощность усложняла построение программ. По выражению Е.М. Лаврищевой, активно работавшей с ЯАП, «за красоту этого языка приходилось расплачиваться сложностью отладки программ». Поэтому не удивительно, что позже начали рассматривать более простые программные системы, в частности алгоритмических алгебр [6], близкие к системам структурного программирования и которые не имеют операций непрямой адресации.

2. ОСНОВЫ КОМПОЗИЦИОННОГО ПРОГРАММИРОВАНИЯ

Принципы КП изложены в работах [7–9]. За основу взят семиотический подход, согласно которому главными аспектами программ являются прагматический, семантический и синтаксический. Принципы КП поделены на общие и специальные. Первые (принципы подчиненности, отделения, функциональности) фактически определяют отношения между основными аспектами программ, вторые (принципы композиционности, полноты, адекватности, сведения) раскрывают свойства композиций программ. Формулировка принципов такова.

Принцип подчиненности: прагматический аспект программ является определяющим, ему подчинен семантический аспект, а синтаксический аспект в свою очередь есть производным от семантического. Принцип отделения: семантический и синтаксический аспекты относительно независимы. Принцип функциональности: в математико-семантическом плане программы суть функции, которые отображают одно множество (начальных данных) в другое (результатов).

Принцип композиционности: программы строятся из относительно более простых программ с помощью композиций. Для системы композиций и функций формулируются дополнительные требования, а именно, такая система должна быть достаточно мощной, чтобы представить произвольную программу с сохранением ее внутренней структуры (принципы полноты и адекватности). Дальнейшее ограничение системы композиций задается принципом сведения, который говорит о целесообразности учета лишь определенных свойств функций, а именно свойств быть именной или параметризованной конечнозначной функцией. Приведенные принципы дают определенное обоснование основного тезиса подхода: программирование суть композиционное программирование.

Дальнейшее уточнение КП сводится к экспликации данных, функций и композиций.

Структуры данных КП. Сделаны такие обогащения:

- понятие адреса повышается до более общего понятия имени;
- явно вводится множество имен V и множество базовых значений W. Тем самым четко различаются имена и базовые значения, которые в АП представлены одним множеством кодов S;
- на передний план рассмотрения выходит понятие именного множества, аналогом которого в АП было понятие состояния системы кодов S;
- значения рассмотрены как иерархически построенные из элементов множеств V и W.

Именное множество определено как множество, состоящее из именованных элементов, т.е. пар, первый компонент которых является именем, а второй — значением. Понятие именного множества лежит в основе определения универсума $NamD(V,W)$ именных данных. А именно, элементы из W, а также пустое множество $\varnothing$, являются объектами (именными данными) ранга 0. Именные данные ранга $i+1$ — конечные множества пар вида $\{(v_1,\ d_1),\ldots,(v_n,\ d_n)\}$, где $v_1,\ldots,v_n$ — имена из V, а $d_1,\ldots,d_n$ $(n\geq 0)$ — именные данные ранга, равного или меньшего i. Имена $v_1,\ldots,v_n$ должны быть разными, это требование называется принципом однозначности именования.

С помощью именных данных можно промоделировать большинство традиционных структур данных, которые используются в программировании, а именно, записи, файлы, массивы, реляции и тому подобное [9, 10].

Типы функций КП. Введение именных данных как универсального класса позволяет КП ограничиться рассмотрением функций типа $NamD(V,W)\to NamD(V,W)$, которые называются именными функциями.

Отметим, что «обычные» n-арные функции вида $D^n\to D$ можно достаточно естественно трактовать как конкретизацию именных функций, подавая кортеж данных $(d_1,\ldots,d_n)$ как именное множество $\{(1,\ d_1),\ldots,(n,\ d_n)\}$ с именами $1,\ldots,n$. Приведенное толкование позволяет выделить разные категории именных функций, в частности [8]:

- ординарные n-арные функции, заданные на значениях (например, арифметические операции, сравнение и тому подобное);

- структурные n-арные функции, областью значения которых является множество имен (например, функции конкатенации имен, индексирования и тому подобное);
- интерфейсные функции, которые позволяют «превращать» ординарные функции с помощью структурных в именные произвольной сложности (например, функции именования, разыменования, пересылки и тому подобное).

