ТАРТУСКИЙ
ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

ТРУДЫ

ВЫЧИСЛИТЕЛЬНОГО ЦЕНТРА

43

ТАРТУ
1980

ТАРТУСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

БАНК ДАННЫХ ДЛЯ ЕС ЭВМ

ТРУДЫ
ВЫЧИСЛИТЕЛЬНОГО
ЦЕНТРА

ВЫПУСК 43

ТАРТУ 1980

БАНК ДАННЫХ ДЛЯ ЕС ЭВМ. Труды вычислительного центра. Выпуск 43. На русском языке. Тартуский государственный университет. ЭССР, г. Тарту, ул.Юликооли, 18. Ответственный редактор Ю. Таппер. Корректор А. Райд. Сдано в печать 12.11.1979. Бумага писчая 30х42 1/4. Печ. листов 8,25(условных 7,67). Учетно-издат. листов 6,29. Тираж 500. МВ 07574. Типография ТГУ, ЭССР, г. Тарту, ул. Пялсона, 14. Зак. № 1549. Цена 95 коп.

ДЕРЕВО ОПИСАНИЯ ЗАПИСИ В СИСТЕМЕ РАМА

А. Изотамм

В статье [1] представлен язык RDL для определения записи в системе РАМА. Каждое описание записи на этом языке называется легендой. В настоящей статье представляется результат транслирования легенды - дерево описания записи. Соответствующий транслятор построен при помощи СПТ WIRTH[1], вырабатывающей разреженное дерево анализа легенды; тело самого транслятора содержит процедуры преобразования этого дерева.

В ходе реализации проекта РАМА введены некоторые несущественные изменения в язык RDL. Во-первых, если в легенде нет повторяющихся или селективных групп, то характеристикой легенды может быть свойство PACK. Во-вторых, из свойств атома исключено свойство "указатель": в целях повышения скорости обработки ссылки DPOINT и RPOINT устанавливаются системой и являются недоступными для пользователя. В-третьих, в начале строки дополнительного доступа следует вместо символа "*" написать символ "¤". Наконец, в данной статье вводятся еще некоторые ограничения на язык RDL (напр., пределом количества размерностей массива ставится 15).

---

[1] Система построения трансляторов (СПТ), получающая название WIRTH, создана на ВЦ ТГУ (М.Томбак, А.Нигуль и др.).

1. Принципы трансляции

Каждая запись в системе РАМА должна иметь логическую структуру корневого дерева. Поэтому естественно представить и результат транслирования легенды (описания логической записи) в виде специального дерева описания записи, которое в целях краткости будем иногда называть также просто деревом.

Движение по физической записи сопровождается, как правило, движением по соответствующему дереву описания записи. Путь движения по дереву удобно определить координатами тех вершин дерева, которые находятся на этом пути. Вектор координат данной вершины дерева будем называть меткой этой вершины. Метки вершин дерева образуются следующим образом (см. [2], стр. 56).

Корень дерева не имеет метки. Метками вершин первого уровня являются их порядковые номера в множестве сыновей корня. Метка вершины i-того уровня ($i > 1$) состоит из метки отца, к которой после точки прибавляется порядковый номер этой вершины среди братьев.

Пример образования меток приведен на рис. 1, где кружки обозначают вершины легенды, буквы являются их именами, а метки находятся рядом с вершинами. Например, доступ к вершине J задается меткой 2.1.3. и может быть реализован таким образом:

```
A → B → C → B → H → I → J .
        2   1            3
```

Каждая вершина дерева описания записи представляется в виде 16-байтового поля, структура которого приведена на рис.2.

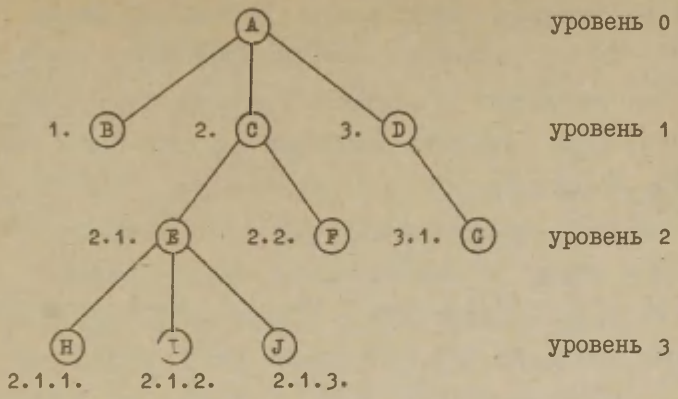


Рис. 1.

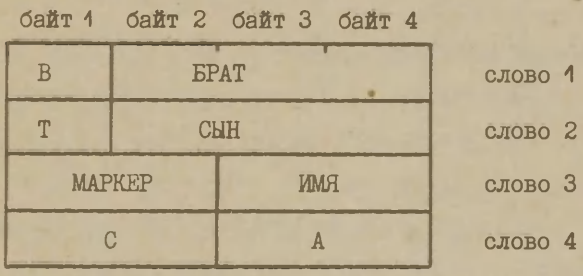


Рис. 2.

Подполе БРАТ этого поля содержит либо ссылку на соседнюю вершину (в этом случае B=0), либо ссылку на отца, если данная вершина является последней в своей группе (B=1). Поле СЫН содержит ссылку на первую подчиненную вершину; в висящей вершине это поле пусто. Поле ИМЯ предназначено в общем случае для ссылки на тот элемент таблицы имен, который содержит

имя вершины легенды, имя оформления и соответствующую данной вершине метку (в приводимых ниже примерах на это поле пишется просто имя вершины в скобках). В виде исключения это поле в корне дерева отводится для ссылки на специальную таблицу легенды (содержащую таблицу имен и некоторые другие таблицы).

Семантика остальных полей вершины дерева зависит от того, какой вершине легенды данная вершина соответствует. Все вершины дерева описания записи в этом смысле целесообразно разделить на четыре класса: корень дерева, вершины структуры, вершины организации и вершины сечения. Между вершинами легенды и вершинами дерева имеются следующие соответствия.

1. Каждой вершине легенды соответствует по меньшей мере одна вершина дерева.

2. Между вершинами легенды и дерева имеется взаимно-однозначное соответствие в том случае, когда они описывают атом, внешнюю структуру, корень неповторяющейся группы или корень альтернативной группы.

3. Копирующей вершине легенды соответствует лист, или поддерево, которое в точности совпадает с копируемым поддеревом.

4. Корню повторяющейся группы в дереве всегда соответствует более одной вершины, по меньшей мере их будет два - корень группы и вершина промежуточного уровня. Последняя в виде исключения имеет координату 0; в дереве описания записи у такой вершины нет соседей. Если предусмотрены основной и/или дополнительные доступы к экземплярам повторяющейся группы, а также если организацией группы не является REP, то в список вершин группы, содержащей корень, прибавляются одна или не-

сколько вершин организации. Например, отрывку легенды

```
* 2 H
  * 3 A NAT
  * 3 B HASH KEY= C REP TEXT
    * 4 C
    * 4 D
  * 3 E NAT
```

соответствует поддерево, представленное на рис. 3. На этом рисунке предполагается, что корень H имеет в дереве описания записи метку 3.1; метки вершин указаны над их полями. Вершины с метками 3.1, 3.1.1, 3.1.2, 3.1.4, 3.1.2.0.1 и 3.1.2.0.2 являются вершинами структуры и соответствуют вершинам легенды H, A, B, E, C и D соответственно; вершина структуры 3.1.2.0 является вершиной промежуточного уровня, а метку 3.1.3 имеет вершина организации, соответствующая доступу HASH.

