Глава 2. Алгебраический подход к представлению понятий

1 Понятия как объекты моделирования, примеры

1.1 Основные характеристики баз понятий

Тема этой главы — некоторые математические средства представления понятий для автоматизированной работы с ними.

В этой главе, следуя [40, 46, 54, 70, 9], рассматривается алгебраический подход к моделированию понятий и их представлений.

Алгебраические методы лежат в основе построения реляционных моделей баз данных и абстрактных типов данных в программировании. В диссертации опыт применения алгебры в этих областях используется для приложения к более широкой задаче: моделированию понятий и формированию баз понятий.

Базы понятий — это новая складывающаяся область в программировании, связанная с использованием и построением больших библиотек типов объектов в объектно-ориентированных языках программирования, а также с использованием CASE–технологий и построением специализированных программных систем типа TOOLKIT, элементы которых предназначены для многоразового многоцелевого использования в различных приложениях.

Практика и опыт использования растущих библиотек подобного рода требует некоторой стандартизации в их организации и формах обращения к ним, которая была бы основана на ясной модели отдельных понятий и базы понятий в целом.

Для начала сформулируем несколько свойств, выделяющих базы понятий среди других объектов программирования.

1. База понятий — это большая совокупность (сотни — тысячи) отдельных понятий (типов объектов, абстрактных типов данных, фрагментов схем баз данных, баз данных и т.д.).

2. Понятия из базы понятий предназначены для многоразового многоцелевого использования в приложениях, в постановках задач, в формулировках запросов.

3. Система понятий должна быть открыта для расширения и модификации пользователями (введение новых понятий, версий понятий, уничтожение неиспользуемых понятий).

4. Понятия в базе понятий взаимосвязаны по отношениям:

   (a) использования (одно понятие может использоваться в другом);

   (b) конкретизации (понятие может быть уточнением более абстрактного другого понятия);

   (c) реализации (одно понятие может иметь несколько реализаций).

5. Работа с базой понятий состоит из интерактивного процесса формирования новых понятий (задач) с использованием понятий из базы понятий, постановок вопросов в среде нового понятия и получения ответов с последующим возможным использованием понятия в приложениях.

6. Использование понятий должно обеспечивать открытость языка работы с понятиями и их элементами, обеспечивать пользователю возможность расширения языка языковыми конструкциями прикладной области.

7. Должна быть обеспечена принципиальная совместимость различных баз понятий — возможность использования понятий из нескольких специализированных баз понятий.

Состояние работ и исследований по базам понятий очень напоминает ситуацию с базами данных в конце 60-х начале 70-х годов, когда была осознана необходимость отделения процессов ведения данных от их использования, были созданы файловые системы, но не было ясной концепции баз данных. Для формулировки одной из таких концепций был успешно использован алгебраический язык в работах Е. Кодда.

Цель этой главы показать, что алгебраические методы, с учетом опыта их применения в теории баз данных и абстрактных типов данных, могут оказаться полезными и в моделировании баз понятий.

Представление данных в виде алгебраических структур является естественным для многих специалистов в области программирования. Начиная с известной работы Е.Кодда [43], применение алгебраических методов распространилось на моделирование баз данных. Объектом изучения стали алгебраические системы с операциями над отношениями — реляционные алгебры. В настоящее время реляционные алгебры хорошо изучены в работах зарубежных и отечественных специалистов, включая работы автора.

Следующим шагом явилось изучение алгебраическими методами схем баз данных, расширение систем операций операциями над типами данных и над схемами. Это направление привело к построению алгебраических моделей систем представления знаний, в которых существенна неполнота знаний. Среди алгебраических средств в таких моделях особую роль играют понятия и методы теории категорий [16].

Основное, что используется из алгебры в приложениях,— это алгебраический язык. Следующее — это алгебраический стиль мышления: использование понятий изоморфизма и гомоморфизма алгебраических систем, задание алгебраических систем с помощью образующих и соотношений. Дальнейшее использование алгебры приводит к использованию специфических понятий и результатов, введенных и исследованных в алгебре математиками. К ним относятся специфические операции и алгебраические системы, теоремы о разрешимости в алгебрах, специфические алгоритмы.

Результатами алгебраического исследования, помимо построения языка, на котором могут ставиться вопросы в области теории баз данных и понятий, могут быть теоремы классификации построенных алгебр и, в идеале, алгоритмы для распознавания тождеств, изоморфизмов, гомоморфизмов систем и решений уравнений, которые могут быть использованы в системах проектирования, алгоритмах оптимизации запросов, в алгоритмах получения ответов на запросы.

Кроме того, как и в базах данных, в базах понятий существенный вопрос — это полнота использованной системы операций, который в некоторой постановке тоже может быть сведен к алгебраической задаче.

1.2 Типы данных как примеры понятий

Начнем с примеров понятий, которые обычно представляются в развитых системах программирования, — это типы данных, абстрактные типы данных, типы объектов.

В настоящее время имеется большое количество статей и монографий по типам данных в программировании, по абстрактным типам данных, по применению современных математических методов к моделированию типов данных в программировании, по языкам спецификаций типов данных [20, 1, 46, 25, 12, 7] . В этой области сложились общее понимание задач, некоторая система понятий, выработался язык и некоторые методы, то есть произошло становление области науки в информатике, которое впитало в себя некоторые достижения опыта, методов и представлений в программировании, алгебре и логике.

Основная задача введения развитого механизма типов данных в программирование — это программная поддержка общения человека с машиной в понятиях, привычных и удобных специалисту во время структурирования программы, программирования, отладки программы и ее модификации.

Понятие типа данных используется в том или ином объёме во всех языках программирования "высокого уровня". Как отмечает В.Н.Агафонов в своем обзоре [2], введение типов данных в программирование, по всей вероятности, связано с человеческим способом умственной деятельности, требующим образования понятий и работы с ними. С другой стороны, так как цель программирования — переложить ряд процессов обработки информации на современные машины со сложившимися к настоящему времени принципами их работы, то машинная деятельность накладывает определенные ограничения на форму описания понятий, степень подробности такого описания и на необходимость уложиться в машинные возможности. Однако, с совершенствованием вычислительных машин и использованием новых более совершенных принципов машинной обработки ряд ограничений преодолеваются, и всё ясней выделяются принципиальные ограничения, связанные с вычислимостью процессов, их сложностью и конечностью памяти в каждый момент времени.

В этих условиях первостепенной проблемой становится проблема проектирования общения человека с машиной, обеспечивающего человеку понимание и управление процессом обработки данных на уровне, что должна сделать машина без подробной информации о том, как она это делает. Большую подробность система представляет по особому запросу и, может быть, специалисту другого класса, специализирующемуся на эффективных реализациях введенных типов данных.

В каждом развитом языке программирования имеются исходные (базо-

вые или основные) типы данных, которые реализуются на уровне машинных команд. В языке АДА [14] такие типы называются предопределенными. К ним обычно относятся следующие типы данных (в разных языках программирования они могут отличаться названиями) INTEGER, FLOAT (REAL), BOOLEAN и CHARACTER (целые числа, числа с плавающей запятой, булевы значения, полный набор символов).

Рассмотрим некоторые примеры распространенных базисных типов.

```
    П р и м е р 1.1. Тип: BOOLEAN     /* булев, логический */
BOOLEAN = { TRUE, FALSE }   /* множество констант */
    Операции:
TRUE: -> BOOLEAN;           /* нульарная операция */
FALSE: -> BOOLEAN;          /* нульарная операция */
AND: BOOLEAN x BOLEAN ->BOOLEAN;
OR: BOOLEAN x BOLEAN ->BOOLEAN;
NOT: BOLEAN ->BOOLEAN;
IF_THEN_ELSE: BOOLEAN x  BOOLEAN x BOLEAN ->BOOLEAN;
    Соотношения:  p,q,r - BOOLEAN;
/* операция IF_THEN_ELSE (p,q,r) - это обозначение */
/* логического выражения IF p THEN q ELSE r,       */
/* она будет изображаться как в том, так и в другом*/
/* виде                                            */
IF_THEN_ELSE (TRUE, q, r) = q;
IF FALSE THEN q ELSE r = r;
AND(p,q) = IF p THEN q ELSE FALSE;
OR(p,q) = IF p THEN TRUE ELSE q;
NOT(p) = IF p THEN FALSE ELSE TRUE;
```

В этом примере TRUE и FALSE являются как логическими константами, обозначающими истину и ложь, так и нульарными операциями, выделяющими эти константы в булевом типе данных.

```
П р и м е р 1.2.  Тип: INTEGER      /* целый */
/* множество   константа   типа   INTEGER   зависит */
/* от  языка  транслятора и типа  ЭВМ ; константы  */
```

```
/* выражаются знаком и набором цифр                */
    Операции:
+ : INTEGER x INTEGER -> INTEGER ;   /* сложение */
- : INTEGER x INTEGER -> INTEGER ;   /* вычитание */
* : INTEGER x INTEGER -> INTEGER ;   /* умножение чисел */
DIV : INTEGER x INTEGER -> INTEGER ; /* частное от деления*/
MOD : INTEGER x INTEGER -> INTEGER ; /* остаток от деления*/
ABS : INTEGER -> INTEGER ; /* абсолютное значение числа */
=:  INTEGER x INTEGER -> BOOLEAN; /*отношение равенства */
>:  INTEGER x INTEGER -> BOOLEAN; /*отношение больше */
<:  INTEGER x INTEGER -> BOOLEAN; /*отношение меньше */
```

Деление нацело, обозначаемое словом DIV, дает целую часть частного от деления первого числа на второе.

В этом примере в определении типа INTEGER используется тип BOOLEAN.

```
П р и м е р 1.3. Тип: CHAR     /* литерный, символьный */
/* множество констант типа CHAR - множество  символов в*/
/* некотором порядке; различные типы машин могут иметь */
/* свой набор символов,  но  существует международный  */
/* стандартизованный   набор   ISO   и   американский  */
/*              стандарт ASCII                         */
    Операции:
ORD: CHAR -> INTEGER;  /* порядковый номер символа */
CHR: INTEGER -> CHAR;
    Соотношения: ch - CHAR;    n - INTEGER;
CHR (ORD (ch)) = ch;          /* операции CHR и ORD */
IF ((-1<n) AND (n<128))       /* взаимно обратны    */
   THEN  (ORD (CHR (n)) = n) = TRUE
   ELSE FALSE;
```

В этом определении не только используется уже определенный тип INTEGER, но и вводится на нем новая операция CHR, определяющая символ по его порядковому номеру, и соотношение, использующее операцию IF_THEN_ELSE типа BOOLEAN.

Кроме базисных типов данных в языках программирования высокого уровня имеются средства определения сложных структурированных типов данных, с помощью которых представляются в программе состояния сложных объектов или их совокупностей. К ним относятся конструкции построения перечислимых типов, массивов, множеств, записей, списков и т. д.

Рассмотрим некоторые примеры структурированных типов.

В разных языках форма описания структурированных типов различна. В приведенных ниже примерах приводятся, в основном, конструкции языка PASCAL.

Перечислимый тип задается именем типа и перечислением его элементов. Например, тип: дни_недели = (понедельник, вторник, среда, четверг, пятница, суббота , воскресенье).

Тип массив задается следующей конструкцией:

Имя_типа = ARRAY[k..n] OF тип_элемента.

При этом имеются операции, позволяющие:

определить новый массив;

положить элемент в массив по адресу s, где $k \leq s \leq n$;

получить s-ый элемент массива (тот, который был положен по адресу s в последний раз).

Тип запись задается следующей конструкцией:

```
Имя_типа  RECORD
        имя_поля_1:  тип_1
        ..................
        имя_поля_k:  тип_k
       END
```

Каждый элемент объявленного выше типа мыслится в виде набора полей: имя_поля_1,..., имя_поля_k, соответственно типов: тип_1,...,тип_k. При этом предполагается, что имеются операции построения нового элемента этого типа, присвоения нового значения для каждого поля, выдачи текущего значения любого поля записи.

В некоторых языках имеются конструкции построения множественного типа. В языке PASCAL эта конструкция имеет вид SET OF T, где T может быть скалярным (перечислимым), ограниченным (интервальным) или символьным типом. Значениями переменных множественного типа являются множества значений типа T. Над множествами определены операции

объединения (+), пересечения (*) и вычитания (−). Допускается проверка того, принадлежит ли данное значение X множеству S. Для этого применяется конструкция X IN S, имеющая логическое значение.

В программах обработки символьной информации часто используются списочные типы данных, которые реализуются через стандартные конструкции языка (например, в языках PASCAL, C), либо задаются специальными конструкциями языка (например, в языках LISP, PROLOG).