Приведем определения лишь некоторых структурных функций ($d \in NamD(V, W)$):

- функция именования $\Rightarrow v$ (с параметром $v \in V$): $\Rightarrow v(d) = \{(v, d)\}$;
- функция разыменования $v \Rightarrow$ (с параметром $v \in V$): $v \Rightarrow (d) = d'$, если $(v, d') \in d$;
- функция пересылки: $v \Rightarrow u$ (с параметрами v, $u \in V$): $v \Rightarrow u(d) = d'$, где d' отличается от d только значением имени u, которое становится равным значению имени v;
- функция генерации имени $> w <$ (с параметром $w \in W$): $> w < (d) = v$, если $(v, w) \in d$.

Как видно из приведенных определений, в КП используется более богатый класс функций, чем в АП.

Композиции КП. В АП не сформулировано четкое понятие композиции как средства конструирования. Для КП это понятие центральное ввиду трактовки композиций как общезначимых средств конструирования программ. В этом смысле композиции задают логическую составляющую программирования, а функции — его предметную составляющую. Поэтому в КП происходит дальнейшее развитие принципа программного управления в таких направлениях:

- четко разграничена семантика и синтаксис программ, главную роль играет семантический аспект;
- предоставлено семантическое уточнение программных управляющих структур как композиций программ;
- построен ряд композиций программ разного уровня абстракции, которые ориентированы на именные структуры данных и функций, и в том числе — на непрямое именование (непрямую адресацию);
- сформулированы важные проблемы полноты и адекватности класса композиций.

В КП композиции разделены на два класса: коннотативных и денотативных. Неформально коннотативные композиции характеризует наличие процедур их вычисления, а для денотативных такие процедуры явно не заданы. Примерами коннотативных композиций являются операторы последовательного выполнения, разветвления, цикла и тому подобное. Денотативные композиции задаются как решения определенных уравнений.

Программные системы КП. Применение в КП семантико-синтаксического подхода к определению программ индуцирует соответствующее определение программной системы, в основе которого находится алгебра программ (функций), операциями которой являются композиции. Для представления синтаксиса использованы как традиционные грамматики, так и разные их обобщения [11, 12]. Интегрированное семантико-синтаксическое представление программ осуществляется с помощью программных дефиниторов [7]. Неформально программные дефиниторы — БНФ, в которых каждое правило отвечает определенной функции или композиции.

Применение КП касается построения формальных моделей языков программирования и языков запросов к базам данных [10, 13]. На основе этих моделей разработаны системы разного назначения.

Отметим, что КП стимулировало идентификацию и точную формулировку многих проблем теории программирования. В частности, большое значение имеют принципы полноты и адекватности. Полнота понималась преимущественно в двух аспектах:

- прагматическая полнота (мощность программной системы достаточна для построения практически важных программ);
- вычислительная полнота (программная система порождает полный класс вычислимых функций над структурами данных, которые задаются этой системой).

Таким образом, в КП развиты принципы АП введением класса именных данных, основанных на отношении (*имя*, *значение*), а также формулировкой и исследованием четкого математического понятия композиции как общезначимого средства конструирования программ.

3. ОСНОВЫ КОМПОЗИЦИОННО-НОМИНАТИВНОГО ПРОГРАММИРОВАНИЯ

Принципы КНП [14, 15]. Его целью было дальнейшее развитие АП и КП в нескольких направлениях. Для всех трех подходов главным объектом исследования являются программы, но АП и КП сосредоточены преимущественно на уточнении структурных, внутренних аспектов программ (в первую очередь семантического и синтаксического), а КНП рассматривает программы в более широком внешнем контексте, связанным с особенной деятельностью субъекта. Такое расширение контекста требует соответствующего обогащения уровней рассмотрения исследуемых понятий, а именно, выделены такие уровни:

- общеметодологический (философский);
- обще- и конкретнонаучный (профессиональный);
- математический (формальный).

Приведенным уровням соответствует три класса понятий, на которых основывается теория программирования:

- категории, задающие всеобщие признаки предметов;
- научные понятия, задающие особенности программирования;
- формальные понятия как математические уточнения научных понятий.

Категории не являются непосредственным предметом теории программирования, вместе с тем исследования в этой области невозможны без понимания системы категорий и их диалектики. Примеры важных для теории программирования категорий: абстрактное–конкретное, качество–количество–мера, элемент–часть–целое, форма–содержание, явление–сущность, единичное–особенное–всеобщее и много других. Научные понятия часто выступают как определенное развитие, так и определенное ограничение (проекция) категорий. Демонстрация такой связи между категориями и понятиями представляется очень важной, учитывая интегрирующий аспект теории программирования.