5. Множественному атому в легенде соответствуют три вершины структуры в дереве описания записи: корень повторяющейся группы, вершина промежуточного уровня и вершина атома. Если заданы доступы, то в качестве соседей корня появляются еще соответствующие вершины организации. Например, если в легенде имеется отрывок

```
* 5 A NAT REP SORT
* 5 M TEXT
```

то ему соответствует поддерево, представленное на рис. 4.

6. Массив размерности n транслируется в поддерево, состоящее из корня повторяющейся группы, которому подчиняется

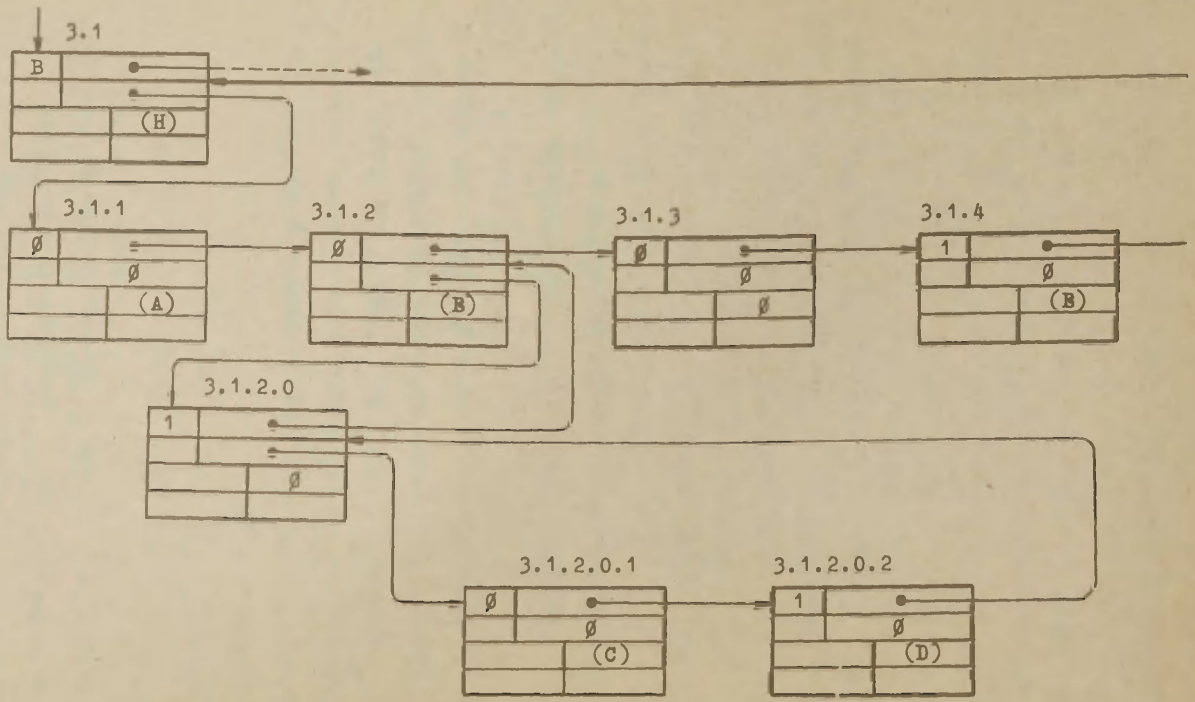


Рис. 3.

цепь из n вершин промежуточного уровня (связанных между собой ссылками на сын), последней из которых подчиняется список вершин, соответствующих содержанию массива. Корню могут при этом следовать еще вершины организации, если предусмотрены дополнительные доступы. На рис. 5 приведена схема поддерева, которое соответствует отрывку легенды

```
* 2 A ARRAY [7,3] NAT
  * 3 P
  * 3 Q
```

(при допущении, что меткой вершины A является 1.1).

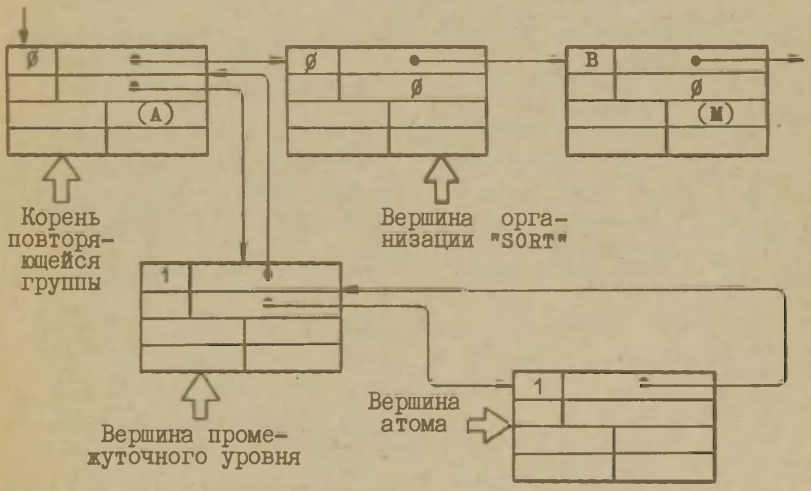


Рис. 4.

7. Альтернативная группа транслируется так же, как и неповторяющаяся группа.

8. Сечения атома транслируются как группа, корнем которой является атом. Каждому сечению соответствует одна вершина дерева.

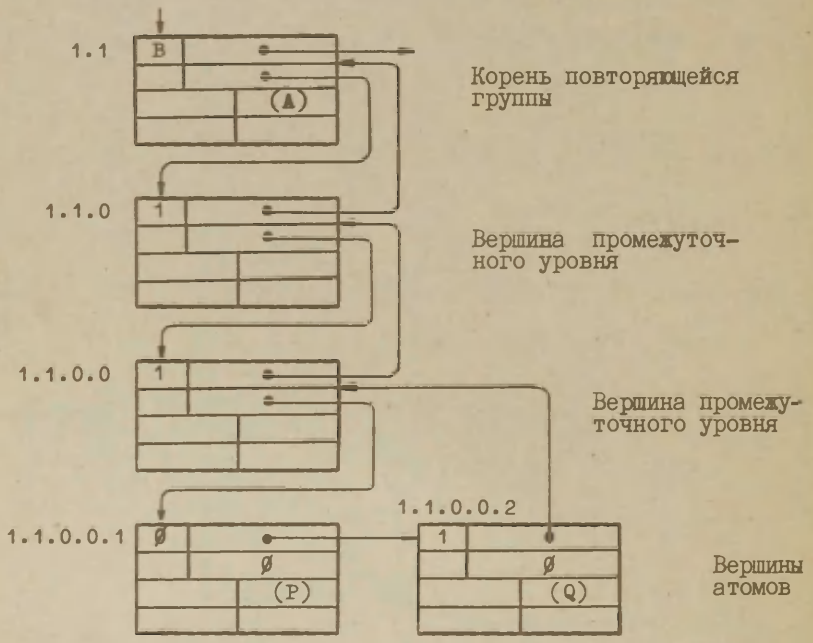


Рис. 5.