Ввиду важности этого типа приведем его описание.

```
П р и м е р 1.4. Тип: LIST_OF_T /*список элементов типа T*/
 /* элементы этого типа в LISP имеют вид (t1 t2 ... tn); */
 /* в языке PROLOG тип LIST_OF_T обозначается T* , а его */
 /*             элементы - [t1,t2,...,tn]                */
    Операции:
 NIL: -> LIST_OF_T;          /* пустой список; [] в PROLOGe */
 CONS:T x LIST_OF_T -> LIST_OF_T;/*вставить элемент в список*/
 CAR: LIST_OF_T -> T; /* начальный элемент непустого списка */
 CDR: LIST_OF_T -> LIST_OF_T /* список без первого элемента */
 /*  в PROLOGe операции CAR и CDR задаются выражением  */
 /*        List = [CAR|CDR]                            */
 APPEND:LIST_OF_T x LIST_OF_T -> LIST_OF_T;
                                        /*соединить списки*/
 IS_NIL: LIST_OF_T -> BOOLEAN;
 EQ: LIST_OF_T x LIST_OF_T->BOOLEAN; /* равенство списков */
 Соотношения: t,t1 - T;  list, list1 - LIST_OF_T;
 CAR(CONS(t,list)) =  t;
 CDR(CONS(t,list)) = list;
 APPEND(NIL,list) = list;
 APPEND(CONS(t,list),list1)=CONS(t, APPEND (list,list1));
 IS_NIL(NIL) = TRUE;
 IS_NIL(CONS(t,list)) = FALSE;
 EQ(NIL,NIL) = TRUE;
 EQ(NIL,CONS(t,list)) = FALSE;
 EQ(CONS(t,list),NIL) = FALSE;
 EQ(CONS(t,list),CONS(t1,list1)) = IF (t=t1)
                     THEN EQ(list,list1)  ELSE FALSE;
```

В этом примере приведены основные операции над списками и соот-

ношения между операциями. Реализация типа данных список может быть возложена на трансляторы языка, как например в LISP, PROLOG, или на программистов, как например в языках PASCAL и C. В этом случае реализации могут быть в различной степени правильными (соответствовать описанию) и отличаться эффективностью. По счастью, реализации типа данных список описаны в литературе и имеются в библиотеках примеров программ на этих языках.

Однако, программисты придумывают все новые типы данных (множество, файл, процедура, окно, меню), которые удобны в различных классах задач. С другой стороны, в прикладных областях складываются свои типы данных (матрицы, диаграммы, таблицы, алгебраические выражения, химические формулы, шахматные позиции), которые естественны для этих областей при постановках задачи, описаний алгоритмов, для понимания промежуточных результатов в процессах отладки и доводки алгоритмов.

Таким образом, хотелось бы иметь языковые и программные средства для произвольного расширения системы типов данных в программе и средства, поддерживающие процессы введения новых типов данных и процессы программирования с новыми типами данных. Но сначала нужно сформулировать, что такое тип данных, и каких целей хотят достичь при введении этого механизма.

Из приведенных здесь примеров следует, что тип данных определяется следующими составляющими:

именем — разным именам типов соответствуют разные типы;

системой операций — тем, что с элементами этого типа можно делать;

соотношениями между операциями, выражающими смысл операций;

реализацией — конкретным представлением элементов этого типа и объявленных операций.

Введение в программирование механизма определения произвольных типов данных должен позволить писать и отлаживать программы в общих понятиях (типах и операциях над ними) данной прикладной области или области программирования, не влезая в подробности реализации типа данного.

С другой стороны, отделение реализации типов от их определения и использования позволяет повысить эффективность работы прикладных программ без их переписывания, а только путем усовершенствования реализаций введенных типов. При этом, программы, использующие новые типы, и программы, эффективно реализующие типы, могут писать разные програм-

мисты, специализирующиеся в своих областях. Для стыковки работы этих программистов нужны средства контроля правильности реализации типа данного, то есть средства проверки выполнения в реализации объявленных соотношений между операциями.

1.3 Абстрактные типы данных

Рассмотрим более подробно некоторые математические средства, используемые в литературе по типам данных. В основном, в этом подразделе мы будем следовать [46]

О п р е д е л е н и е 1.1. Многосортной алгебраической системой A называется система, состоящая из трех наборов:

- набора множеств $s_1, ..., s_n$, которые называются несущими множествами алгебраической системы — сортами;
- набора $f_1, ..., f_k$ отображений вида

$$f_i : s_{i1} \times ... \times s_{ni} \to s_{mi}, \text{ для } i = 1, ..., k,$$

которые называются операциями из сортов $s_{i1}, ..., s_{ni}$ в сорт s_{mi};
- набора $r_1, ..., r_l$ подмножеств вида

$$r_j \subset s_{a1} \times ... \times s_{aj}, \text{ для } j = 1, ..., l,$$

которые называются отношениями алгебраической системы.

Многосортной алгеброй называется многосортная алгебраическая система с пустым набором отношений.

Примеры, рассмотренные в предыдущем разделе, показывают, что типы данных являются многосортными алгебрами.

З а м е ч а н и е 1.1. В отличие от моделей баз данных, где отношения играют главную роль, в типах данных отношения принято представлять не подмножествами, а соответствующими им функциями в тип данных BOOLEAN. Таким способом достигается терминологическая однородность: отношения в типах данных мыслятся как операции.

Понятие алгебраической системы было введено в математике и изучалось в связи с математическими задачами [65]. Первоначально работы по изучению алгебраических систем казались экзотическими и слишком абстрактными даже для самих математиков. Использование этого понятия

в теории программирования придало ему образность и обогатило новыми задачами.

При работе с типами данных, представленными в виде многосортных алгебраических систем, программисту-пользователю может быть неизвестна (чаще всего и неважна) конкретная реализация алгебраической системы, но необходимо знание имен её составных частей и знание некоторых свойств операций и отношений, отражающих существо типов данных, с которыми имеют дело. Такое знание программисту достаточно для того, чтобы задать данные и определить способ их обработки в программах, отвлекаясь от несущественных подробностей реализации. В связи с этим, в типах данных используется понятие абстракции данных, то есть переход от множеств, операций и отношений алгебраических систем, соответствующих типам данных, к их именам и соотношениям между ними.

Для определения абстрактного типа данных необходимо ввести понятия сигнатуры и множества определяющих соотношений.

О п р е д е л е н и е 1.2. Сигнатура Σ многосортной алгебры задается двумя наборами:

- набором $S = \{S_1, ..., S_n\}$ имен несущих множеств;
- набором $OP = \{F_1, ..., F_k\}$ имен операций с указанием типа операции вида

$$F_i : S_{i1} \times ... \times S_{ni} \to S_{mi}, \text{ для } i = 1, ..., k.$$

Прежде, чем ввести определение множества определяющих соотношений, необходимо определить множество правильно построенных выражений — термов в сигнатуре Σ.

Пусть $\Sigma = (S, OP)$ — сигнатура. Пусть для каждого сорта S_i из S задано множество переменных X_i. Множество всех переменных X есть объединение всех множеств X_i . Предполагается, что X_i и X_j не имеют общих элементов, если сорта S_i и S_j — различные сорта.

Множество термов $Term_\Sigma(X, S_i)$ сорта S_i с множеством переменных X определяется индуктивно по следующим правилам:

1. Каждая переменная сорта S_i является термом сорта S_i, то есть $X_i \subset Term_\Sigma(X, S_i)$ для всех S_i из S;

2. Каждое имя нульарной операции из множества OP сорта S_i является термом сорта S_i;

3. Для любого имени операции F из OP типа:

$$F : S_{d1} \times ... \times S_{dm} \to S_i,$$

и термов $t_1 \in Term_\Sigma(X, S_{d1})$, ..., $t_m \in Term_\Sigma(X, S_{dm})$ слово $F(t_1, ..., t_m)$ является термом сорта S_i.

Рассмотрим пример сигнатуры и термов в этой сигнатуре.

```
П р и м е р 1.5. Сигнатура типа Nat имеет:
 Сорта: nat   /* т.е. S = { nat } */
 Операции: 0 -> nat;     /* т.е. OP = { 0,succ, add} */
           succ: nat -> nat;
           add: nat x nat -> nat;
```

Пусть $\{m, n\} = X$ — множество переменных типа nat. Тогда слова $succ(0)$, $succ(n)$, $add(m, n)$, $succ(add(m, n))$, $succ(add(0, add(m, m)))$ являются термами типа $Term_{Nat}(X, nat)$, потому что они могут быть построены по правилам 1-3.

Слова $succ(m, n), add(0)$ не являются правильно построенными термами в сигнатуре Nat, так как их нельзя построить по правилам 1-3. Слово $add(x, m)$ не является термом типа $Term_{Nat}(X, nat)$, так как x не входит в множество переменных $X = \{n, m\}$.

Множество всех термов в сигнатуре Σ от переменных, входящих в множество X, обозначается через $Term_\Sigma(X)$, то есть

$$Term_\Sigma(X) = \bigcup_{Si \in S} Term_\Sigma(X, S_i).$$

Итак, мы определили термы, которые представляют собой синтаксические конструкции, выражающие композиции операций сигнатуры от констант и переменных. Поэтому, если переменным терма придать значения в какой либо алгебре с сигнатурой Σ, то и всему терму можно однозначно придать значение, которое получается в результате применения операций в последовательности, указанной в терме.

Более точно. Пусть A — алгебра с сигнатурой $\Sigma = (S, OP)$. Тогда для любого множества переменных X сортов S и любого отображения $ev : X \to A$, сохраняющего сорта, существует продолжение этого отображения в A на все множество термов $Term_\Sigma(X)$, заданное следующим индуктивным правилом:

1. если терм t — переменная из множества X, то $ev(t)$ уже определено; если терм $t = F$— нульарная операция, то $ev(t) = F$ — элемент алгебры A, определенный нульарной операцией F;

2. если терм $t = F(t_1, ..., t_s)$ для некоторой операции F из OP и некоторых термов $t_1, ..., t_s$, то

$$ev(F(t_1, ..., t_s)) = F(ev(t_1), ..., ev(t_s)).$$

Так как множество термов в сигнатуре Σ является алгеброй в этой сигнатуре, то примером такого продолжения отображения является подстановка термов.

Пусть $X = \{x_1, ...x_n\}$ — множество переменных и $x_1 \mapsto b_1, ..., x_n \mapsto b_n$ — сопоставление переменным термов $b_1, ..., b_n \in Term_\Sigma(X)$ того же сорта. Тогда подстановка σ этих термов в произвольный терм $t \in Term_\Sigma(X)$ определяется обычным индуктивным правилом:

1. Если $t = x_i,\ x_i \in X$, то $\sigma(t) = b_i$;

2. Если $t = F()$, где $F() \in OP$ — нульарная операция, то $\sigma(t) = F()$;

3. Если $t = F(t_1, ..., t_n)$, то $\sigma(t) = F(\sigma(t_1), ...\sigma(t_n))$.

В качестве другого примера рассмотрим множество неотрицательных целых чисел как алгебру в сигнатуре Nat, в которой нульарной операции 0 соответствует число 0, операции $succ$ соответствует сопоставление числу n число $n + 1$, а операции add — сумма чисел. Пусть $X = \{n, m\}$ и $ev(n) = 1,\ ev(m) = 5$. Тогда по определению
$ev(add(succ(n), m)) = add(ev(succ(n), ev(m)) = add(succ(ev(n)), 5) =$
$= add(succ(1), 5) = add(2, 5) = 7.$

О п р е д е л е н и е 1.3. Уравнением в сигнатуре $\Sigma = (S, OP)$ называется тройка $e = (X, L, R)$, где X — множество переменных, L, R —термы одного сорта из множества $Term_\Sigma(X)$ в сигнатуре Σ. Иногда, для краткости, уравнение e мы будем записывать также в виде $L = R$, если множество переменных X очевидно. Уравнение выполняется в Σ–алгебре A, если для любого отображения $ev : X \to A$ выполняется равенство $ev(L) = ev(R)$ в A. Если уравнение e выполняется в алгебре A, то говорят также, что алгебра A удовлетворяет уравнению e.

Теперь мы готовы к тому, чтобы дать определение абстрактного типа данных.

О п р е д е л е н и е 1.4. Абстрактным типом данных $\mathcal{A}$ называется тройка $\mathcal{A} = (Name, \Sigma, E)$, где $Name$ — имя типа, $\Sigma = (S, OP)$ — сигнатура типа данного, $E = \{e_1, ..., e_u\}$ — определяющие соотношения абстрактного типа данных (конечное множество уравнений в сигнатуре Σ).