В соответствии с уровнями рассмотрения формулируют принципы КНП. К общеметодологическим относятся принципы, характеризующие общие законы развития.

Принцип гносеологичности: уточнение основных понятий программирования осуществляется в соответствии с общими законами (принципами) гносеологии, применение которых имеет программистскую направленность.

Гносеология как наука о познании должна в частности предоставить общие законы уточнения понятий конкретных предметных областей. Сложность использования гносеологии с этой целью заключается в том, что, невзирая на многовековую историю ее развития, все же не созданы общепринятые законы научного познания. Поэтому в КНП ограничиваются лишь принципами, релевантными программированию.

Принцип универсальной взаимосвязи: существует универсальная взаимосвязь и взаимообусловленность предметов и явлений мира.

Как следствие, каждый предмет имеет много аспектов. Их исследуют согласно принципу развития от абстрактного к конкретному (от простого к сложному, от низшего уровня к более высокому, от старого к новому): понятия программирования уточняются в процессе их развития. Этот процесс начинается с наиболее абстрактных представлений, отображающих общие свойства программирования, затем переходит к более конкретным представлениям в их единстве, отображающим особенные и специфические свойства, постепенно раскрывая понятия программирования в их богатстве и взаимосвязях.

Такой процесс преимущественно происходит согласно основным схемам развития, среди которых центральное место принадлежит схеме триадичности развития: тезис — антитезис — синтез (принцип триадичности развития). Этот принцип широко использовал Г.В.Ф. Гегель при построении системы философии.

Выбор направления развития определяет принцип единства теории и практики: теорию и практику программирования необходимо рассматривать в их един-

стве и взаимообусловленности. Этот принцип фактически утверждает «равноправие» и взаимозависимость теории и практики.

Сделаем два замечания: 1) приведены не все методологические принципы, а лишь те, которым следует уделить особенное внимание в теории программирования; 2) методологические принципы не абсолютны, они имеют относительную природу.

Среди общенаучных основным будет принцип системности, требующий всестороннего изучения предметов и явлений, в единстве целого и частей, с учетом разнообразных связей между частями системы и с внешней средой. Этот принцип можно рассматривать как определенную «проекцию» на общенаучный уровень категорий сущности, в частности категорий «часть–целое».

Согласно принципу единства интенсиональных и экстенсиональных аспектов [16] научные понятия должны быть поданы в единстве этих аспектов. Здесь берется традиционное толкование интенсионала как содержания понятия, экстенсионала — как его объема. Указанные аспекты можно рассматривать как проекции категорий «(все)общее–особенное–единичное». Интенсиональный аспект играет ведущую роль относительно экстенсионального.

Наконец, принимается один из дескриптологических принципов КП о ведущей роли семантического аспекта по отношению к синтаксическому. Эти аспекты можно рассматривать как проекции категорий «содержание–форма».

Среди конкретнонаучных принципов выделяются лишь принципы, которые задают особенности подхода: композиционности и номинативности. Основной для КП принцип композиционности отмечает необходимость исследования композиций как средств конструирования программ, принцип номинативности утверждает важность отношений именования в их построении и описании.

Приведенные принципы определяют направления и особенности экспликации основных понятий программирования в КНП. Примем трехуровневую схему экспликации, средний уровень которой определяет научные понятия. Для них устанавливаются их связи с категориями (верхний уровень), а также строятся их формализации (нижний уровень).

Проиллюстрируем связь категорий и понятий с позиций праксеологического (деятельного) аспекта. Здесь начальной категорией (тезисом) является категория «субъект». Ее антитезой является категория «цель», синтезом — категория «средство». Проекция категориальной триады *субъект–цель–средство* на научный (профессиональный) уровень дает триаду понятий *пользователь–проблема–программа.* Дальнейшее развитие этой триады с помощью триад *пользователь–программа–процесс выполнения* и *проблема–программа–процесс программирования* приводит к развитию основных понятий теории программирования и их отношений в пентаде *пользователь–проблема–программа–процесс выполнения–процесс программирования* [14] (рис. 1).