2. Маркировка вершин

Значение 16-битового подполя МАРКЕР (см. рис. 2) задает основную характеристику вершины. МАРКЕР разделяется на множество подполей, причем их семантика зависит от типа вершины. Сам тип определяется двумя старшими битами поля МАРКЕР, как указано в таблице 1.

Таблица 1

Тип вершины	Биты	
	0	1
Корень дерева описания записи	0	0
Вершина структуры	0	1
Вершина организации	1	0
Вершина сечения	1	1

Если вершина является корнем дерева или вершиной структуры, то биты 2...5 поля МАРКЕР носят информацию, указанную в таблице 2. Бит 6 поля МАРКЕР устанавливается в 1 тогда, когда корень дерева или вершина структуры имеет свойство РАСК.

Таблица 2

Спецификация вершины структуры	Биты			
	2	3	4	5
1. Висящая вершина:	0			
1.1. Атом без сечений	0	0	0	0
1.2. Атом со сечениями	0	0	1	0
1.3. Внешняя структура	0	1	0	0
2. Невисящая вершина:	1			
1.1. Корень группы:	1	0	0	
1.1.1 неповторяющейся группы	1	0	0	0
1.1.2 повторяющейся группы	1	0	0	1
1.2. Промежуточный уровень	1	0	1	0
1.3. Корень альтернативной группы	1	1	0	0

Семантика битов 2...4 поля МАРКЕР для вершин организации приведена в таблице 3, бит 6 указывает и здесь на наличие

свойства РАСК, а бит 5 не используется.

Таблица 3

Спецификация вершины организации	Биты		
	2	3	4
Основной доступ	1		
Дополнительный доступ	0		
Неопределенное количество повторений экземпляров группы		0	0
Количество повторений задано явно		0	1
Количество повторений задано ссылкой на атом типа NAT		1	0
Количество повторений задано ссылкой на атом со свойством SCOPE		1	1

В вершине сечения из битов 2...6 поля МАРКЕР используются только первые три, значение которых определяет тип сечения так, как указано в таблице 4.

Таблица 4

Тип сечения	Биты		
	2	3	4
Тип TEXT без ограничений	0	0	0
Тип TEXT с заданным запасом значений	0	0	1
Числовый тип без ограничений	0	1	0
Числовый тип с заданным максимальным значением	0	1	1
Числовый тип с заданным запасом значений	1	0	0

Биты 7...12 поля МАРКЕР не используются в том случае,

если вершина соответствует внешней структуре или корню неповторяющейся группы, или же является вершиной промежуточного уровня. В случае атома семантика этих битов указана в таблице 5 (соответствующий бит устанавливается в 1). Для маркировки атома биты 11 и 12 не используются.

Таблица 5

Свойство атома	Номер бита
Свойство CONST	7
Свойство NIL	8
Свойство PSEUDO	9
Свойство SCOPE	10

В таблице 6 указано, как значения битов 7 и 8 поля МАРКЕР задают для вершины сечения способ выделения сечения из значения атома численного типа. Если длина сечения выражается в полных байтах, а сечение выравнено на границе байта, то для его выделения не нужны никакие дополнительные средства. Если же эти условия не удовлетворяются, то применяется либо маска, либо константа сдвига, либо то и другое (см. [5]).

Таблица 6

Способ выделения сечения	Биты	
	7	8
Маска и константа сдвига отсутствуют	0	0
Задана маска		1
Задана константа сдвига	1	

Для корня повторяющейся группы, а также вершин организа-

ции битами 7...12 поля МАРКЕР характеризуются свойства, указанные в таблице 7.

Таблица 7

Спецификация	Биты					
	7	8	9	10	11	12
1. Организацией является:						
REP	0	0				
ARRAY	0	1				
LIST	1	0				
2. Вариант доступа:						
нет			0	0		
HASH			0	1		
SORT			1	0		
SORTDOWN			1	1		
3. Экземпляры имеют уникальный ключ					1	
4. Данная вершина сопровождается вершиной организации						1

В случае корня альтернативной группы битами 7 и 8 поля МАРКЕР указывается, как задан ключ выбора: если ключом является атом типа NAT с указанным максимальным значением, то в эти биты пишется значение 10, если же ключ выбора имеет запас значений, то значение 11 (биты 9...12 в такой вершине не используются).

Три последние бита поля МАРКЕР с порядковыми номерами 13...15 образуют подполе DYN и предназначены для указания типа кодослова, соответствующего данной вершине в физической

записи (см. [3] и [4]). Для этого предусмотрены следующие четыре варианта.

1. Тип кодослова имеет значение 000, если описываемые вершиной данные входят либо в состав упакованной группы, либо являются сечением атома. В таком случае описываемому элементу данных не соответствует кодослово.

2. Значение типа 001 указывает, что описываемым данным соответствует ссылка либо на атом, либо на упакованную группу. В дальнейшем будем такой тип называть "типом а".

3. Если неупакованный атом имеет длину 1...7 байтов, то он представляется в физической записи при помощи кодослова, содержащего атом. В поле МАРКЕР такой тип ("тип ь") имеет значение 010.

4. Корню любой неупакованной группы в физической записи соответствует ссылка на вектор, состоящий из кодослов. В такой вершине дерева описания записи тип кодослова ("тип с") имеет значение 011.

Назначение подполей Т, С и А вершины дерева (см. рис. 2) зависит от типа (а также от спецификации) вершины и будет описано в следующих главах.

3. Формат корня поддерева

В вершине, соответствующей корню поддерева, поля Т, С и А предназначены для информации о количестве элементов подчиненной группы и длине каждого элемента. Что является элементом, это зависит от спецификации вершины. Если вершина служит корнем неповторяющейся неупакованной группы, то элемен-

том считается кодослово атома или вложенной группы; если же такая группа упакована, то единственным элементом считается сама группа. В случае неупакованной повторяющейся группы элементом считается кодослово экземпляра группы, или же сам экземпляр, если повторяющаяся группа упакована.

Количество элементов неповторяющейся группы всегда устанавливается легендой (если группу следует упаковывать, то это количество равно 1). Для определения количества повторений экземпляров повторяющейся группы предусмотрено несколько возможностей, что и служит причиной использования нескольких форматов вершин дерева описания данных.

Формат определяется содержанием поля T (см. рис. 2), которое разделяется на два 4-битовые подполя T' и T". Первое из них используется только в том случае, когда вершина описывает корень n-мерного ($1 \le n \le 15$) массива: значением подполя T' будет количество размерностей n. Второе подполе T" может иметь одно из следующих значений.

1. Если количество повторений экземпляров повторяющейся группы не определено легендой, то T" = 0.