Мы уже видели, что именование является важным средством построения новых типов. В сигнатуре отражаются имена операций этого типа данных, а в соотношениях требуемые программистом - пользователем свойства операций.

```
П р и м е р 1.6. Абстрактный тип данных Nat.
 Сорта: nat   /* т.е. S = { nat } */
 Операции: 0 -> nat;     /* т.е. OP = { 0,succ, add} */
           succ: nat -> nat;
           add: nat x nat -> nat;
 Соотношения: /* множество E */
           add(0,n) = n;
           add(n,0) = n;
           add(succ(n),m) = succ((add(n,m));
           add(n,succ(m)) = succ((add(n,m));
```

Реализацией абстрактного типа данных называется сопоставление именам несущих множеств конкретные множества, именам операций конкретные операции на этих множествах.

О п р е д е л е н и е 1.5. Синтаксически корректной реализацией абстрактного типа данных $\mathcal{A} = (Name, \Sigma, E)$ называется произвольная алгебра A с сигнатурой Σ . Семантически корректной реализацией абстрактного типа данных $\mathcal{A}$ называется алгебра A , удовлетворяющая уравнениям множества E.

Семантически корректная реализации называется также моделью абстрактного типа данных $\mathcal{A}$. У одного абстрактного типа данных может быть несколько моделей.

О п р е д е л е н и е 1.6. Пусть A и A' две модели абстрактного типа данных $\mathcal{A}$. Гомоморфизмом $j : A \to A'$ моделей (алгебр) называется отображение j несущих множеств A в несущие множества A', сохраняющее сорта элементов и обладающее следующим свойством: для любого F

из множества имен операций OP абстрактного типа данных $\mathcal{A}$ и любого элемента модели A вида $F(a_1, ..., a_s)$ выполнено равенство:

$$j(F(a_1, ..., a_s)) = F(j(a_1), ..., j(a_s)).$$

Если, кроме того, j является взаимно однозначным отображением, то такой гомоморфизм называется изоморфизмом.

П р и м е р ы 1.7. (Реализации, гомоморфизм, изоморфизм)
Рассмотрим следующие реализации абстрактного типа данных Nat с сигнатурой $\Sigma = (nat; 0, succ, add)$. Это натуральные числа N и целые числа $Integer$. При этом, сорту nat сопоставляются множества N и $Integer$, соответственно, а именам операций $0, succ, add$ сопоставляются операции $0, +1, +$ в натуральных и целых числах. Нетрудно видеть, что в этих реализациях выполняются все соотношения абстрактного типа данных Nat, то есть это корректные реализации, и естественное вложение N в $Integer$ совместимо с действием операций, то есть это вложение — гомоморфизм моделей Nat. Так как вложение N в $Integer$ не является взаимно однозначным отображением, то этот гомоморфизм не изоморфизм. Обратное отображение $j : Integer \to N$, сопоставляющее целому числу его абсолютное значение, не гомоморфизм, так как $j(3 + (-3)) = 0$ и не равно $j(3) + j(-3) = 6$.

Другую модель Nat, изоморфную N, дает следующая реализация. Сорту nat сопоставляется множество всех слов A^* в однобуквенном алфавите $A = \{a\}$. Именам операций $0, succ, add$ сопоставляются, соответственно, пустое слово, приписывание к слову одной буквы, конкатенация слов. Отображение $\_ : A^* \to N$ — взаимно однозначный гомоморфизм, то есть изоморфизм.

Прикладные программисты, как правило, не отличают две изоморфные реализации абстрактного типа данных, потому что все, что выполняется и может быть построено в одной реализации, выполняется и может быть построено в другой реализации (хотя может отличаться эффективностью выполнения операций и объемами памяти, занимаемого данными). С этим свойством связано следующее понятие абстракции данных.

О п р е д е л е н и е 1.7. Абстрактной структурой данных называется модель абстрактного типа данных, заданная с точностью до изоморфизма.

Для построения данных и определения способов их обработки в программах должны указываться лишь имена, входящие в определение абстрактного типа данных, без ссылок на конкретные свойства реализации

типа. Это делает возможным использование одной и той же программы для работы с разными реализациями абстрактного типа данных. Так обеспечивается свойство, которое программисты называют независимостью работы программы от физического представления данных.

Таким образом, в описаниях (спецификациях) абстрактных типов данных обязательно задаются имя типа, сигнатура и система соотношений между операциями, то есть дается описание абстрактного тапа данных. Кроме того, иногда в спецификации указывается, а иногда подразумевается некоторая абстрактная структура данных, то есть некоторая реализация абстрактного типа данных с точностью до изоморфизма. Это делается в тех случаях, когда программист хочет использовать свойства данных из некоторой абстрактной структуры данных, не ограничивая себя только теми свойствами, которые следуют из множества определяющих соотношений соответствующего абстрактного типа данных.

Распространенный прием задания абстрактных структур данных — это задание их в виде инициальных моделей [50].

О п р е д е л е н и е 1.8. Инициальная модель $I(\mathcal{A})$ абстрактного типа данных $\mathcal{A}$ — это модель, обладающая следующим универсальным свойством: для любой модели A абстрактного типа данных $\mathcal{A}$ существует единственный гомоморфизм из модели $I(\mathcal{A})$ в модель A.

У т в е р ж д е н и е 1.1. Если $I(\mathcal{A})$ и $I'(\mathcal{A})$ — инициальные модели абстрактного типа данных $\mathcal{A}$, то они изоморфны.

Д о к а з а т е л ь с т в о. Пусть $I(\mathcal{A})$ и $I'(\mathcal{A})$ — инициальные модели абстрактного типа данных $\mathcal{A}$. Тогда, по определению инициальности существуют гомоморфизмы:

$$h : I(\mathcal{A}) \to I'(\mathcal{A}) \text{ и } h' : I'(\mathcal{A}) \to I(\mathcal{A}).$$

Композиция гомоморфизмов h и h' является гомоморфизмом из $I(\mathcal{A})$ в $I(\mathcal{A})$. Но из определения инициальности следует, что гомоморфизм из $I(\mathcal{A})$ в $I(\mathcal{A})$ только один, и, следовательно, эта композиция совпадает с тождественным отображением на $I(\mathcal{A})$. Аналогично показывается, что композиция h' и h является тождественным отображением на $I'(\mathcal{A})$. Отсюда следует, что h — взаимно однозначное отображение и, следовательно, — изоморфизм.

Т е о р е м а 1.2.[50] Для любого абстрактного типа данных с соотношениями типа уравнений существует инициальная модель.

Доказательство теоремы проводится стандартным образом. На множестве термов без переменных в сигнатуре абстрактного типа данных индуктивно строится отношение эквивалентности, удовлетворяющее естественным свойства конгруэнтности, исходя из соотношений абстрактного типа данных. Инициальная модель представляется в виде фактор множества таких термов по построенному отношению эквивалентности.

Пусть $\mathcal{A} =< Name, \Sigma, E >$ — произвольный абстрактный тип данных. Рассмотрим множество термов $Term_\Sigma(\emptyset)$ с пустым множеством переменных. Тогда, для любой модели A абстрактного типа данных $\mathcal{A}$ существует единственное отображение

$$ev : Term_\Sigma(\emptyset) \to A,$$

сопоставляющее каждому терму его значение в алгебре A.

Обозначим через $eq_\mathcal{A}$ множество всех пар термов (t_1, t_2) таких, что уравнение $t_1 = t_2$ выполняется во всех моделях A, то есть $ev(t_1) = ev(t_2)$ верно во всех моделях $\mathcal{A}$.

У т в е р ж д е н и е 1.3. Отношение $eq_\mathcal{A}$ обладает следующими свойствами:

1. (Рефлексивность.) Для любого терма $t \in Term_\Sigma(\emptyset)$ пара $(t, t) \in eq_\mathcal{A}$;
2. (Симметричность.) Если $(t_1, t_2) \in eq_\mathcal{A}$, то $(t_2, t_1) \in eq_\mathcal{A}$;
3. (Транзитивность.) Если (t_1, t_2) и $(t_2, t_3) \in eq_\mathcal{A}$, то $(t_1, t_3) \in eq_\mathcal{A}$;
4. Если $(t_1, t'_1), ..., (t_n, t'_n) \in eq_\mathcal{A}$ и $F(t_1, ..., t_n)$ —терм для некоторого $F \in OP$, то $(F(t_1, ..., t_n), F(t'_1, ..., t'_n)) \in eq_\mathcal{A}$;
5. Если соотношение $(X, L, R) \in E$ и $\sigma : X \to Term_\Sigma(\emptyset)$ — произвольное означивание переменных термами того же типа, то пара $(\bar{\sigma}(L) = \bar{\sigma}(R)) \in eq_\mathcal{A}$.

Доказательство утверждения следует из определений и очевидно.

О п р е д е л е н и е 1.9. Отношение, обладающее свойствами 1–3, называется отношением эквивалентности. Отношение, обладающее свойствами 1–4, называется конгруэнцией. Отношение, обладающее свойствами 1–5, называется конгруэнцией, удовлетворяющей равенствам E. Таким образом, отношение $eq_\mathcal{A}$ — отношение конгруэнции на множестве термов $Term_\Sigma(\emptyset)$, удовлетворяющее соотношениям E.

О п р е д е л е н и е 1.10. Обозначим через $\cong_E$ отношение эквивалентности на множестве термов $Term_\Sigma(\emptyset)$, построенное по правилам 1–5, то есть $\cong_E$ — наименьшее отношение, обладающее свойствами 1–5. Если терм $t \in Term_\Sigma(\emptyset)$, то через $[t]$ обозначается множество всех термов, эквивалентных t по отношению $\cong_E$, то есть $[t] = \{t' : t' \cong_E t\}$.

У т в е р ж д е н и е 1.4. Для любых двух термов $t, t' \in Term_\Sigma(\emptyset)$ множества $[t_1]$ и $[t_2]$ либо совпадают, либо не пересекаются.

Утверждение верно для любого отношения эквивалентности. Действительно, если $[t_1]$ и $[t_2]$ имеют общий элемент t, то, по определению, $t \cong_E t_1$ и $t \cong_E t_2$. Отсюда, из свойств симметричности и транзитивности следует, что $t_1 \in [t_2]$, $t_2 \in [t_1]$ и $[t_1] = [t_2]$.

Теперь мы готовы к доказательству теоремы существования инициальной алгебры.

В качестве множества элементов алгебры $I(\mathcal{A})$ рассмотрим множество:

$$I(\mathcal{A}) = \{[t] : t \in Term_\Sigma(\emptyset)\},$$

которое называют фактор-множеством множества термов по конгруэнции $\cong_E$.

Операции в этом множестве действуют по следующим правилам:

1. Если $F \in OP$ — нульарная операция,то ее действием будет элемент $[F]$ в $I(\mathcal{A})$;

2. Если $F(x_1, ..., x_n) \in OP$ — n-арная операция и $[t_1], ..., [t_n] \in I(\mathcal{A})$, то $F([t_1], ..., [t_n])$ полагается равным $[F(t_1, ..., t_n)] \in I(\mathcal{A})$ (корректность этого определения обеспечивается свойством 4).

Таким образом, $I(\mathcal{A})$ — Σ - алгебра. Более того, благодаря свойству 5, $I(\mathcal{A})$ — модель абстрактного типа данного $\mathcal{A}$. Докажем, что эта модель инициальна.

Пусть A — произвольная модель абстрактного типа данного $\mathcal{A}$. Тогда, как было показано выше, существует единственное отображение $ev : Term_\Sigma(\emptyset) \to A$, совместимое с действием операций. При этом, если $t \cong_E t'$, то $ev(t) = ev(t')$, так как в алгебре A выполняются соотношения E абстрактного типа данных $\mathcal{A}$. Поэтому отображение $j : I(\mathcal{A}) \to A$, заданное соотношением $j([t]) = ev(t)$, определено корректно. Так как отображение ev по определению перестановочно с именами операций, то j — гомоморфизм. Его единственность следует из единственности отображения ev, перестановочного с именами операций.

Следствиями этой теоремы и приведенных построений являются следующие утверждения.

Т е о р е м а 1.5. Уравнение $t_1 = t_2$ для термов $t_1, t_2 \in Term_\Sigma(\emptyset)$ выполняется во всех моделях абстрактного типа данных $\mathcal{A}$ тогда и только тогда, когда оно может быть получено по правилам 1–5, то есть, когда оно выполняется в инициальной модели.