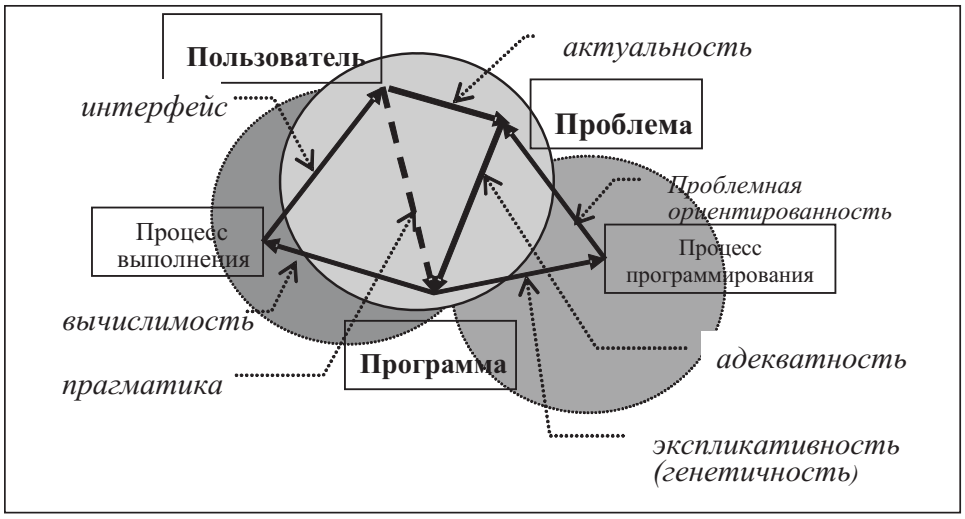

Рис. 1. Пентада основных понятий программирования

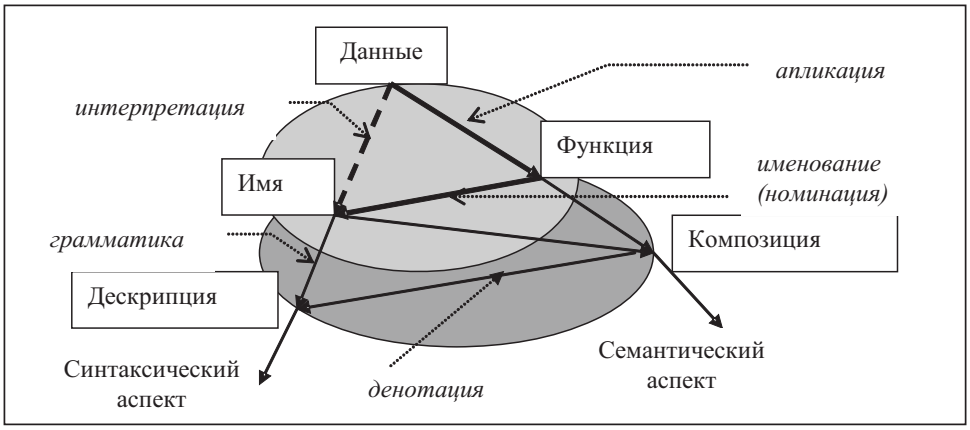

Рис. 2. Пентада основных программных понятий

Следующий этап заключается в развитии понятий приведенной пентады. Здесь рассмотрим развертывание понятия программы в его дескриптивном аспекте, заданное пентадой: *данные–функция–имя функции–композиция–дескрипция* [14].

Введенные понятия связаны рядом отношений (рис. 2), которые здесь не раскрыты. Отметим, что пентады позволяют сформулировать более богатую, чем семиотическая, систему сущностных аспектов программ, которая состоит из двух классов:

- внешние аспекты: адекватность, прагматика, вычислимость, генетичность,
- внутренние (структурные) аспекты: семантика, синтаксис, денотация.

Дальнейшие определения раскрывают введенные понятия, тем самым создавая иерархию понятий (онтологию) теории программирования. Разработка такой системы понятий предоставляет возможность их формализации. Для этого следует формализовать в первую очередь понятия структур данных, функций и композиций.

Структуры данных КНП. Традиционно данные формализуют на теоретико-множественной платформе, в частности так сделано в АП и КП. Вместе с тем современные методы разработки программ остро ставят проблему построения формальных моделей программ разного уровня абстракции, для которых теоретико-множественная платформа не всегда адекватна [16]. Поэтому в КНП разработана специальная классификация данных разных уровней абстракции, названная 3× 3 типологией данных. Эта классификация — «произведение» проекций категорий целое–часть и абстрактное–конкретное.

Первая пара категорий развивается в соответствии с триадой *целое* (W) — *часть* (P) — *иерархия* (H), а вторая — *абстрактное* (A) — *конкретное* (C) — *синтетическое* (S). Получается девять типов данных (рис. 3). Их характеристика такова.