2. Если количество повторений задано легендой явно (вершина соответствует корню неповторяющейся группы, или количество повторений указано натуральным числом), то T" = 1.

3. Если количество повторений задано ссылкой на атом типа NAT со свойством CONST, то T" = 2.

4. Если количество повторений определено мощностью запаса значений некоторого атома со свойством SCOPE, то T" = 3.

Значение подполя C (см. рис. 2) во всех случаях указывает длину элемента: если кодослово имеет тип "a", то длина

выражается в байтах, если же тип "с", то в двойных словах. Таким образом, поле С совпадает с атрибутом длины в кодослове (см. [4]).

Семантика поля А зависит от значения Т": если Т" = 0, то А = 0; если Т" = 1, то А задает количество элементов; если же Т" > 1, то А является ссылкой либо на метку соответствующего атома, либо в таблицу запасов значений.

В таблице 8 показано, какие значения приобретают подполя С, А и DYN в зависимости от различных условий. Символ "Р" в заголовке таблицы указывает на свойство РАСК, символ "¬Р" значит, что группа не упакована. В клетках таблицы используются следующие обозначения:

* - отсутствие варианта;

q - количество элементов;

l_i - длина i-того элемента в байтах, i=1,2,...,q;

m - количество экземпляров повторяющейся группы;

⟶ - ссылка.

4. Формат атом-вершины

В дереве описания записи атом может оказаться либо действительным тупиком структуры, либо невисящей вершиной. Последнее имеет место тогда, когда легендой зафиксированы сечения этого атома. В физической записи атом всегда является элементом конечного уровня. Если атом имеет сечения, то в поле Т указывается количество сечений (длина цепи вершин сечений), а поле СЫН содержит ссылку на первую вершину сечения.

Поле С разделяется на две части (байта) D и Р, причем

Таблица 8

Спецификация вершины	поля	$T'' = \emptyset$		$T'' = 1$		$T'' > 1$	
		⌉P	P	⌉P	P	⌉P	P
Корень неповторяющейся группы	C	*	*	1	Σl_i	*	*
	A	*	*	q	1	*	*
	DYN	*	*	c	a	*	*
Корень повторяющейся группы	C	1	Σl_i	1	Σl_i	1	Σl_i
	A	0	0	m	m	⟶	⟶
	DYN	c	a	c	a	c	a
Корень альтернативной группы	C	*	*	*	*	1	*
	A	*	*	*	*	⟶	*
	DYN	*	*	*	*	c	*
Вершина промежуточного уровня	C	*	*	1	Σl_i	*	*
	A	*	*	q	1	*	*
	DYN	*	*	c	a	*	*

D = 0 означает тип кодослова "а", а D = 1 тип "ъ". В части P задается длина атома в байтах.

В поле A помещается ссылка на таблицу характеристик атома, которая имеет формат, указанный на рис. 6. Значение по-

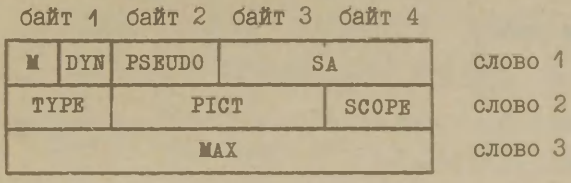


Рис. 6.

лубайта M приравняется нулю, если максимальное значение атома не зафиксировано. В этом случае подполе MAX отсутствует, и длиной таблицы является 8 байтов. При заданном максимальном значении M = 1, а таблица занимает 3 машинных слова. Само максимальное значение представляется в случае $P \leq 4$ на поле MAX. Когда $P > 4$, то поле MAX содержит ссылку на P-байтовое поле, представляющее это значение. Полубайт DYN уточняет физическое расположение атома следующим образом.

1. Значение DYN = 0 указывает, что на поле значения атома ссылает кодослово типа "а" (длина этого поля $P > 7$). Так как на этом поле находится только данный атом, то его относительный адрес SA = 0.

2. Значение DYN = 1 соответствует типу кодослова "ъ". Длина значения атома $P \leq 7$, а относительный адрес SA = 8 - P указывает на первый байт кодослова, занятый значением атома[2].

[2] Атрибут P устанавливается транслятором так, чтобы значение атома численного типа оказалось выравненным на границе полуслова, слова или двойного слова.

3. Значение DYN = 2 устанавливается тогда, когда атом входит в состав упакованной группы, которая определена как множественный атом. Тип кодослова, а также значение поля SA приравниваются в этом случае нулю.

4. Значение DYN = 3 показывает, что атом входит в состав упакованной группы, которая не является множественным атомом. Тип кодослова и в этом случае 0, а значение поля SA вычисляется как сумма длин атомов, предшествующих данному атому в экземпляре группы.

Поле TYPE предназначено для представления типа атома. Зафиксированы следующие коды:

00: тип NAT с длиной полного слова
01: NAT (полуслово)
02: NAT (1 байт)
10: INT (слово)
11: INT (полуслово)
20: REAL (слово)
21: REAL (двойное слово)
22: REAL (четыре слова)
30: DEC (фиксированной длины)
31: DEC (неопределенной длины)
40: HEX (фиксированной длины)
41: HEX (неопределенной длины)
50: DATE
51: FDATE
52: дата в машинном формате
60: TEXT (определенной длины)

61: TEXT (неопределенной длины)

FF: NIL - отсутствие значения.

Поле PICT указывает на формат печатного изображения значения атома. Второй байт поля задает количество знаков атома нечисленного типа или количество цифр перед запятой (точкой), первым байтом задается количество цифр за запятой для атома численного типа.

Поля PSEUDO и SCOPE используются тогда, когда атом имеет такие атрибуты. Значениями этих полей представляются порядковые номера элементов таблицы псевдоатомов и запасов значений соответственно (см. гл. 8).

5. Формат вершины сечения

В вершине сечения поле T дублирует часть информаций маркера: первый полубайт представляет тип сечения (возможные значения принадлежат по таблице 4 к промежутку 0...4), а второй полубайт задает способ выделения значения сечения (коды 0...3 по таблице 6).

Поле C разделяется на два байта: L и SA. Значение L задает длину сечения в байтах, а SA - относительный адрес первого байта сечения.

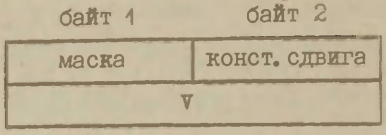
Использование поля A зависит от значения T следующим образом.

1. Если T = 00, то A = 0.

2. Если T = 0y (т.е. нет ограничений, а задан способ выделения значения y=1,2,3), то первый байт поля A содержит маску, а второй байт - константу сдвига. Если одна компонен-

та отсутствует, то ее значением будет 0.

3. Если T = zy (z=1,2,3,4; y=1,2,3), то значением A является ссылка на таблицу характеристик сечения:



где V может быть максимальным значением сечения (если его длина не превосходит 2 байтов), ссылкой на максимальное значение или индексом элемента таблицы запасов значений.