Т е о р е м а 1.6. Отношения $eq_\mathcal{A}$ и $\cong_E$ совпадают.

1.4 Фрагменты схем баз данных как примеры понятий

В настоящее время возникают базы данных со сложно организованными схемами. К таким базам данных относятся распределенные базы данных с большим числом локальных баз данных, научные базы данных со сложной организацией областей, базы данных со сложно организованными объектами.

В таких базах данных, прежде чем обратиться к самим данным, необходим поиск по схеме базы данных для определения места и структуры требуемых пользователю данных. Знания о схеме базы данных требуется для составления запроса к базе данных. Учитывая сложность и величину схемы базы, для удобства просмотра и составления запросов схема базы данных разбивается по смыслу на фрагменты.

Каждый фрагмент представляет собой осмысленную часть схемы базы данных, которая может быть использована в других фрагментах схемы и в запросах. Спецификация различных фрагментов базы данных может производиться различными людьми — специалистами в областях знаний, к которым относятся данные фрагменты. Более того, некоторые фрагменты схем баз данных, являются стандартами, общеизвестными специалистам и используемыми как в базах данных, так и в запросах. Использование стандартных фрагментов является существенным требованием при проектировании схем современных баз данных.

Рассмотрим некоторые примеры спецификаций фрагментов схем баз данных.

П р и м е р 1.8.

Имя фрагмента: PERSON

Образующие:

PersonIdentityNo : PERSON $\rightarrow$ INT;
Name: PERSON $\rightarrow$ STRING;
Age: PERSON $\rightarrow$ INT;
Children: PERSON $\rightarrow$ Set_of(PERSON);
Соотношения: Var p_1, p_2 : PERSON;
IF PersonIdentityNo(p_1) = PersonIdentityNo (p_2) THEN
$p_1 = p_2$.

Имя фрагмента: STUDENT
Образующие:
Isa: STUDENT $\hookrightarrow$ PERSON;
GroupNamber: STUDENT $\rightarrow$ INT;
Advisor: STUDENT $\rightarrow$ PROF;
.....

Имя фрагмента: PROF
Isa: PROF $\hookrightarrow$ PERSON;
.....

В этих примерах используются стандартные типы данных INT, STRING. В общем случае, в описаниях фрагментов схем баз данных могут использоваться произвольные типы данных, включая структурированные типы и абстрактные типы данных. В приведенных фрагментах, как и в абстрактных типах данных, вводятся имена областей: PERSON, STUDENT, PROF; имена функций: PersonIdentityNo, Name, Age, Advisor, отражающие свойства элементов соответствующих областей; соотношения. Кроме того, в фрагменте PERSON используется конструкции Set_of и рекурсивная конструкция Set_of(PERSON). В фрагментах STUDENT и PERSON используется конструкция Isa:$T_1 \hookrightarrow T_2$, определяющая область T_1 как подобласть области T_2.

Другие примеры дают обычные схемы реляционных баз данных, рассмотренные в предыдущей главе. В этом случае, схема базы данных задается набором доменов $D_1, ..., D_n$, набором схем отношений

$$R^1(a^1_1, ..., a^1_{n_1}),\ ...,\ R^k(a^k_1, ..., a^k_{nk})$$

(они называются базисными), где $R^1, ..., R^k$ — имена отношений, $^1_1, ..., a^k_{nk}$

имена атрибутов, для каждого из которых отображением

$$t : \{{}^1_1, ..., a^k_{nk}\} \to \{D_1, ..., D_n\}$$

определен домен значений. Кроме того, задаются отображения вложения

$$R^j(a^j_1, ..., a^j_{nj}) \hookrightarrow D_{t(a_1)} \times ... \times D_{t(a)}, \text{ for } j = 1, ..., k$$

и соотношения (зависимости, ограничения целостности базы данных) между этими отношениями вида:

$$\begin{array}{rcl} f_1(R^1, \ldots, R^k) & = & g_1(R^1, \ldots, R^k) \\ \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots \\ f_l(R^1, \ldots, R^k) & = & g_l(R^1, \ldots, R^k), \end{array}$$

где f_i и g_i — операции над отношениями, выраженные в виде композиций основных операций.

В частности, в виде подобных соотношений можно записать условие функциональной зависимости какого–либо атрибута от набора других, вхождение одного отношения в другое как подмножества, пустоту пересечения каких–либо отношений и т.д. Соотношения, зафиксированные в базе данных, отражают условия целостности базы данных, которые обязательно должны выполняться, отражают смысл представленных данных и используются в приложениях.

1.5 Примеры представления общих понятий

В этом разделе рассмотрим некоторые примеры понятий из научных областей знаний и некоторые принципы их представления для автоматизированной обработки.

Нас будут интересовать средства представления понятий, ориентированные, главным образом, не на программирование отдельных задач, а на создание автоматизированной информационной системы о знаниях в некоторой предметной области и, затем, на автоматизированную поддержку постановок и решений многочисленных задач, для формулировок которых используются понятия, представленные в этой системе. При этом знания предметной области представляются в виде системы понятий, в которой

при определении одних понятий могут использоваться другие, ранее определенные в системе понятия.

Естественно, что одну и ту же предметную область можно по-разному структурировать, и у определений понятий могут быть разные эквивалентные (или не совсем эквивалентные) формы. Выбор структуризации предметной области и форм определений при представлении знаний предметной области средствами языка представления должно, в основном, определяться традициями в этой предметной области и искусством проблемного специалиста, представляющего знания.

С этой точки зрения задача представления знаний для автоматизированной обработки аналогична задаче подготовки учебника по специальности представляемой области знания. Использование традиций данной предметной области при представлении знаний необходимо для облегчения восприятия представленных знаний и для удобства использования опыта специалистов в предметной области при обращении к системе. Система представления знаний должна организовывать естественную среду знаний для специалистов проблемной области. Это тем более важно в случаях, когда представлением знаний занимается одна группа проблемных специалистов, а использованием представленных в системе знаний — другая группа специалистов.

Структуризация представления знаний обеспечивает возможность работы с обозримыми фрагментами представленных знаний. Она выбирается также, исходя из удобства выражения знаний, постановок задач, проверки правильности выражений и локализации ошибок. Система понятий должна быть достаточно полной, обеспечивающей эффективное выражение постановок типичных задач данной предметной области. Кроме того, структуризация представления знаний и система понятий строится так, чтобы обеспечить устойчивость системы понятий при изменениях в определениях отдельных понятий.

С точки зрения проблемного специалиста средства языка представления служат для выражения определений понятий проблемной области, для выражения доказательств некоторых свойств введенных понятий, для постановок задач и задания запросов к системе представленных знаний. С другой стороны, выражения на языке представления имеют внутреннюю интерпретацию, которая может быть скрыта от пользователя и служит для организации эффективного получения ответов на запросы и проверки непротиворечивости введенных спецификаций понятий.

Программирование с использованием языка представления понятий состоит из следующих процессов:

формулировка и ввод определений понятий предметной области;

формулировка и доказательство некоторых свойств понятий, которые не входят в определение понятий, но известны специалисту и по его мнению должны быть отражены в системе;

постановка задач проблемной области с использованием введенных понятий.

Процесс программирования в такой системе — это диалог, в котором специалист формулирует поставки задач проблемной области, используя уже введенные понятия как модули, или вводит новые понятия и их свойства.

Пусть, к примеру, наша предметная область — это понятия школьного учебника физики. Первый пример соответствует понятию механическое движение.

Имя понятия: Движение.
ПРОСТРАНСТВО, ВРЕМЯ, ТЕЛА — области.
Место — отображение.
Место: ТЕЛА $\times$ ВРЕМЯ $\rightarrow$ ПРОСТРАНСТВО.

По этому определению каждое механическое движение требует наличия трех областей, которые называются ПРОСТРАНСТВО, ВРЕМЯ, ТЕЛА и отображения Место, которое каждому телу в каждый момент времени задает положение в пространстве. Например, в движениях по прямой область ПРОСТРАНСТВО представляет собой прямую, в движениях по окружности — окружность, в движениях в трехмерном пространстве — трехмерное пространство. Область ВРЕМЯ обычно представляется множеством действительных чисел, но в задачах с дискретном временем может быть представлена множеством целых чисел. Область ТЕЛА в разных задачах также может быть разной. В задачах о движении одного тела она состоит из одного тела, а в задачах о движении n тел область ТЕЛА состоит из n элементов. В любом случае, когда речь идет о механическом движении, то выделяются области ПРОСТРАНСТВО, ВРЕМЯ, ТЕЛА и подразумевается, что есть отображение, задающее место в пространстве для каждого тела в каждый момент времени.

При представлении знаний, исходим из следующих требований к системе представлений:

модульность — возможность формирования новых определений и запросов на основе уже введенных в систему;

открытость — возможность ввести в любой момент новые определения и запросы и использовать их затем как модули;

понятность — близость формальных выражений на языке представления исходным формулировкам определений и запросов на естественном языке;

возможность использования сокращений и условных обозначений для понятий и формул;

способность интерпретировать неполную или противоречивую информацию.

Проиллюстрируем некоторые из этих принципов примером.

Имя понятия: Равномерное движение Т.
Другое имя: Т равномерно движется.
Это: Движение. Используются: ЧИСЛА.
Образующие:
Т — элемент области ТЕЛА.
СКОРОСТЬ — элемент области ЧИСЛА.
Промежуток времени: ВРЕМЯ$\times$ВРЕМЯ $\rightarrow$ ЧИСЛА.
Если t_1, t_2 — ВРЕМЯ, то Длина пути(t_1, t_2) — ЧИСЛА.
Сокращения:
t = ПРОМЕЖУТОК ВРЕМЕНИ.
S = ДЛИНА ПУТИ.
V = СКОРОСТЬ.
Соотношения:
Если t_1, t_2 — ВРЕМЯ, то $S(t_1, t_2) = V * t(t_1, t_2)$.
Если t_1, t_2, t_3 — ВРЕМЯ, то $t(t_1, t_3) = t(t_1, t_2) + t(t_2, t_3)$.
Утверждения:
Если t_1, t_2 — ВРЕМЯ, то $V = S(t_1, t_2)/t(t_1, t_2)$ и
$t(t_1, t_2) = S(t_1, t_2)/V$

Здесь определяется понятие с двумя синонимичными именами "Равномерное движение Т" и "Т равномерно движется". В этом примере определяется понятие с выделенным параметром Т .

Использование в определении имен ранее построенных понятий или терминов, входящих в ранее построенные понятия, означает, что тексты определений и внутренние представления этих понятий включаются в текст и внутреннее представление формируемого понятия. В данном случае это

означает, что внутреннее представление понятия "Равномерное движение Т" строится из внутренних представлений ранее определенного понятия "Движение" и базисного понятия "ЧИСЛА".

Здесь не приводится формального определения понятия "ЧИСЛА", но предполагается, что по определению оно содержит область "ЧИСЛА", на которой определены операции сложения $(+)$, вычитания $(-)$, умножения $(*)$, деления $(/)$ удовлетворяющие стандартным аксиомам поля. Предполагается также, что аппроксимация понятия "ЧИСЛА" содержит машинные числа с соответствующими операциями на них.

При построении представления понятия "Равномерное движение" к образующим и соотношениям понятий "Движение" и "ЧИСЛА" добавляются новые преобразования, "Промежуток времени", "Длина пути" и элементы "Т" и "Скорость". При этом область определения и область значений преобразователя "Промежуток времени" указаны явно, а область определения и область значений преобразования:

$$\text{Длина пути: ВРЕМЯ}\times\text{ВРЕМЯ} \to \text{ЧИСЛА}$$

задается с помощью оператора языка "Если ..., то ...".

Выражения " Т — элемент области ТЕЛА" и "СКОРОСТЬ — элемент области ЧИСЛА" означают , что вводимые в определении термины " Т " и "скорость" являются элементами, соответственно, областей "ТЕЛА" и "ЧИСЛА".

Сокращения t, S, V являются сокращенными синонимами для имен соответствующих преобразований.

Дополнительные определяющие соотношения понятия "Равномерное движение" представляют собой равенства между композициями некоторых преобразований и также задаются с помощью оператора языка "Если ... , то ... ".

Равенства, входящие в "утверждения" определения, могут быть получены (выведены) из образующих и определяющих соотношений понятия "Равномерное движение", но вводятся сюда для того, чтобы они вошли во во внутреннее представление этого понятия.

Используя введенные понятия, может быть сформулирована следующая задача.