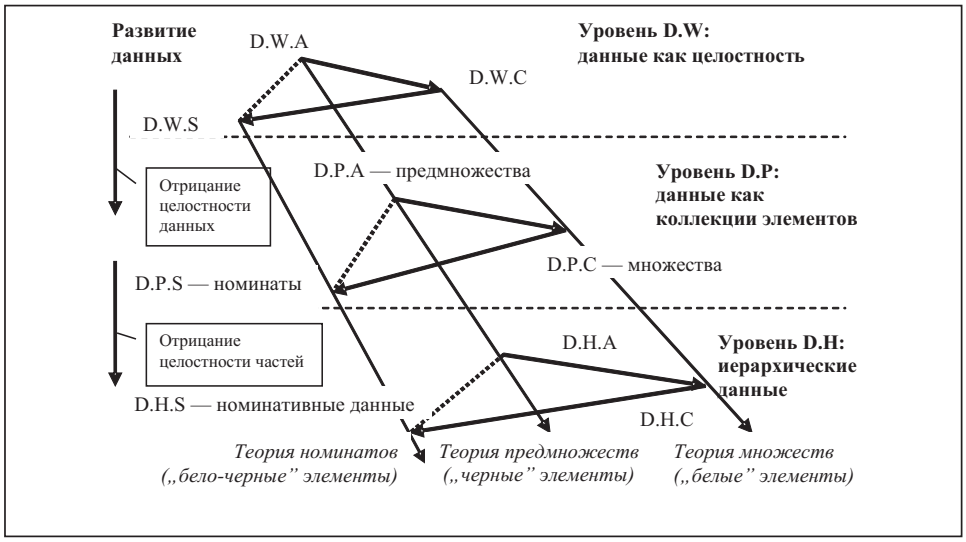

Рис. 3. Диаграмма 3 × 3 типологии данных

Данные верхнего уровня (D.W) рассматриваются как целые (целостные, неструктурированные объекты), неизменные, пока это явно не указано (закон тождества). На этом уровне выделяют три подтипа: абстрактный (D.W.A, данное — «черный ящик»), конкретный (D.W.C, данное — «белый ящик»), синтетический (D.W.S, данное — «черный или белый ящик»).

На втором уровне (D.P) данные имеют части. Такие данные назовем коллекциями. Их неформальное толкование:

- каждая часть коллекции является определенным целым;
- к таким частям применим закон тождества , т. е. части не изменяются, пока это явно не сказано;
- части отделены одна от другой;
- части независимы одна от другой, т.е. не имеют сильных связей между собой (в терминологии Гегеля «безразличны» друг к другу), такие части называются элементами целого;
- все части «доступны», т. е. каждую часть можно получить для обработки;
- можно говорить об исчерпывающей обработке всех частей.

Все эти свойства неформальны, экспликация определяет разные типы коллекций. Коллекции с абстрактными элементами (D.P.A) называют предмножествами, с конкретными (D.P.C) — множествами, с синтетическими элементами (D.P.S) — номинатами (от латинского *nomen — имя*).

Данные третьего уровня (D.H) являются иерархическими данными, классифицируемые соответственно как иерархические предмножества (D.H.A), иерархические множества (D.H.C), иерархические номинаты — номинативные данные (D.H.S).

Наиболее важный тип данных для программирования — номинаты — построены как отношение *имя* $\mapsto$ *значение*. Здесь имя рассматривается на конкретном уровне («белый ящик»), значение — на абстрактном уровне и может быть «черным ящиком». Для номинатов используем обозначение вида $[v_1 \mapsto d_1, \ldots, v_n \mapsto d_n]$, где $v_1, \ldots, v_n$ — имена из V, а $d_1, \ldots, d_n$ — значения. В отличие от КП имена в этой последовательности могут быть одинаковыми, т.е. допускается многозначное именование. Кроме того, в КНП рассматриваются не только конечные, но и бесконечные номинаты. Номинаты (как синтетические объекты) имеют двойственную природу: с одной стороны — это коллекции, а с другой — функции, потому что связь *имя* $\mapsto$ *значение* имеет функциональный оттенок. Можно утверждать, что функциональная природа номинатов является важной для операций над ними, потому в КНП часто используют это свойство номинатов (принцип теоретико-функциональной формализации [16]).