6. Формат вершины внешней структуры

Для транслятора вершина внешней структуры является висящей вершиной, а это значит, что поле СЫН пусто. Транслятором заполняются лишь поля В, БРАТ, МАРКЕР, ИМЯ и С (см. рис. 2). Значением поля С при этом является ссылка на имя внешней структуры, длина которого 8 байтов.

Поля СЫН и А предназначены теперь для формирования ссылок перед использованием дерева описания записи или во время работы с этим деревом: СЫН указывает на дерево, граф или таблицу описания данных внешней структуры, а поле А может быть использовано для ссылки на какие-то дополнительные характеристики внешней структуры. Заполнение поля Т также остается в компетенции пользователя данной внешней структурой.

7. Формат вершины организации

Вершины организации сопровождают корни повторяющихся групп и содержат информацию об организации таких групп или о способе доступа к их экземплярам. В физической записи таким вершинам дерева соответствуют кодослова, ссылающие на таблицы определенной структуры. Если повторяющаяся группа определена без указания ключа и варианта доступа, а ее организацией не является списочная структура, то для такой группы не понадобится ни одной вершины организации.

Формат вершины организации следующий: поля В, БРАТ и МАРКЕР несут такую же информацию, что и в предыдущих случаях, ИМЯ используется лишь в вершинах, соответствующих дополнительным доступам, а поле СЫН всегда имеет значение 0.

Значение поля Т задает порядковый номер данной вершины организации в списке таких вершин у корня повторяющейся группы. Если вершина должна содержать информацию о ключе доступа (или уникальности), то значение поля А принимается равным порядковому номеру элемента таблицы ключей.

Поле С содержит ссылку на макет таблицы организации или доступа. Формат макета зависит от разновидности описания группы или доступа; макет дает возможность сэкономить время инициирования соответствующих таблиц в физической записи во время работы. В качестве примера приведем в таблице 9 макет доступа путем хэширования[3] (длина макета 24 байта).

---

[3] В макете заполнены лишь те поля, которые не отмечены звездочкой; последние получают значения в таблицах, соответствующих макету в физической записи. В интересах наглядности здесь представлены значения всех полей. Запись "(*)" означает, что, возможно, поле заполняется в таблице, а не в макете.

Таблица 9

Отн. адрес	Назначение	*
0	чтение = 0, запись = 1	*
1	длина ключа в байтах	
2	количество полных слов значения ключа	
3	остаток значения ключа (если длина не делится на 4)	
4	длина звена хэш-таблицы в полусловах	
6	длина звена дополнительной цепи в полусловах	
8	количество экземпляров	*
10	SCOPE-индекс	
12	максимальное предполагаемое количество экземпляров	(*)
14	длина хэш-таблицы	(*)
16	маркер вершины организации	
18	порядковый номер элемента таблицы ключей	
20	длина таблицы	(*)
22	первый свободный адрес поля звеньев	*

В качестве метода хэширования используется разрешение коллизий методом цепочек (см. [6], стр. 610), хэш-функция взаимствована у Д.Гриса ([7], стр. 255-256): машинное слово S' вычисляется при помощи поразрядного сложения слов и остатка значения ключа, а хэш-адрес вычисляется как остаток деления S'/m, где m - длина хэш-таблицы (последняя вычисляется, как наименьшее простое число, удовлетворяющее условию $m \geq n$, где n - максимальное количество экземпляров).

Если требуется уникальность ключей, то дополнительные цепи не образуются, а форматом звена служит следующий 4-байтовый вектор:

полуслово 1	полуслово 2
ссылка по коллизии	ссылка на экземпляр

Если одно и то же значение ключа может встречаться у нескольких экземпляров (как это допускается при дополнительном доступе), то длиной звена будет 6 байтов - прибавляется поле для ссылки на начало дополнительной цепи.

8. Формат корня дерева

В корне дерева помимо информации о группе первого уровня представляются еще данные, характеризующие дерево в целом. На рис. 7 приведена схема формата корня. Семантика полей Т

байт 1	байт 2	байт 3	байт 4	
Ø		Ø		слово 1
Ø	1	ссылка на дерево		слово 2
МАРКЕР		ссылка на таблицу		слово 3
С		А		слово 4

Рис. 7.

(значение которого теперь 01), С и А такая же, что в вершине, соответствующей корню неповторяющейся группы. Поле МАРКЕР имеет теперь возможные значения, указанные в таблице 10.

Таблица, на которую ссылает второе полуслово третьего слова вершины, состоит из заголовка длиной в 30 байтов, таблицы имен и, если понадобится, таблиц ключей, псевдоатомов и запасов значений. Формат заголовка приведен в таблице 11.

Таблица 10

Биты	Значение	Семантика
0 и 1	0 0	признак корня
2	1	невисящая вершина
3 и 4	0 0	корень группы
5	0	группа не повторяется
6	0	группа не упаковывается
	1	запись состоит из одной единственной упакованной группы
7...12	0	не используются
13...15	0 1 1	тип "с" кодослова, если бит 6 = 0
	0 0 1	тип "а" кодослова, если бит 6 = 1

Таблица 11

Отн. адрес	Содержание
0	имя легенды
8	длина дерева описания записи
12	предполагаемая длина записи
16	количество ключей
18	ссылка на таблицу ключей
20	ссылка на таблицу имен
22	количество запасов значений
24	ссылка на таблицу запасов значений
26	количество псевдоатомов
28	ссылка на таблицу псевдоатомов

Таблица имен организована как хэш-таблица с внешними цепочками. Пользователю может представлять интерес формат звена цепи омонимов, приведенный на рис. 8. Метка представляется как последовательность двухбайтовых полей. Обратим внимание еще на то, что если имя связывается с повторяющимся атомом или атомарным массивом, то метка содержит как метку кор-

ня повторяющейся группы, так и нули, соответствующие вершинам промежуточного уровня, а также единицу в качестве последней координаты.

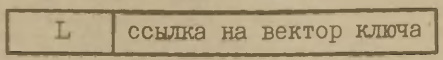
байт 1	байт 2	байт 3	байт 4	
длина метки	длина имени оформления	ссылка на метку		слово 1
ссылка на имя оформления		ссылка на следующее звено		слово 2

Рис. 8.

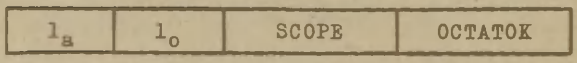
Каждая цепь омонимов связывается с именем вершины, сечения или дополнительного доступа из легенды. Если это имя уникально, то цепь состоит из одного единственного звена; в общем же случае длина цепи равняется количеству объектов с данным именем. Так как требуется только уникальность каждого имени в пределах одной группы, то допустимость омонимов проверяется путем сравнивания меток: если они все различимы, то требование уникальности удовлетворяется. Распознаватель сложных имен дает на выход метку, соответствующую рассматриваемому сложному имени. Не останавливаясь на алгоритме распознавателя, отметим, что он отличается от известных (например, описанного в [8]) и направлен на работу с минимальными сложными именами. Например, если имеются вершины с именами B.A и C.A, то имя A идентифицирует ту вершину из них, которой соответствует меньшая метка в смысле лексикографической упорядоченности, другую же вершину следует наименовать сложным именем.