Задача:
Автомобиль равномерно движется.

t_1, t_2, t_3 — элементы области ВРЕМЯ.
t_1 =Время начала движения.
t_2 =Время замера.
t_3 =Время конца движения.
$t(t_1, t_2) = 10$. $S(t_1, t_2) = 5$. $S(t_1, t_3) = 15$.
Чему равно $t(t_1, t_3)$?

Тексту этой задачи сопоставляется внутреннее представление, которое строится из внутреннего представления понятия "Т равномерно движется", где вместо параметра Т подставлен термин "Автомобиль", добавлены элементы t_1, t_2, t_3 в области ВРЕМЯ и соотношения $t(t_1, t_2) = 10$, $S(t_1, t_2) = 5$, $S(t_1, t_3) = 15$. Вопрос задачи интерпретируется как запрос к внутреннему представлению задачи. В ответ выдается элемент области "ЧИСЛА", равный $t(t_1, t_3)$.

Можно было бы сформулировать и другие задачи с несколькими телами и другими вопросами. Можно было бы ввести понятия равноускоренного движения и задачи на равноускоренное движение по аналогии с приведенными. Если бы мы захотели, чтобы система отслеживала единицы измерения физических величин, то нужно было ввести понятия единицы измерения, величины времени, величины длины и потребовать, чтобы преобразования "Промежуток времени" и "Длина пути" принимали значения, соответственно, в областях "ВЕЛИЧИНА ВРЕМЕНИ" и "ВЕЛИЧИНА ДЛИНЫ", а не в области "ЧИСЛА".

В языке представления кроме понятий, которые используются для формулировки других понятий, можно определить схемы вопросов, которые могут быть использованы для формулировки других вопросов или вопросов задач. Примерами таких определений являются следующие схемы вопросов "где?" и "когда?".

Имя понятия: Где Т?
Образующие:
ТЕЛА, ПРОСТРАНСТВО — области.
Положение: ТЕЛА $\rightarrow$ ПРОСТРАНСТВО.
Т — элемент области ТЕЛА.
Чему равно Положение (Т)?

Имя понятия: Когда S?
Образующие:

СОБЫТИЯ, ВРЕМЯ — области.
Время: СОБЫТИЯ $\rightarrow$ ВРЕМЯ.
S — элемент области СОБЫТИЯ.
Чему равно Время (S)?

Чтобы применить эти вопросы в конкретной задаче или понятии, нужно проинтерпретировать схемы вопросов в представлении задачи или понятия.

Итак, перечислим некоторые функции системы представления понятий.

Система строит внутреннее представление понятия по выражениям языка представления, описывающим данное понятие или задачу. Строит запрос к представлению, соответствующий вопросу задачи или схеме вопроса. Кроме того, система осуществляет логический контроль за правильностью написания выражений на языке представления прикладным специалистом и может задавать вопросы пользователю для уточнения интерпретации текста представления, а также постоянно проводит контроль на непротиворечивость представления, уже построенного системой. Система автоматически расширяет текст пользователя за счет текстов модулей, указание на которые делает пользователь, а также выполняет необходимые справочные функции о текстах, введенных в систему.

2 Категорный подход к представлению понятий

2.1 Общее описание подхода

В настоящее время одним из основных средств, которое применяется для представления декларативных знаний (понятий) в системах искусственного интеллекта являются семантические сети. Семантическая сеть представляет собой сеть с поименованными элементами. В методе представления знаний семантическими сетями предполагается, что всякое декларативное знание можно закодировать в виде сети.

Такой подход полностью аналогичен сетевому подходу к моделированию баз данных. Он имеет те же достоинства и недостатки. Основной недостаток заключается в том, что из-за закодированности информации внешняя логика понятий сложно представляется, и сама информация становится трудно доступной проблемному специалисту.

Альтернативой семантическим сетям для представления понятий является категорный подход, использующий аппарат и методы математической теории категорий для определения семантики логических конструкций. Соотношение между подходом, использующим семантические сети, и предлагаемым почти такое же, как между сетевыми и реляционными подходами к моделированию баз данных. Здесь, как и в реляционном подходе, предполагается, что понятие (базу данных) можно представить в виде набора областей, системы отображений и отношений, но имеются меньшие требованиями к полноте представляемой информации.

Категорный подход к представлению знаний является естественным развитием реляционного подхода к базам данных, дополненного идеями и методами абстрактных типов данных в программировании [3],[72],[50],[41], [53],[54]. При этом используются достижения приложений математической теории категорий к математической логике [64],[16].

В категорном подходе к представлению знаний предполагается, что представляемые знания прикладной области могут быть структурированы в виде системы понятий. С каждым понятием связывается имя понятия, определение и знания, вытекающие из определения. Теория категорий предоставляет математические средства для отражения семантики и логики понятий. Каждому представляемому понятию в этом подходе сопоставля-

ется алгебраическая модель понятия (категория), в которой потенциально отражается полное знание о предмете понятия, вытекающее из определения этого понятия, а также строится конечная аппроксимация категории (фрагмент категории), в которой отражаются уже имеющиеся знания о представляемом понятии, отображенные в системе представления и непосредственно доступные пользователю.

Средства категорного подхода к представлению знаний условно можно разделить на три основные части:

- средства построения алгебраической модели (категории);
- средства построения конечной аппроксимации категории;
- язык представления знаний.

Все три составляющие подхода сильно связаны между собой:

- категория и ее аппроксимация задаются языковыми средствами;
- корректность новых языковых выражений определяется по уже построенной аппроксимации;
- категория, отражающая полные знания о представляемом понятии, — это идеальный объект, который является пределом своих конечных аппроксимаций.

В основе категорного подхода к представлению знаний так же, как и в основе реляционного подхода к моделированию баз данных, лежит некоторый набор операций. Мы будем называть их категорными операциями, так как они были введены в математической теории категорий (см.,например, [16]).

Категорные операции позволяют выразить представляемые понятия и определить требуемую их обработку. Особенностью категорных операций является возможность построения любых теоретико множественных конструкций из областей, не все элементы которых известны.

Единая категорная интерпретация понятий, основанная на общем наборе операций, позволяет достичь высокой степени интеграции понятий в систему знаний.

В данной работе дается описание предлагаемых средств категорного подхода к представлению понятий. Некоторые из представленных здесь результатов были опубликованы автором в [8], [9], [10], [11].

В настоящее время разработан макет диалоговой системы, основанный на категорном подходе, для создания баз понятий. Макет разработан на языке PDC Prolog и функционирует на персональных ЭВМ типа IBM PC AT.

2.2 Категорные средства представления понятий

В этом разделе неформально описываются некоторые средства математической теории категорий, служащие для представления понятий.

Почему именно категории выбраны для представления понятий и что это такое?

Основным тезисом категорного подхода к представлению знаний является предположение, что определение всякого понятия может быть представлено в виде наборов имен областей, имен преобразований и соотношений. При этом, не все элементы областей могут быть известны внутри данного понятия и могут уточняться в других понятиях.

Широта принятого тезиса подтверждается тем, что схемы базы данных и абстрактные типы данных являются примерами таких понятий. Эксперименты показывают, что так представляются понятия таких предметных областей, как механика, химия и т.д.

Уточним теперь, какими свойствами обязательно обладают совокупности имен областей и отображений независимо от того, какое понятие ими представляется.

Так как именам отображений в конкретных примерах соответствуют отображения одних множеств в другие, то выполняются следующие свойства.

Если $f_1 : T_1 \to T_2$ и $f_2 : T_2 \to T_3$ — имена отображений такие, что $\_(f_1) = \_(f_2)$, то строится имя $f_2 *$
f_1 отображения из T_1 в T_2, которое называется композицией отображений f_1 и f_2 .

Композиция отображений обладает свойством ассоциативности. Это значит, что для любых отображений f_1, f_2 и любого отображения $f_3 : T_3 \to T_4$ выполнено равенство

$$f_3 * (f_2 * f_1) = (f_3 * f_2) * f_1. \qquad (1)$$

Для отображений множеств равенство (1) легко проверяется на элементах.

Кроме того, для любой области T можно ввести тождественное отображение $id(T) : T \to T$, композиция которого с любыми отображениями $f : T_1 \to T$ и $g : T \to T_2$ дает следующие равенства:

$$id(T) * f = f, \quad g * id(T) = g. \qquad (2)$$

Математическая структура, обладающая операцией типа операции композиция со свойствами (1) и (2), называется категорией [16]. Таким образом, система имен областей и отображений, которая сопоставляется представляемому понятию и удовлетворяет условиям (1) и (2), — это категория. Объектами этой категории являются имена областей, а морфизмами — имена отображений, рассмотренные с точностью до равенства.

Обозначим эту категорию через $\mathcal{C}$. Объекты этой категории мы будем называть областями. Множество всех морфизмов из области $_1$ в область $_2$ категории $\mathcal{C}$ обозначается через $\mathcal{C}(T_1, T_2)$.

Для указания элементов объектов в теории категорий имеется специальный способ, который естественно использовать и в представлении знаний.

Вводится особая область ТОЧКА, которая сокращенно обозначается через I, обладающая следующим свойством: для каждой области T из категории $\mathcal{C}$ существует единственный морфизм из T в I, который обозначается через $I(T) : T \to I$. Это свойство полностью характеризует область ТОЧКА. В теории категорий ее называют финальным объектом.

Полезно также иметь пустую область $\emptyset$, которая задается следующим свойством: для любой области категории $\mathcal{C}$ множество всех морфизмов $\mathcal{C}(\emptyset, T)$ состоит из единственного морфизма $\emptyset(T) : \emptyset \to T$. В теории категорий такой объект называют инициальным.

В дальнейшем будет предполагаться, что рассматриваемые здесь категории $\mathcal{C}$ имеют области $\emptyset$ и I.

Категория областей $\mathcal{C}$ называется непротиворечивой, если область $\emptyset$ неизоморфна области I. Соответственно, если понятие представляется непротиворечивой категорией, то представление называется непротиворечивым.

Пусть — некоторая область из категории $\mathcal{C}$. Известные в представлении элементы области T задаются морфизмами $t : I \to T$ из области ТОЧКА в область T. Множество морфизмов $\mathcal{C}(I, T)$ соответствует множеству всех известных элементов области T в категории $\mathcal{C}$.

Если $f : T_1 \to T_2$ — морфизм, то он индуцирует отображение известных элементов области T_1 в известные элементы области T_2. Действительно, пусть $t : I \to T_1$ — произвольный элемент области T_1, тогда композиция морфизмов $f * t : I \to T_2$ — элемент области T_2.

В общем случае возможно, что два разных морфизма индуцируют одно и то же отображение на множестве известных элементов. Таким образом, область в категорном подходе не полностью характеризуется множеством

ее известных элементов. В общем случае область определяется взаимоотношениями со всеми областями категории $\mathcal{C}$, а не только взаимоотношениями с областью ТОЧКА. Это означает, что области в категорном подходе это не множества. Их можно себе представлять как недостроенные множества.

Более того, в приведенном ранее примере определения движения области ПРОСТРАНСТВО, ВРЕМЯ, ТЕЛА вообще не имеют известных элементов, так как их нельзя получить из этого определения. Это распространенная ситуация при представлении общих понятий, в которых неполнота является важным свойством. Такие понятия в разных случаях наполняются разным содержанием. Например, понятие движение может быть дополнено до понятий движения по прямой, движения по окружности, движения с дискретным временем, в которых области с одинаковыми именами могут иметь разные множества элементов.

Для представления понятий категорными средствами нужны категории с операциями, которые позволяют по одним областям и морфизмам строить другие области и морфизмы. Некоторые операции такого типа были уже введены — это $\_()$, $\_()$, которые каждому морфизму сопоставляют соответствующие области, операция $id()$, которая каждой области сопоставляет тождественное отображение. Если $\_(f_1) = \_(f_2)$, то определена операция (f_1, f_2), которая сопоставляет паре морфизмов f_1, f_2 морфизм $f_2 * f_1$. Другие примеры операций — это выделенные области $I, \emptyset$, операции $I(), \emptyset()$, которые области T сопоставляют морфизмы $I(T), \emptyset(T)$.

Для представления сложных понятий перечисленных выше операций недостаточно. Нужны также операции типа декартова произведения областей, суммы областей, пересечения областей, фактор областей и т.д. При этом следует учесть, что область, как показано было выше, не определяется своими элементами, поэтому теоретико множественные определения этих операций не подходят.

Математики давно искали конструкции, соответствующие теоретико множественным операциям, но не использующие элементы областей. Такая работа была проделана в теории категорий. Точные определения таких конструкций можно найти в [16].