Понятие номината лежит в основе определения универсума $NomD(V, W)$ номинативных данных, который строится на основе множества имен V и предмножества базовых значений W. А именно, элементы из W, а также пустой номинат [] являются объектами (номинативными данными) ранга 0. Номинативные данные ранга $i+1$ являются номинатами вида $[v_1 \mapsto d_1, \ldots, v_n \mapsto d_n]$, где $v_1, \ldots, v_n$ — имена из V, а $d_1, \ldots, d_n$ $(n \geq 0)$ — номинативные данные ранга, равного или меньшего i $(i \geq 0)$.

Типология номинативных данных основана на классификации фундаментального отношения *имя* $\mapsto$ *значение*, естественно возникающей как классификация составляющих этого отношения и их взаимосвязей:

- значения классифицируют как простые (неструктурированные) и сложные (структурированные);
- имена классифицируют как простые и сложные (структурированные);
- имена и значения могут быть независимы (только прямое именование) или зависимы (возможно непрямое именование).

Три бинарных параметра (три оси) дают восемь типов номинативных данных. Такую классификацию назовем типологическим кубом номинативных данных (рис. 4). Представление выбрано в виде куба, поскольку его ребра фактически задают интенсиональное включение одного типа в другой (слева направо и снизу вверх). Тем самым тип TND_1 интенсионально самый простой, а TND_8 самый сложный.

Отметим, что в АП преимущественно рассматривались типы TND_1 и TND_2 (имена и значения простые, есть непрямое именование), а в КН — $TND_1 - TND_4$ (есть сложные значения, есть непрямое именование).

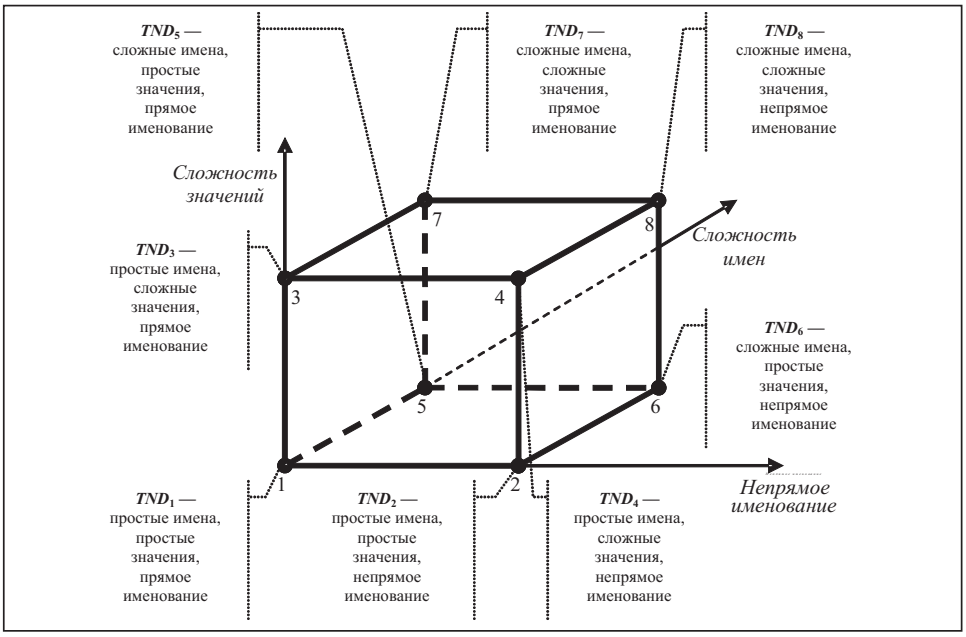

Рис. 4. Типологический куб номинативных данных

Типы функций КНП. Приведенные классификации данных индуцируют соответствующие классификации функций, которые исследуются в КНП. Классы функций (программ) над данными абстрактного уровня D.W.A интенсионально очень бедны. Более богатыми являются классы функций над данными синтетического уровня D.W.S (булев уровень), которые позволяют задавать предикаты разных типов [17]. Самыми богатыми являются классы функций над данными номинативного уровня (D.P.S и D.H.S). Наиболее важны для программирования функции типа $NomD(V,W) \to NomD(V,W)$. Приведем некоторые примеры таких функций (операций):

- именование $\Rightarrow v$ с параметром $v \in V$:
$$\Rightarrow v(d) = [v \mapsto d] \text{ для любого } d \in NomD(V,W),$$
- разыменование $v \Rightarrow$ с параметром $v \in V$:
$$v \Rightarrow (d) = d', \text{ если } d(v) = d',$$
- функция $v!$ наличия компоненты с именем $v \in V$:
$$v!(d) = \begin{cases} [], \text{ если } d(v) \text{ определено,} \\ d, \text{ если } d(v) \text{ не определено;} \end{cases}$$
- многозначная функция выбора $\searrow$ между пустым и входным данным: $\searrow(d)$ равняется или $\varnothing$, или d.