Таблица ключей содержит k записей, где k - количество

объявленных в легенде ключей доступа. Каждая запись данной таблицы является 4-байтовой и распадает на два поля:



Значение поля L задает количество компонент ключа (сложных имен)[4], а вектор ключа состоит из L полей следующей структуры:



Через l_a здесь обозначена длина атома ключа, уменьшенная на единицу (в байтах), а l_o – длина остатка его метки. Поле SCOPE задает порядковый номер записи таблицы запасов значений, соответствующий данному атому ключа, если он имеет запас значений (в противном случае SCOPE = 0), а ОСТАТОК – это в общем случае ссылка на остаток метки. Под "остатком метки" подразумевается та часть метки атома ключа, которая начинается от координаты корня данной ветки. Применение такой части метки становится возможным благодаря требованию, что ключ должен находиться в ветке данной группы (см. [1], стр. 13 и 40). Если длиной остатка является 1, то единственная координата представляется непосредственно на поле ОСТАТОК.

Таблица псевдоатомов построена аналогично таблице ключей. Длина таблицы, как и максимальное количество псевдоато-

---

[4] Синтаксис определения ключа см. [1], стр. 39.

мов в одной легенде, не может превысить 255. Формат записи таблицы представляется таким образом:

байты 1 и 2	байт 3	байт 4	байты 5 и 6	байты 7 и 8
тип PSEUDO	тип атома	длина атома	ссылка N	ссылка P

"Ссылка N" указывает на звено цепи омонимов таблицы имен, соответствующее данному атому со свойством PSEUDO. "Тип" и "длина" соответствуют также указанному атому. "Тип PSEUDO" имеет 3 различных значения. Семантика этих значений следующая:

1. Тип PSEUDO имеет значение 4 тогда, когда в легенде данное свойство объявлено без параметров, т.е. в вершине легенды встречается только слово PSEUDO. Значение такого атома не вводится, а вычисляется; способ и время вычисления остаются в компетенции пользователя. Поле "ссылка P" в этом случае не используется.

2. Значение 8 типа PSEUDO означает, что в легенде при помощи PSEUDO определяется константа, тип и длина которой заданы байтами 3 и 4 в записи таблицы. Если длина константы не превосходит 2 байтов, то сама константа является значением поля "ссылка P"; в общем же случае это поле содержит ссылку на значение константы.

3. Если в легенде (явно или неявно) задана функция вычисления значения псевдоатома, то значением типа PSEUDO является 12. Говорим, что функция задана явно, если в легенде имеется имя этой функции, например, PSEUDO = TODAY( ) или PSEUDO = MEAN1 (ТАБЕЛЬ) (см. [1], стр. 23 и 57). Неявно заданными функциями считаются свойства атома INDEX и "дополни-

тельное упорядочение": INDEX интерпретируется транслятором как функция PSEUDO = INDEX( ), а "дополнительное упорядочивание" как функции

$$\text{PSEUDO} = \left\{ \begin{array}{l} \text{BASE} \\ \text{BASEDOWN} \end{array} \right\} (p_1, p_2, \ldots, p_n),$$

где p_i (i=1,2,...,n) является i-тым сложным именем в составе ключа упорядочения. "Ссылка P" указывает теперь на таблицу, структура которой приведена на рис. 9.

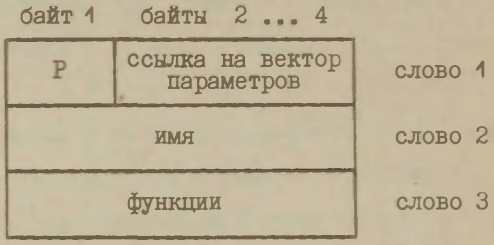
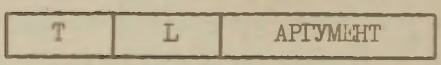


Рис. 9.

Если функция не имеет параметров, то первое слово таблицы имеет значение 0; в общем же случае значение P задает количество параметров, а вектор параметров состоит из P полей следующего формата:



.

Возможные значения подполей T, L и АРГУМЕНТ представлены в таблице 12.

Таблица 12

Тип параметра	T	L	АРГУМЕНТ
сложное имя	04	длина метки	ссылка на метку
число длиной до двух байтов	08	0	число формата полуслова
число длиной выше двух байтов	12	выравненная длина в байтах	ссылка на число
строка символов (стринг) длиной выше двух байтов	16	длина строки в байтах	ссылка на строку
строка символов длиной до двух байтов	20	длина строки в байтах	строка

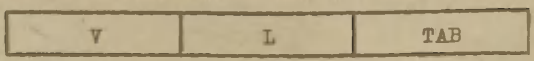
Таблица запасов значений имеет структуру, близкую к структуре таблиц псевдоатомов и ключей. Верхней границей количества атомов с объявленными запасами значений в одной легенде является 255. Каждому запасу соответствует одна запись таблицы в таком формате:

байты 1 и 2	байт 3	байт 4	байты 5 и 6	байты 7 и 8
тип SCOPE	тип атома	длина атома	ссылка N	ссылка S

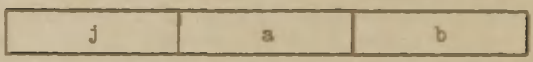
Тип и длина атома, а также "ссылка N" имеют такую же семантику, что соответствующие поля в записи псевдоатома (с той разницей, что здесь атом имеет свойство SCOPE). В зависимости от характера объявленного запаса значений предусматриваются три варианта его физического представления.

1. Тип SCOPE = 4 означает, что запас задается в легенде

как SCOPE = [p_1,p_2,...,p_n], где p_i является константой (числом или буквой), или интервалом чисел или букв. Предполагается, что n является сравнительно небольшим, а мощность множества возможных значений - сравнительно большой. "Ссылка S" указывает в данном случае на вектор следующей структуры:



Количество различных возможных значений (т.е. мощность запаса) данного атома представляется на поле V; значение поля L задает количество элементов n в таблице, на которую ссылает содержание поля TAB. Формат элемента этой таблицы зависит от типа атома. Если типом не является REAL, то элемент содержит следующие 3 поля:



Поле "a" содержит либо значение j-того ($1 \leq j \leq V$) элемента запаса значений (если длина атома не превышает 2 байта), либо ссылку на этот элемент. Если в определении запаса i-тый компонент p_i задан как одиночное значение, то поле "b" в i-том элементе таблицы пусто; в обратном случае это поле содержит либо последнее значение интервала, либо ссылку на это значение.

Например, пусть среди характеристик какого-то атома типа NAT имеется свойство

SCOPE = [2, 7-9, 14-100].

Соответствующая этому свойству данного атома запись в таблице запасов приведена на рис. 10 (значения переменных представлены шестнадцатеричными числами).