Для примера рассмотрим операцию декартова произведения. Декартово произведение $T_1 \times T_2$ двух множеств T_1, T_2 определяется в теории множеств формулой: $T_1 \times T_2 = \{(t_1, t_2) : t_1 \in T_1, t_2 \in T_2\}$. А как опреде-

лить произведение областей T_1, T_2, если не все элементы областей известны? Имеется следующий вариант определения декартова произведения в теории категорий.

Пусть T_1, T_2 — объекты категории $\mathcal{C}$. Декартово произведение $T_1 \times T_2$ — это объект вместе с морфизмами (проекциями) $pr_1 : T_1 \times T_2 \to T_1$ и $pr_2 : T_1 \times T_2 \to T_2$, обладающие следующими свойствами: для любой пары морфизмов $f_1 : T \to T_1$, $f_2 : T \to T_2$ есть морфизм $(f_1, f_2) : T \to T_1 \times T_2$, композиция которого с проекциями дает равенства $pr_1 * (f_1, f_2) = f_1$ и $pr_2 * (f_1, f_2) = f_2$, и, наоборот, любой морфизм вида $f : T \to T_1 \times T_2$ представляется в виде $f = (pr_1 * f, pr_2 * f)$.

В этом определении декартово произведение объектов задается набором операций: произведение объектов $T_1 \times T_2$, проекции $pr_1(T_1, T_2)$, $pr_2(T_1, T_2)$, произведение морфизмов (f_1, f_2), которые по областям T_1, T_2 и морфизмам f_1, f_2 строят соответствующие объект и морфизмы. Кроме того, задаются определяющие соотношения между этими операциями.

Сходным способом могут быть определены аналоги многих других теоретико множественных конструкций. В теории категорий (см., например [16]) определен некоторый набор категорных операций, моделирующих полный набор обычных теоретико множественных операций. Математический объект с этой системой операций называется в теории категорий топосом.

В категорном подходе к представлению понятий, в соответствии со стандартной категорной идеологией, области представляются объектами, отображения — морфизмами, отношения — подобъектами, а предикаты — морфизмами в выделенный объект Ω, который называется классификатором подобектов. Объект Ω имеет два выделенных элемента $true : I \to \Omega$ и $false : I \to \Omega$. Свое название объект Ω получил в связи с требованием, чтобы для любого подобъекта $A \subset B$ существовал единственный "характеристический" морфизм $\chi(A) : B \to \Omega$, область истинности которого, то есть предел диаграммы

$$\begin{array}{ccc} & & B \\ & & \downarrow \chi(A) \\ I & \stackrel{true}{\to} & \Omega \end{array}$$

совпадает с подобъектом A.

В общем случае, объект Ω может иметь другие элементы $I \to \Omega$,

отличные от $true$ и $false$, то есть другие значения истинности, связанные с неполнотой представленной информации в данной категории. В связи с этим объект Ω будет также называться областью истин.

2.3 Конечные задания категорий и конечные аппроксимации категорий

В алгебраическом топосе для каждой аксиомы элементарного топоса вместо требования существования объекта или морфизма с требуемыми свойствами полагается, что этот объект или морфизм строится соответствующей операцией. Алгебраические топосы сопоставляемые представляемому понятию задаются конечным множеством образующих и определяющих соотношений. Существование таких топосов следует из следующего утверждения.

У т в е р ж д е н и е 2.1. Аксиоматика алгебраического топоса может быть задана хорновскими формулами вида $P_1 \wedge ... \wedge P_n \rightarrow P$, где P_i — элементарные формулы, выражающие либо равенство объектов, либо равенство морфизмов, а P — также либо равенство, либо условие применимости одной из операций алгебраического топоса.

Доказательство утверждения носит технический характер и здесь не приводится. Оно состоит в вводе операций над объектами и морфизмами, требуемых в определении топоса и проверке того, что все условия накладываемые в аксиоматике топоса на соотношения между этими операциями, и условия их применимости имеют хорновский вид.

Как известно (см., например, [54]), среди моделей теории с аксиомами, заданными в виде хорновских формул, всегда найдется наименьшая (инициальная) модель, которая может быть индуктивно построена как фактор множество правильно построенных термов из имен операций по индуктивно построенному отношению эквивалентности на термах. Эта конструкция позволяет задавать алгебраические топосы, сопоставляемые представляемым понятиям, конечным множеством образующих — именами областей, отображений и отношений — и конечным набором соотношений в виде хорновских формул относительно равенства.

Однако требование машинной реализации этих категорий предъявляет более сильные требования к конечности представления, точно такие же, как к машинной реализации целых чисел: возможно конечное определение мно-

жества целых чисел, но в машине целые числа представляются в виде машинных целых чисел, которые образуют конечное множество. Аналогично возникает задача о конечной аппроксимации категорий, т.е. представления категорий в виде конечных множеств с некоторой точностью.

Примерами конечных аппроксимаций конечно заданных категорий являются конечные множества термов этой категории и конечные подотношения эквивалентности, полученные на каком-либо конечном шаге индуктивного построения категории. Аппроксимация считается более точной, если она получена на более позднем шаге индуктивного построения категории.

Заметим, что категория, сопоставляемая представляемому понятию, является идеальным (как правило, бесконечным и с алгоритмически неразрешимой проблемой равенства) математическим объектом, отражающим полное знание о понятии, а конечная аппроксимация этой категории отражает знания полученные о представляемом понятии и доступные пользователю. Предполагается, что "хорошие" аппроксимации содержат все, что обычно используется при обращении к аппроксимируемому понятию. С другой стороны, предполагается, что в "хороших" системах имеются возможности расширения аппроксимации, если в этом возникнет необходимость, по ходу использования понятия автоматически или самим пользователем, исходя из знаний прикладной области. В аппроксимациях должны быть отражены уже известные полезные знания, которые могут быть введены в систему в виде теорем и их доказательств, либо представлены в виде правил переписывания термов.

Особая роль имеет использование аппроксимации для распознавания эквивалентности термов. Так как категория $\mathcal{C}$ является фактор множеством термов по некоторому отношению эквивалентности, то возникает задача о каноническом представлении областей и морфизмов категории $\mathcal{C}$ среди эквивалентных термов.

Поясним эту задачу на примере. Если в категории $\mathcal{C}$ представляется тип данных целые числа, то в множестве правильно построенных выражений имеются следующие эквивалентные выражения: 2*10+4; (8+4)*2; 3*8. Однако, когда мы спрашиваем, чему равно $(8 + 4) * 2$, то в ответ хотим получить один выделенный элемент в классе эквивалентных выражений, содержащих выражение $(8 + 4) * 2$. Вычислить арифметическое выражение $(8 + 4) * 2$, это значит выдать более простое (в каком-то смысле) или каноническое выражение, равное данному. Если каноническое представление арифметических выражений есть их представление в десятичном исчисле-

нии, то в результате мы ожидаем получить $2 * 10 + 4$. Если система исчисления другая, например восьмеричная то результат будет $3 * 8$. Таким образом, понятие вычисления термов тесно связано с понятием канонической системы термов.

Выделенный элемент в классе эквивалентных (синонимичных) термов будем называть дескриптором или каноническим термом этого класса. В теории вычисления в абстрактных типах данных его принято называть каноническим [46, 27].

О п р е д е л е н и е 2.1. Множество термов $C \subset Term_{\Sigma}(\emptyset)$ называется канонической системой термов абстрактного типа данных $\mathcal{A} =< Name, \Sigma, E >$, если выполнены следующие условия:

для всякого $t \in Term_{\Sigma}(\emptyset)$ существует $c \in C$ такой, что $t \cong_E c$;

для любых $c_1, c_2 \in C$ из соотношения $c_1 \cong_E c_2$ следует, что термы c_1 и c_2 совпадают.

Т е о р е м а 2.2. Если C — каноническая система термов, то отображение, сопоставляющее элементу $c \in C$ его класс эквивалентности $[c] \in I(\mathcal{A})$, является взаимно однозначным отображением.

Доказательство теоремы очевидно.

О п р е д е л е н и е 2.2. Вычислением значений термов в I(A) при выбранной канонической системе термов C называется функция $calc : Term_{\Sigma}(\emptyset) \to C$, сопоставляющая $t \in Term_{\Sigma}(\emptyset)$ элемент $calc(t) \in C$ такой, что $t \cong_E calc(t)$.

Так как вычисление термов должно проводиться на машине, то хотелось бы иметь алгоритм определения значений функции $calc$. Однако доказано, что в общем случае такого алгоритма не существует. Имеются примеры алгебр, заданных конечной сигнатурой и конечным множеством определяющих соотношений, для которых проблема равенства двух термов алгоритмически неразрешима. Поэтому можно надеяться в получении единой техники вычислений лишь для случаев, когда это возможно, либо можно получить лишь частичное вычисление, когда система термов уточняется в процессе пополнения знаний об известных соотношениях в множестве термов.

Поэтому выделение дескрипторов в аппроксимации категории производится либо пользователем системы как при определении понятия, так и в диалоге с системой во время получения ответов на запросы, либо автоматически в процессе построения конечной аппроксимации. В последнем случае выделение дескрипторов может происходить, например, по следу-

ющему правилу: терм является дескриптором в данной аппроксимации категории, если все остальные эквивалентные ему в аппроксимации термы получены на более поздних шагах индуктивного построения категории.

Среди способов вычисления термов выделяют два способа: вычисление снизу по шагам в той последовательности, как это указано в терме, и переписывание термов — упрощение их до канонического вида.

В первом способе предполагается, что функция $calc()$ уже определена на термах вида $F()$, где F— имя нульарной операции, и термах вида $F(c_1, ..., c_n)$, где F— имя n-арной операции и $c_1, ..., c_n \in C$. Для конечного фрагмента алгебры или категории эта функция может быть представлена в некоторой базе данных. Тогда на произвольном терме вида $F(t_1, ..., t_n)$, где $t_1, ..., t_n \in Term_\Sigma(\emptyset)$, функция $calc$ определяется следующим индуктивным правилом:

$$calc(F(t_1, ..., t_n)) = F(calc(t_1), ..., calc(t_n)).$$

Именно этим способом обычно вычисляются значения арифметических выражений.

Объемы данных, необходимые для хранения аппроксимации категории в таком виде, могут быть очень велики, поэтому возникает проблема хранения аппроксимации в памяти ЭВМ. Представляется, что это хранение должно быть организовано средствами баз данных.

Если такой системе предъявляется терм категории $\mathcal{C}$, то, последовательно применяя операции категории в порядке, указанном термом, система, используя таблицы результатов операций, найдет дескриптор, соответствующий терму, если он входит в данную аппроксимацию категории. Для проверки равенства двух термов система вычисляет дескрипторы этих термов в данной аппроксимации. Если дескрипторы совпадают, то термы эквивалентны. В противном случае в данной аппроксимации они не эквивалентны.

Другой способ вычисления термов основывается на понятиях подстановки термов в терм, подтерма и системы правил переписывания термов. В [25] дан обзор теории систем переписывания термов. Эта теория была успешно применена в системе AFFIRM [27] для вычислений в инициальных моделях абстрактных типов данных.

В конкретных реализациях систем использующих данный подход, возможны различия в выборе того, какая часть аппроксимации действительно хранится в том или ином виде, а какая часть вычисляется с помощью пра-

вил переписывания. Это связано с эффективностью системы и с конструкторскими соображениями и знаниями в предметной области о представляемом понятии. В любом случае, на запросы пользователей система должна выдавать ответы в виде выражений из дескрипторов.

Теперь несколько слов о запросах к аппроксимации категории и построении ответов. Рассмотрим для примера следующие запросы: "Чему равно A?","Равны A и B?", "Какие элементы области D известны системе?". В этих запросах A, B — термы, задающие область или морфизм категории $\mathcal{C}$, а D — терм, задающий область в $\mathcal{C}$.

Ответом на первый вопрос является дескриптор терма A в данной аппроксимации. Если A не входит в аппроксимацию, то система найдет первое неизвестное ей применение операции в терме A и выдаст в ответ сообщение, что значение терма A неизвестно, так как неизвестно значение такой-то операции на таких-то дескрипторах.

Второй вопрос распадается для системы на следующие вопросы: "Чему равно A?", "Чему равно ?", "Совпадают дескриптор A и дескриптор B".

В ответ на третий вопрос, если дескриптор D является дескриптором области, выдается множество всех дескрипторов подобластей области D, изоморфных ТОЧКЕ.