Композиции КНП также упорядочиваются в соответствии с классификациями данных и функций. Основными композициями функций являются [17]:

- бинарная композиция умножения $\bullet$ (последовательное выполнение);
- тернарная композиция разветвления Δ (условный оператор *if_then_else*);
- бинарная композиция условной итерации * (цикл *while_do*);
- бинарная композиция наложения ∇ (для вычисления $f \nabla g\ (d)$ нужно вычислить $f\ (d)$ и «записать» на полученные номинативные данные результаты $g\ (d)$).

Каждая композиция отвечает определенному уровню абстракции в 3×3 типологии данных. А именно, композиция умножения задана на абстрактном уровне рассмотрения данных D.W.A (данные — «черные ящики»). Композиции разветвления и условной итерации задаются на уровне D.W.S, для которого достаточно наличия только одной логической константы T («белый ящик»), остальные данные — «черные ящики». Композиция наложения задается на номинативном уровне D.H.S, потому что для ее вычисления следует опираться на структуры номинативных данных.

Наибольшее внимание в КНП уделено исследованию композиций, которые упорядочиваются согласно кубической типологии номинативных данных. Это,

в первую очередь, композиции, основанные на непрямом именовании: бинарная композиция присваивания $AS(f,g)$ и унарная композиция взятия значения $VAL(f)$. Эти композиции являются соответственно аналогами таких операторов АП, как $g \Rightarrow f$ и $'f$. Рассматриваются также композиции и функции, связанные со сложными именами. Исследуются проблемы независимости композиций и их вычислительной полноты [18].

Программные системы КНП определяют в соответствии со схемой развития программных понятий, задаваемой программной пентадой, в которой выделены семантический, синтаксический и денотационный аспекты. Каждый из аспектов представляется соответствующей системой, фиксирующей его связь с другими аспектами. Такие системы называются соответственно композиционными, дескриптивными и денотационными. Композиционные системы определяют средства построения функций над некоторым множеством данных; дескриптивные — дескрипции как описания функций; денотационные — значения дескрипций. В зависимости от уровня абстракции такие системы могут быть простыми или очень сложными.

В простейшем случае под композиционной системой понимают тройку $(\boldsymbol{D},\boldsymbol{Fn},\boldsymbol{C})$, где $\boldsymbol{D}$ — множество данных, $\boldsymbol{Fn}$ — множество функций над $\boldsymbol{D}$, $\boldsymbol{C}$ — множество композиций (операций) над $\boldsymbol{Fn}$. Если множество $\boldsymbol{D}$ явно не указано, то композиционные системы приобретают вид $(\boldsymbol{Fn},\boldsymbol{C})$. Их называют функциональными (программными) алгебрами.

Дескриптивные системы определяют дескрипции индуктивно, используя имена базовых функций и предикатов, а также имена композиций. Как правило, дескрипциями являются термы программной алгебры.

Программные системы определяют как композиционно-номинативные системы $(\boldsymbol{Cs},\boldsymbol{Ds},\boldsymbol{Dns})$, где $\boldsymbol{Cs}$ — композиционная, $\boldsymbol{Ds}$ — дескриптивная, $\boldsymbol{Dns}$ — денотационная системы.

Классы предикатов также можно задавать композиционно-номинативными системами, представляя разные логики предикатов в едином стиле с программными системами [17].

Применение КНП. В первую очередь КНП разворачивалось как интеграционный проект, который имеет целью интегрировать такие базовые для программирования дисциплины, как теорию программирования, математическую логику и теорию алгоритмов. Центральным понятием в КНП является понятие композиционно-номинативного языка, который в семантическом плане основывается на понятии композиционно-номинативной алгебры. Это позволило в рамках одного подхода построить иерархию взаимосвязанных формальных систем, которые служат моделями языков программирования и языков спецификаций, логик предикатов разного типа, а также задают разные типы абстрактной вычислимости функций [14–18].