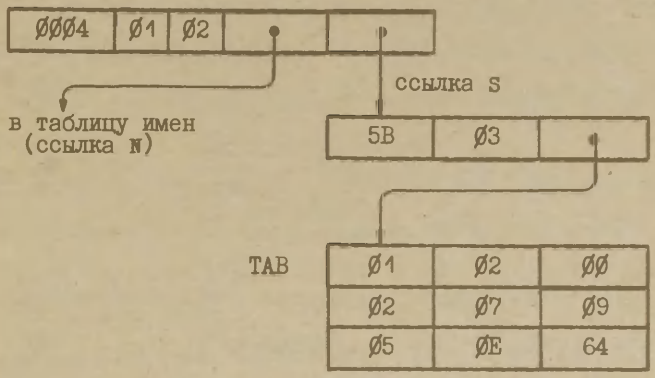


Рис. 10.

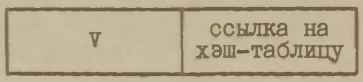
Замысел представленного формата элемента таблицы запасов заключается в том, что атом со свойством SCOPE может выступать в роли ключа выбора: порядковый номер его значения в запасе значений служит в таком случае индексом экземпляра повторяющейся группы или альтернативного поддерева.

Ключем выбора не может быть атом типа REAL. Если у такого атома имеется установленный запас значений, то им пользуются лишь для проверки логической правильности данных. В таком случае значение V задает длину атома в байтах, а каждый элемент таблицы, ссылаемой при помощи указателя TAB, состоит из двух V-байтовых подполей:

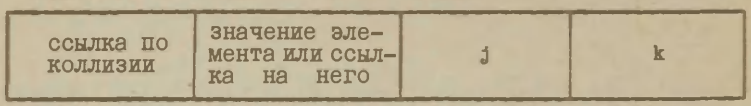
.

5

2. Тип SCOPE = 8 означает, что мощность множества значений является сравнительно небольшой, количество элементов p_i в определении запаса сравнительно большое, а элементы p_i заданы все (или почти все) как одиночные значения. В таком случае запас значений хэшируется, а "ссылкой S" в записи таблицы запасов указывается на вектор:

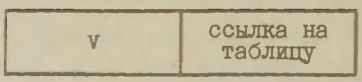


где через V опять обозначена мощность запаса значений. Формат звена цепей хэш-таблицы следующий:



Значение j задает порядковый номер элемента запаса значений в соответствии с определением запаса (j=1,2,...,V), а k - в обратном порядке (k=V,V-1,...,1).

3. Тип SCOPE = 12 означает, что как мощность запаса, так и количество элементов p_i небольшие. Таким типом определяется просматриваемая таблица, "ссылка S" указывает на поле



а таблица состоит из V полей, на которых представлены допустимые значения атома.

Литература

1. Изотамм А., Каазик Ю., Томбак М., Язык определения записи. Труды ВЦ ТГУ, 1978, № 41, 7-64.
2. Виллемс А.Л., Изотамм А.А., Язык манипулирования данными в системе VILLIS. Труды ВЦ ТГУ, 1976, № 38, 40-103.
3. Изотамм А., Физическое представление записи в системе РАМА. Наст. сборник, 36-54.
4. Нигуль А., Система динамического распределения памяти для иерархических структур данных. Наст. сборник, 55-73.
5. Микли Т.Ю., Томбак М.О., Причципы организации больших массивов информации в системе "СОДИ". Труды Таллинского Политехнического Института, 1971, серия А, № 313, 21-29.
6. Кнут Д., Искусство программирования для ЭВМ. т. 3, "Мир", Москва, 1978.
7. Грис Д., Конструирование компиляторов для цифровых вычислительных машин. "Мир", Москва, 1975.
8. Gates, G.W., Poplawski, D.A., A Simple Technique for Structured Variable Lookup. CACM, September 1973, Vol. 16, Numb. 9, 561-565.

ФИЗИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ ЗАПИСИ В СИСТЕМЕ РАМА

А. Изотамм

В этой статье рассматривается структура физической записи СУБД РАМА в оперативной памяти ЭВМ. Так как физическое представление данных зависит от соответствующего дерева описания записи, а последнее - физическое представление легенды, определяющей запись на языке RDL, то данная статья является логическим продолжением статей [1] и [2]. Эту серию мы намерены продолжить. Так, например, в следующей статье серии будут освещены способы и программные средства доступа к данным, представленным на полях записи. Поэтому в настоящей статье эти вопросы, как правило, и не рассматриваются.

1. Список записей. Заголовок записи

В системе РАМА все физические записи хранятся в виде их объединений - файлов (см. [9]). При этом в одном файле могут содержаться записи, соответствующие нескольким разным легендам. В оперативную память обычно вводится только некоторое подмножество записей данного файла и такое подмножество изображается как списочная структура, заголовок которой ("заголовок файла") содержит, между прочим, головку цепи легенд, при помощи которых описаны физические записи данного файла.

Таблица 1

Отн. адрес	Содержание
0	Имя легенды
8	Ссылка на заголовок файла
12	Ссылка на следующее звено цепи легенд
16	Ссылка на дерево описания записи
20	Ссылка на цепь физических записей, описанных данной легендой
24	Адрес корня дерева описания записи
28	Адрес таблицы легенды
32	Адрес таблицы имен
36	Адрес таблицы ключей
40	Адрес таблицы запасов значений
44	Адрес таблицы псевдоатомов

Каждое звено этой цепи имеет формат, представленный в таблице 1. Элементы звена, начиная с относительного адреса 24, дублируют таблицу легенды (см. [2]), с той лишь разницей, что здесь адреса представляются как абсолютные. Следует учесть, что в каждую цепь записей включаются только те записи, соответствующие данной легенде, которые в данный момент находятся в оперативной памяти. Принципиальная схема указанных ссылок приведена на рис. 1.

Память для цепей выделяется из динамической области памяти при помощи макрокоманды GETMAIN с указанием номера подпула (см. [8]).

Поле каждой записи является связной областью, которая разделяется на две части: заголовок и дерево данных. На рис. 2 представляется общая структура этой области, причем подробно указана структура заголовка (структура дерева данных

рассматривается в следующих главах).

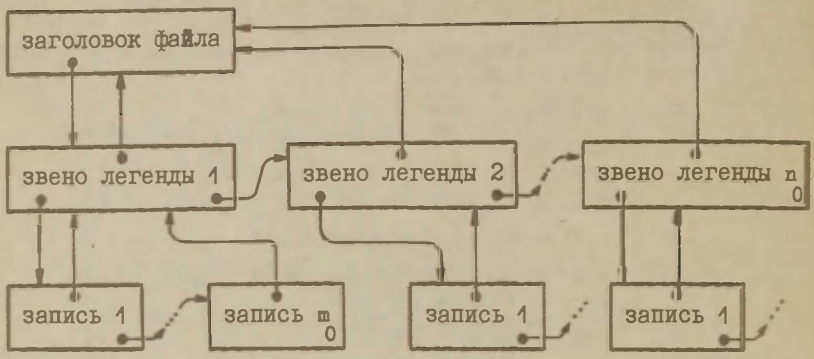


Рис. 1.