Учитывая выразительные возможности операций рассматриваемых категорий, уже эти типы запросов позволяют построить большое разнообразие запросов к системе. Запрос типа "Чему равно A?" позволяет, например, проводить вычисления и преобразование термов. Частным случаем запроса "Равны A и B?" является запрос об истинности замкнутой формулы в категории $\mathcal{C}$ (при моделировании логик в топосах, замкнутой формулой по определению называется терм, задающий морфизм из области ТОЧКА в область истин Ω [16]). В этом случае в запросе нужно положить $A =$ и $B =$ — выделенный элемент в области Ω.
Таким образом, в язык запросов входит любая замкнутая формула языка категорий, которая может быть получена из образующих и определяющих соотношений категории.

К запросу типа: "Какие элементы области D известны системе?", относятся запросы: "Выдать всех поставщиков завода X", где D — область "поставщики завода X", или запросы: "Выдать все химические элементы, удовлетворяющие условию P", где D — это область "химические элементы, удовлетворяющие условию P".

Все перечисленные запросы являются частными случаями запросов к

реляционной базе данных, "хранящей" аппроксимацию категории $\mathcal{C}$ в виде набора отношений (для каждой категорной операции свое отношение) между дескрипторами. В общем случае возможен произвольный запрос языка запросов реляционной модели к этой базе данных. Ответами на такие запросы являются отношения из дескрипторов данной аппроксимации, получающиеся в результате применения операций запроса к базисным отношениям аппроксимации. Отношения - ответы трактуются как известные в данной аппроксимации наборы дескрипторов, удовлетворяющие условиям запроса. Пустые отношения трактуются как "неизвестно".

3 Операции над типами данных и параметрические типы данных

В данном разделе предлагается некоторое общее определение понятия операции над типами данных в рамках алгебраического подхода к типам данных в программировании [12], [52], [53].

Пусть имеются две модели A и A' абстрактного типа данных $\mathcal{A}$ (см. определения 1.4, 1.5 этой главы). Напомним, что взаимно однозначное отображение $h : A \to A'$ несущих множеств моделей называется изоморфизмом, если оно переводит функции и отношения одной модели в соответствующие функции и отношения другой модели.

О п р е д е л е н и е. 3.1. Операцией α над типами данных из набора абстрактных типов данных $\mathcal{A}_1, ..., \mathcal{A}_n$ в абстрактный тип данных $\mathcal{B}$ называется сопоставление наборам моделей $A_1, ..., A_n$, соответствующих абстрактным типам данных $\mathcal{A}_1, ..., \mathcal{A}_n$, некоторой модели $\alpha(A_1, ..., A_n)$ абстрактного типа данных $\mathcal{B}$. При этом требуется, чтобы такое сопоставление удовлетворяло следующему условию естественности: по набору $h = (h_1, ..., h_n)$ изоморфизмов $h_1 : A_1 \to A'_1, ..., h_n : A_n \to A'_n$ между моделями абстрактных типов данных операция строит изоморфизм $\alpha(h_1, ..., h_n)$ между результатами — моделями $\alpha(A_1, ..., A_n)$ и $\alpha(A'_1, ..., A'_n)$ абстрактного типа данных $\mathcal{B}$. Требуется, кроме того , чтобы это сопоставление $h \mapsto \alpha(h)$ было функториальным, т.е. тождественному изоморфизму сопоставлялся тождественный изоморфизм, а композиции изоморфизмов - композиция изоморфизмов.

Таким образом, операция α по реализациям $A_1, ..., A_n$ абстрактных типов данных $\mathcal{A}_1, ..., \mathcal{A}_n$ строит реализацию B абстрактного типа данных $\mathcal{B}$, а ее естественность означает, что по переименованию элементов исходных типов данных она должна построить переименование элементов в результате операции.

Нетрудно видеть, что обычные конструкции структурных типов такие, как $ARRAY$, $RECORD$,$SETOF$, $LIST\_OF\_T$ являются операциями в смысле приведенного выше определения.

Наиболее важными примерами операций над типами данных из абстрактного типа $\mathcal{A}$ в абстрактный тип $\mathcal{B}$ являются операции, индуцированные корректными логическими интерпретациями [32] абстрактного типа $\mathcal{B}$ в тип $\mathcal{A}$, рассмотренными как теории в исчислении предикатов. Рассмотрим здесь более подробно частный случай логической интерпретации —

морфизм абстрактных типов [46].

О п р е д е л е н и е. 3.2. Морфизмом $h : \Sigma \to \Sigma'$ сигнатуры $\Sigma = (S, OP)$ в сигнатуру $\Sigma' = (S', OP')$ называется пара функций

$$h = (h_S : S \to S', h_{OP} : OP \to OP')$$

таких, что для каждого функционального символа $F \in OP$ типа $F : T_1 \times ... \times T_n \to T_k$ выполняется соотношение

$$h_{OP}(F) : h_S(T_1) \times ... \times h_S(T_n) \to h_S(T_k).$$

Если задан морфизм сигнатур $h : \Sigma \to \Sigma'$, то он определяет преобразование термов в сигнатуре Σ в термы в сигнатуре Σ' по следующему правилу: переменные X типа T в терме t преобразовываются в переменные $h(X)$ типа $h_S(T)$, а функциональные символы F в терме t преобразовываются в функциональные символы $h_{OP}(F)$. Нетрудно видеть, что в результате такого преобразования получается терм. Этот терм будет обозначаться выражением $h(t)$.

О п р е д е л е н и е. 3.3. Для двух абстрактных типов данных $\mathcal{B} = (S, OP, E)$ и $\mathcal{A} = (S', OP', E')$ морфизм сигнатур $h : (S, OP) \to (S', OP')$ называется морфизмом абстрактных типов данных $h : \mathcal{B} \to \mathcal{A}$, если для каждого уравнения $(r = l) \in E$ уравнение $h(r) = h(l)$ выводится из определяющих соотношений E' абстрактного типа данных $\mathcal{A}$.

Пусть $h : \mathcal{B} \to \mathcal{A}$ — морфизм абстрактных типов данных. Операцией обратного образа h^* над данными $\mathcal{A}$ (в литературе эту операцию называют также забывающим функтором) из абстрактного типа данных $\mathcal{A}$ в абстрактный тип данных $\mathcal{B}$, индуцированной морфизмом h, называется операция, которая модели A абстрактного типа данных $\mathcal{A}$ сопоставляет модель $B = h^*(A)$ абстрактного типа данных $\mathcal{B}$ по следующему правилу:

– B_T — множество элементов сорта $T \in OP$ алгебры B совпадает с множеством элементов $A_{h(T)}$ алгебры A;

– для любого имени операции $F \in OP$ соответствующая ей операция алгебры B совпадает, как функция, с операцией $h_{OP}(F)$ алгебры A.

Естественность операции h^*, то есть функториальность относительно изоморфизмов моделей $\mathcal{A}$, очевидна.

Для морфизма абстрактных типов данных $h : \mathcal{B} \to \mathcal{A}$ можно определить другую операцию над типами, которая обозначается h_* и называется операцией прямого образа типа данных $\mathcal{B}$, индуцированной морфизмом h.

Она сопоставляет каждой модели B абстрактного типа данных $\mathcal{B}$ модель $A = h_*(B)$ абстрактного типа данных $\mathcal{A}$, которая строится как свободная $\mathcal{A}$ -алгебра, порожденная множествами элементов B_T, рассматриваемых как элементы типа $h_S(T)$, для всех $T \in OP$ с дополнительными соотношениями $h_{OP}(F)(b_1, ..., b_n) = F(b_1, ..., b_n)$ для любых $F \in OP$ и $b_1, ..., b_n \in B$, для которых эти соотношения имеют смысл. Естественность (функториальность) операции h_* также очевидна.

В литературе [46] особое внимание уделяется морфизмам $h : \mathcal{B} \to \mathcal{A}$, удовлетворяющим соотношению $h^*(h_*(B)) = B$ для любой модели B абстрактного типа данных $\mathcal{B}$. Смысл этого соотношения в том, что операция h_* свободного расширения алгебры B по морфизму h не портит исходной алгебры. Такие морфизмы называются сохраняющими (persistent).

При задании операций, индуцированных морфизмами абстрактных типов данных, желательна программная поддержка проверок, что данный морфизм сигнатур является морфизмом абстрактных типов данных или что данный морфизм является сохраняющим. Такую поддержку могут осуществить автоматизированные системы доказательства, основанные на алгоритме Кнута - Бендикса [61], или на методе резолюций, или на их сочетании.

В общем случае операции, индуцированные морфизмами абстрактных типов данных, позволяют расширять множество сигнатурных символов и множество определяющих соотношений абстрактного типа данных за счет определений и следствий из определяющих соотношений. Этими операциями можно выполнить любое сужение множества сигнатурных символов и множества определяющих соотношений, а также переименование сигнатурных символов и т.д.

Операции, индуцированные логическими интерпретациями (морфизмами), дают примеры унарных (одноместных) операций над типами данных. Пример нульарной операции I для алгебраических абстрактных типов данных $\mathcal{A}$ дает операция $I(\mathcal{A})$, выделяющая инициальную модель $\mathcal{A}$.

Все операции над типами данных можно свести к нульарным и унарным с помощью следующей операции объединения типов. Пусть $\mathcal{A}_1, ..., \mathcal{A}_n$ - произвольные абстрактные типы данных, а $\mathcal{B}$ - абстрактный тип данных, множества сигнатурных символов и соотношений которого есть разъединенные объединения соответствующих множеств исходных типов $\mathcal{A}_1$,...,$\mathcal{A}_n$. Операция объединения типов $\mathcal{A}_1$,...,$\mathcal{A}_n$ набору их моделей $A_1, ..., A_n$ сопоставляет этот же набор $A_1, ..., A_n$, рассматриваемый как модель абстракт-

ного типа $\mathcal{B}$.

Используя понятие операции, индуцированной морфизмом абстрактных типов данных, можно дать следующее определение спецификации абстрактного параметрического типа данных [46].

О п р е д е л е н и е. 3.4. Синтаксически параметрический абстрактный тип данных задается четырьмя абстрактными типами данных и четырьмя морфизмами:

PAR— абстрактным типом данных, описывающим параметры;

IMP — абстрактным типом данных, описывающим внешние типы данных, которые используются в данной спецификации;

EXP — абстрактным типом данных, в котором приведены сигнатурные символы и соотношения, доступные для внешних пользователей данной спецификации;

$BODI$ — абстрактный тип данных, описывающий построение параметрического типа данных;

$i : PAR \to IMP$ — морфизм - вложение параметров во внешние типы данных;

$j : PAR \to EXT$ — морфизм, связывающий типы данных параметров с типами данных, выдаваемых спецификацией;

$s : IMP \to BODI$ — морфизм, устанавливающий соответствие между внешними импортируемыми типами данных и типами данных в теле параметрического типа данных;

$h : EXP \to BODI$ — морфизм, выделяющий выдаваемые спецификацией типы данных в теле параметрического типа.

При этом требуется, чтобы следующая диаграмма была коммутативна:

$$\begin{array}{ccc} PAR & \stackrel{j}{\longrightarrow} & EXP \\ i\downarrow & & \downarrow h \\ IMP & \stackrel{s}{\longrightarrow} & BODI \end{array}$$

То есть выполняется соотношение $h \circ j = s \circ i$.

Спецификация параметрического абстрактного типа данных называется семантически корректной, если $s : IMP \to BODI$ является сохраняющим (persistent) морфизмом.

По спецификации параметрического абстрактного типа данных строится операция из абстрактного типа данных IMP в абстрактный тип данных EXP по следующему правилу: реализации A абстрактного типа данных

IMP сопоставляется реализация $B = h^*(s_*(A))$ абстрактного типа данных EXP, где h^* — забывающий функтор по морфизму h, а s_* — функтор, строящий свободную алгебру по мофизму s и алгебре A.

Другими примерами операций над типами данных являются хорошо известные операции декартова произведения, разъединенного объединения, построения множества всех преобразований из одного типа в другой, массивов элементов какого-либо типа, конечных множеств элементов некоторого типа и т.д., то есть операции, часто используемые в программировании. Многие из них могут быть описаны в виде конструкции, описанной выше.

Так, например, операция примера 1.4 — тип данных LIST_OF_T, приведенный в подразделе 1.2, описывается операцией индуцированной морфизмом абстрактных типов данных. В нем абстрактный тип данных $\mathcal{B} = (T, \emptyset, \emptyset)$ без функциональных символов и соотношений, а абстрактный тип данных $\mathcal{A}$ задан спецификацией LIST_OF_T примера 1.4.