ЗАКЛЮЧЕНИЕ

В значительной и многогранной научной деятельности Е.Л. Ющенко мы остановились лишь на методологическом анализе принципов АП. Этот анализ позволяет сделать вывод, что его авторы сумели (невзирая на неразвитость программирования на тот период) просто и четко сформулировать фундаментальные принципы и проблемы теории программирования. Это в первую очередь касается принципа адресности, провозглашающего отношение *имя–значение* (*адрес–код*) базовым для программирования. Его важность заключается и в том, что оно является одним из начальных моментов развертывания категорий «сознание» и «знание», и тем самым ставит программирование в широкий общечеловеческий контекст научных исследований. Другой принцип адресного программирования — *принцип программного управления* — акцентировал внимание на логико-структурной составляющей программ, которая фактически является отображением связей в паре категорий «часть–целое». Поэтому и этот принцип не является лишь профессиональным, а имеет общезначимую природу.

Ввиду фундаментальности приведенных принципов, нет ничего странного в том, что они постоянно развиваются в теории программирования. Так, в рамках алгебраического подхода основателя Киевской школы кибернетики В.М. Глушкова, эти

принципы приобрели новое звучание в композиционном и композиционно-номинативном программировании. А это и утверждает жизненность научных идей Екатерины Логвиновны Ющенко, которая всю свою жизнь посвятила развитию программирования в Украине. За это ей наше глубокое уважение и искренняя благодарность.

СПИСОК ЛИТЕРАТУРЫ

1. Гнеденко Б.В., Королюк В.С., Ющенко Е.Л. Элементы программирования. — М.: Физматгиз, 1961. — 348 с.
2. Королюк В.С. Про один спосіб програмування // Доп. АН УРСР. — 1958. — № 12. — С. 1292–1295.
3. Королюк В.С., Ющенко Е.Л. Вопросы теории и практики программирования // Сб. ВЦ АН УССР. — 1960. — **1**.
4. Ющенко Е.Л. Адресні алгоритми та цифрові автоматичні машини // Там же. — 1960. — **2**.
5. Ющенко Е.Л., Быстрова Л.П. Програмуюча програма, інформацією для якої служить адресний алгоритм // Там же. — 1960. — **3**.
6. Глушков В.М. Теория автоматов и формальные преобразования микропрограмм // Кибернетика. — 1965. — № 5. — С. 3–11.
7. Редько В.Н. Композиции программ и композиционное программирование // Программирование. — 1978. — № 5. — С. 3–24.
8. Редько В.Н. Основания композиционного программирования // Там же. — 1979. — № 3. — С. 3–13.
9. Редько В.Н. Семантические структуры программ // Там же. — 1981. — № 1. — С. 3–14.
10. Басараб И.А., Никитченко Н.С., Редько В.Н. Композиционные базы данных. — Киев: Либідь. — 1992. — 191 с.
11. Редько В.Н., Трубчанинов Г.Г. Синтаксические дефиниции (структурный подход) // Программирование. — 1977. — № 5.
12. Редько В.Н., Шкильняк С.С. Структуры синтаксических дефиниций языков программирования // Там же. — 1979. — №. 4. — С. 3–15.
13. Редько В.Н., Брона Ю.Й., Буй Д.Б., Поляков С.А. Реляційні бази даних: табличні алгебри та SQL-подібні мови. — Київ: Видавничій дім «Академперіодика», 2001. — 198 с.
14. Nikitchenko N. A Composition Nominative Approach to Program Semantics. — Techn. Rep. IT–TR: 1998–020. — Technical University of Denmark. — 1998. — 103 p.
15. Никитченко Н.С. Композиционно-номинативный подход к уточнению понятия программы // Проблемы программирования. — 1999. — № 1. — С. 16–31.
16. Никитченко Н.С. Интенсиональные аспекты понятия программы // Там же. — 2001. — № 3–4. — С. 5–13.
17. Нікітченко М.С., Шкільняк С.С. Математична логіка та теорія алгоритмів. — Київ: ВПЦ «Київський університет», 2008. — 528 с.
18. Nikitchenko N.S. Abstract Computability of Non-deterministic Programs over Various Data Structures // Perspectives of System Informatics (Proc. of Andrei Ershov Fourth Int. Conf.). — Novosibirsk, Russia, July 2–6, 2001, Revised Papers. Lecture Notes in Comput. Sci. — 2244, Springer, 2001. — P. 468–481.

Поступила 08.07.2009