Нетрудно видеть, что композиции операций над типами данных также являются операциями над типами данных.Таким образом, мы имеем клон операций. Исследование этого клона как алгебраической системы — одна из важных задач развития механизма типов в программировании.

Литература

[1] Агафонов В. Н. *Спецификация программ: понятийные средства и их организация.* Новосибирск: Наука,Сиб.отд-ние,1990.

[2] Агафонов В. Н. *Типы и абстракция данных в языках программирования.* В кн. Данные в языках программирования. М.:Мир,1982, с.263–327.

[3] Бениаминов Е. М. *Алгебраический подход к моделям баз данных реляционного типа.* В кн.:Семиотика и информатика, 1980, вып.14, с.44-80.

[4] Бениаминов Е. М. *Алгебраическая стуктура реляционных моделей баз данных.* НТИ, сер.2, 1980, N9, с. 23-25.

[5] Бениаминов Е. М., Березина Н. А. *Об алгебраическом подходе к описанию схем баз данных.//* В сб. Вопросы создания Автоматизированной системы НТИ по документам ГАФ СССР, Москва: ГАУ при Совете Министров СССР, ВНИИДАД, 1981, с.69-77.

[6] Бениаминов Е. М. *О роли симметрии в реляционных моделях баз данных и логических структурах.* НТИ, сер.2, 1984, N5, с.17-25.

[7] Бениаминов Е. М., Березина Н. А., Дунская М. В. *Разработка методов моделирования автоматизированной обработки, поиска и размещения данных в больших информационных системах.* СИФ ОЦ-НТИ, ВНИИДАД, депонированная рукопись, инв. N031–85,М.,1985.

[8] Бениаминов Е. М. *О некотором подходе к представлению знаний.* Тезисы докладов 4 Всесоюзн.конференции "Применение методов мат. логики" г. Таллин, 1986, с.34-36.

[9] Бениаминов Е. М. *Основания категорного подхода к представлению знаний. Категорные средства.* Изв. АН СССР Техн. кибернет.,N 2, 1988 , с.21–33.

[10] Бениаминов Е. М. *Рефлексивные топосы в категорном подходе к представлению знаний.* Тезисы докладов Всесоюзн. школы-семинара

"Семиотические аспекты формализации интеллектуальной деятельности" г. Боржоми, 1988, с.111-113.

[11] Бениаминов Е.М., Вайнтроб А. Ю. *Основные принципы диалогового языка для представления знаний средствами категорного подхода.* Материалы конференции ДИАЛОГ-87, г. Тбилиси, 1988, с.174-177.

[12] Бениаминов Е. М. *Алгебраические системы и типы данных.* //В кн.:Системное и теоретическое программирование, Ростов-н-Д: РГУ, 1988, с.83-92.

[13] Бениаминов Е. М. *Система представления и обработки понятий, основанная на алгебраическом (категорном) подходе.* Труды II Всесоюзной конференции "Искусственный интеллект- 90", т.2, 1990, с.8-11.

[14] Вигнер П. *Программирование на языке АДА.* М.:Мир, 1983.

[15] Глушков В. М., Цейтлин Г. Е., Ющенко Е. Л. *Алгебра, логика, языки, программирование.* Киев:Наукова думка, 1974.

[16] Голдблатт Р. *Топосы. Категорный анализ логики.* М.:Мир, 1983.

[17] Гончаров С. С., Ершов Ю. Л., Свириденко Д. И. *Семантические основы логического программирования*// В сб. Концептуализация и смысл, под ред. Полякова И. В., Новосибирск:Наука, 1990, с.6-20.

[18] Джонстон П. Т. *Теория топосов.* М.:Наука, 1986.

[19] Жожикашвили А.В., Стефанюк В.Л. *Теория категорий в задачах представления знаний и обучения.* Изв. АН СССР. Техн. кибернет., N 2, 1986.

[20] Замулин А. В. *Системы программирования баз данных и знаний.* Новосибирск:Наука, 1990.

[21] Калиниченко Л. А. *Методы и средства интеграции неоднородных баз данных.* М.:Наука, 1983.

[22] Калиниченко Л. А., Рывкин В. М. *Машины баз данных и знаний.* М.:Наука, 1990.

[23] Капитонова Ю. В., Летичевский А. А. *Математическая теория проектирования вычислительных систем.* М.:Наука, 1988.

[24] Кондрашина Е. Ю., Литвинцева Л. В., Поспелов Д. А. *Представление знаний о времени и пространстве в интеллектуальных системах.* М.:Наука, 1989.

[25] Кучеров Г. А. *Системы подстановок термов.* Препринт 601, ВЦ АН СССР Сиб. отделение, Новосибирск, 1985.

[26] Мальцев А. И. *К общей теории алгебраических систем.* Мат. сборник, 1954, т.35, вып.1.

[27] Массер Д. *Спецификация абстрактных типов данных в системе AFFIRM.//* В сб.: Требования и спецификации в разработке программ, М.:Мир, 1984, с.199-222.

[28] Плоткин Б. И. *Универсальная алгебра, алгебраическая логика и базы данных.* М.:Наука, 1991.

[29] Поспелов Д. А. *Логико-лингвистические модели в системах управления.* М.:Энергоиздат, 1981.

[30] Свириденко Д. И. *Проект Сигма. Цели и задачи. //* В сб. Логические методы в программировании под ред. Ершова Ю. Л. (Вычислительные системы, вып. 133), РАН, Сиб. отд.-ние, Ин.-т математики, Новосибирск, 1990, с.68-94.

[31] Цаленко М. Ш. *Моделирование семантики в базах данных.* М.:Наука, 1989.

[32] Шенфилд Дж. *Математическая логика.* М.:Наука, 1975.

[33] Andréka H., Németi I. *Applications of universal algebra, model theory and categories in computer science (Part 1)* Comput. Lingust. Comput. Lang., 13, 1979, p. 152-282.

[34] Andréka H., Németi I. *Applications of universal algebra, model theory and categories in computer science (Part 2)* Comput. Lingust. Comput. Lang., 14, 1980, p. 43-65.

[35] Andréka H., Németi I. *Applications of universal algebra, model theory and categories in computer science (Part 3)* Lect. Notes in Comp. Sci., V.117, Springer-Verlag, Berlin, 1981, p. 281-290.

[36] Banchilhon F. *On the Compliteness of Query Language for Relational Data Bases.* Lect. Notes in Comp. Sci., V.64, Springer-Verlag, 1978, pp.76-98.

[37] Beniaminov E. M. *Concept Bases and Algebraic Modeling Methods.* Proceedings of the International Workshop on Advances in Databases and Information Systems (ADBIS'94), 1994, p.133-135.

[38] Beniaminov E. M. *A Categorical Approach to Knowledge Representation.* Japan-CIS Symposium on Knowledge Based Software Engineering'94 (JCKBSE'94), 1994, p.181-182.

[39] Beniaminov E. M. *Algebraic Invariants of Database Schemes.* Proceedings of the Second International Workshop on Advances in Databases and Information Systems (ADBIS'95), V.1, 1995, p.259-263.

[40] Breu R. *Algebraic Specification Techniques in OOP Environments.* Lect. Notes in Comp. Sci., V.562, Springer-Verlag, 1991.

[41] Burstall R. M., Goguen J. A. *The Semantics of Clear, a Specification Language* In: Proceedings, of the 1979 Copenhagen Winter Shcool on Abstrakt Software Specification, Springer-Verlad, Berlin, 1980, 292-332.

[42] Chandra A. K., Harel D. *Computable queries for relational data bases.* J. Comput. & Syst. Sci., V.21, N2, 1980, p.156-172.

[43] Codd E. F. *A relational model for large shared data banks.* Comm. of ACM 13, 6, 1970, p.377-387.

[44] Dershowitz N. *Orderings for term-rewriting systems.* Theor. Comput. Sci., 1982, V.17, 3, pp.279-302.

[45] Elmasri R., Weeldrever J., Hevner A. *The category concept: an extension to the entity-relatconship model* Data & Knowledge Engineering, V.1, N1,1985.

[46] Ehrig H., Mahr B. *Fundamentals of Algebraic Specification.* V.1, Springer-Verlag, 1985.

[47] Ehrig H., Mahr B. *Fundamentals of Algebraic Specification.* V.2, Springer-Verlag, 1990.

[48] *First International Symposium on Category Theory Applied to Computation and Control* Lect. Notes in Com. Sci, V.25, Springer - Verlad, Berlin, 1975.

[49] Georgescu I. *A Categorial approach to knowledge-based systems.* Computers and Artifical Intelligence, V.3, N2, 1984, pp.105-113.

[50] Goguen J. A., Thatchar J. W., Wagner E. G. *An inititial algebra approach to the specification, correctness and implementation of abstract data types.* In Current Trends in Programming Methodology IV: Data Structuring, Prentice Hall, 1978, pp.80-144.

[51] Goguen J. A. *Some design principles and theory for OBJ-O, a language to express and execute algebraic specifications of problems.* In: Lect. Notes Comput. Sci.,V.75, 1979, pp.425-473.

[52] Goguen J. A., Burstall R. M. *Some fundamental algebraic tools for the semantics of computation.* Theoretical Computer Science, V.31, N2, 3, 1984.

[53] Goguen J. A., Burstall R. M. *Introducing Institution.* Lect. Not. Comp.Sci., V.164, 1984.

[54] Goguen J. A., Meseguer J. *Equality, types, modules, and (why not?) generics for logic programming.* Conf. of Logical Prog., Uppsala, Sweden, 1984, 179-210.

[55] Goguen J. A., Meseguer J. *Eqlog: Equality, types, and generic modules for logic programming.* In Douglas SeGroot and Gary Lindstrom, eds. "Logic Programming: Functions, Relations and Equations", Prentice-Hall, Englewood Cliffs, N.J., 1986, pp.295-363.

[56] Goguen J. A., Meseguer J. *Order-sorted algebra solves the constructor selector, multiple representation and coercion problems.* Symposium on Logic in Comp. Sci., IEEE Comp. Society Press, 1987, pp.18-29.

[57] Goguen J. A., Burstall R. M. *Institutions: Abstract Model Theory for Specification and Programming.* Jorn. of ACM, V.39, 1, 1992, pp.95-146.

[58] Grothendieck A., Verdier J. L. *Théorie des Topos.* (SGA 4, exposés I-VI).–Second edition.–Berlin; Heidelberg; N. Y.:Springer, 1972.

[59] Huet G. *Confluent reductions: abstract properties and applications to term rewriting systems.* Journ. of ACM, 1980, v.27, N4, pp.797-821.

[60] Kaplan S. *Symplifying conditional term rewriting systems: unification, termination and confluence.* Jorn. Symbolic Computation 4(3), 1987, pp.295-334.

[61] Knuth D., Bendix P. *Simple word problems in universal algebras.* In: Conputational Problems in Abstract Algebra, Pergamon Press, Elmsford, New York, 1970, pp.263-297.

[62] Krasner M. I. *Generalization et analogues de la theorie de Galois.* Comptes Rendus de Congress de la Victorie de I'Ass. Franc. pour l'Avancem. Sci., 1945, pp. 54-58.

[63] Lawvere F. W. *Functorial semantics of algebraic theories.* Proc. Nat. Acad. Sci., 1963, V.50, N5, pp.869-872.

[64] Lawvere F. W. *Introduction* In: Toposes, Algebraic Geometry and Logic, Lect. Notes in Math., V.174, 1972.

[65] Maltsev A. *Algebraic Systems.* Springer-Verlag, 1973.

[66] Melton A., Schmidt D., and Strecher G. *Galois connections and computer science applications.* Lect. Notes in Comp. Sci., V. 240, Springer-Verlag, 1986.

[67] *Proceedings, Category Theory and Computer Programming,* Lect. Notes in Comp. Sci., V. 240, Springer-Verlag, 1986.

[68] *Proceedings, Category Theory and Computer Science,* Lect. Notes in Comp. Sci. V. 283, Springer-Verlag, 1987.

[69] Rydeheard D.F., Burstall R.M. *Computational category theory.* Prentice Hall, 1988.

[70] Tuijn C., Gyssens M., Paredaens J. *A Categorical Approach to Object-Oriented Data Modelling.* Proceedings of Third Workshop on Foundation of Models and Languages for Data and Objects, Aigen, 1991, pp.187–196.

[71] Rusinowitch M., Rémy J. L. (Eds.) *Conditional Term Rewriting Systems.* Lect. Notes in Comp. Sci., V. 656, Springer-Verlag, 1993.

[72] Zilles S.N. *Introduction to data algebras.* Lect. Notes Comput. Sci., V.86, Springer-Verlag, 1980.