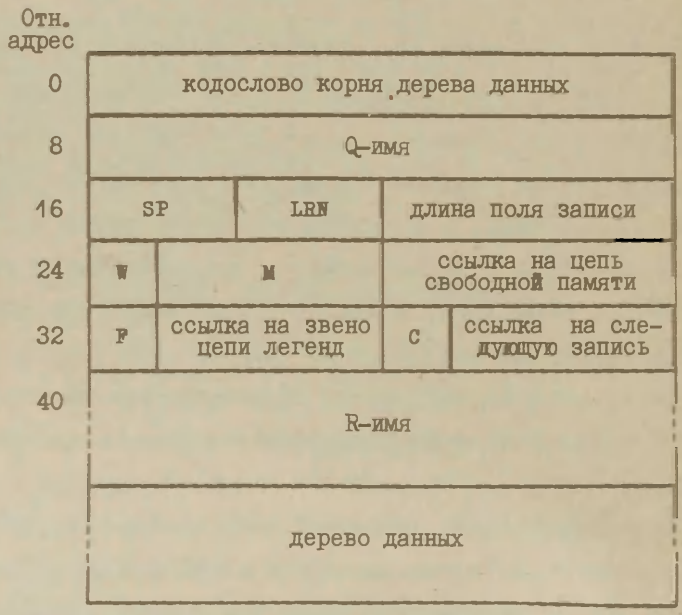


Рис. 2.

Заголовок записи состоит из 40-байтового подполя, к которому прибавляются еще 1...32 двойных слов для т.н. R-имени. Имена в заголовке предназначены для координации пользования записью: они являются параметрами макрокоманд ENQ и DEQ системы ОС ЕС [8]. Через Q-имя на рис. 2 обозначено имя очереди (им является либо имя легенды, либо имя комплекта - в смысле языков FDL [9] и DML [10]), через LRN - длина имени ресурса, а через R-имя - имя ресурса (в качестве этого имени используется значение ключа записи). Значением подполя SP является номер того подпула памяти, где находится данная запись. Цепь свободной памяти образуется в дереве данных записи. Ссылки на звено цепи легенд и на следующую запись показаны на рис.1 стрелками соответственно вверх и направо. Длина поля записи используется программами вывода записи на печать или на внешние накопители (а также в качестве параметра макрокоманды FREEMAIN). Подполе M содержит ссылку на стэк, создаваемый программами обработки записи. Подполя W, F и C служат для помещения разных флажков, состояния которых показывают, например, доступна ли запись программам прибавления или исправления данных, доступны ли экземпляры повторяющихся групп по ключам дополнительных доступов (см. [1], стр. 42) и т.п. Наконец, кодослово корня дерева данных дает как относительный адрес этого дерева в поле записи, так и характеристику единственного экземпляра группы первого уровня (см. [1]).

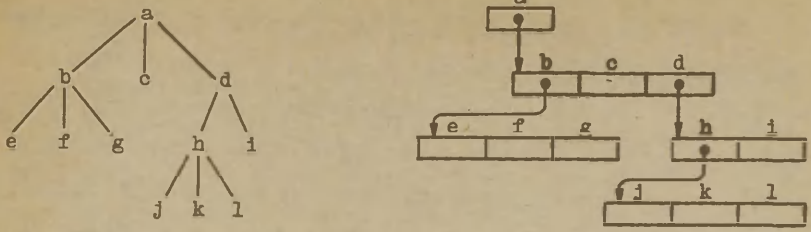
2. Дерево данных

Дерево данных строят на основе дерева описания записи [2]. Следуя общепринятой терминологии скажем, что дерево описания является для дерева данных логическим деревом. Формой изображения физического дерева данных в системе РАМА выбран супермассив. Такая структура данных определяется в статье [4] и используется, например, в системах СОДИ и VILLIS (см. [5], [6] и [7]).

Супермассив выражает, в первую очередь, иерархическую структуру данных. Сами данные связываются с висящими вершинами дерева, т.е. элементами супермассива на самом низком уровне иерархии. Элементами являются кодослова (см. [3] и [4]) - поля памяти фиксированной структуры и длины. Кодослова промежуточных уровней ссылают на векторы кодослов следующего, более низкого уровня, а кодослово последнего уровня может либо ссылаться на данные, либо содержать эти данные.

На рис. 3 показано, как конкретное логическое дерево представляется в виде супермассива. Все кодослова в супермассиве помечаются таким же образом, как и вершины дерева описания записи (см. [2]). Например, меткой кодослова с на рис. 3 является 2, а кодослово 1 имеет метку 3.1.3 .

Структуры, представленные на рис. 3, находятся во взаимно-однозначном соответствии в том смысле, что одна и та же метка идентифицирует как вершину дерева описания записи, так и соответствующее данной вершине кодослово в физической записи. Однако, это не всегда так. Например, элементам упакованных поддеревьев не соответствуют кодослова; физическое

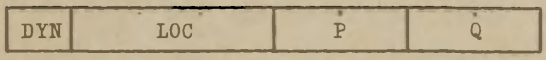


логическое дерево супермассив

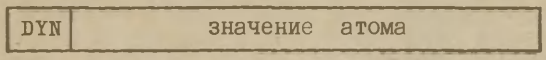
Рис. 3.

представление вершин промежуточного уровня дерева описания записи зависит от организации повторяющейся группы и т.д. Правила установления соответствия между описанием записи и данными представляются в следующих разделах.

Для построения супермассива применяется система распределения памяти, которая описывается в [3]. Однако, поскольку дерево данных имеет довольно специфическую структуру, то более уместным оказалось использовать эту систему в несколько модифицированном виде. Для изображения физической записи применяются кодослова только трех типов, которые именованы типами "a", "b" и "c" (ср. [2]). Их 8-байтовые форматы указаны на рис. 4.



кодослово типа "a" или "c"



кодослово типа "b"

Рис. 4.

Содержание подполя DYN в кодослове задает короткую характеристику данных, связанных с этим кодословом. Длина этого подполя – 1 байт, а семантика значений каждого бита показана в таблице 2. Если условие, приведенное в графе "семантика", удовлетворено, то значение соответствующего бита 1. Значение двух последних битов определяет тип кодослова.

Таблица 2

Номер бита	Семантика
1	Кодослово является фиктивным[1] или ссылает на внешнюю структуру
2	Кодослово ссылает на списочную структуру
3	Поддерево является упакованным
4	Ссылаемое поле можно удлинять
5	В данных поддерева обнаружена ошибка
6	Начало поддерева не выравнено на машинное слово
7 и 8	Значение 01 определяет тип "a", 10 – тип "ъ", а 11 – тип "c"

Подполе LOC в кодослове содержит относительную ссылку на поддерево, а подполя P и Q предназначены для указания длины ссылаемого поля. Если типом кодослова является "a", то значение P задает длину элемента в байтах, а значение Q показывает количество элементов. В кодослове типа "c" значение P определяет количество кодослов в блоке (см. [3]), а значение Q – количество блоков.

---

[1] Фиктивные кодослова в дереве записи не участвуют. Ими пользуются некоторые системные процедуры, например, для образования ссылки на сечение атома.

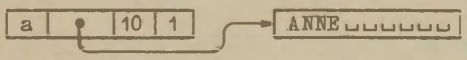
3. Висящие вершины

Висящими вершинами дерева данных считаются все кодослова самого низкого уровня в супермассиве. С их помощью изображаются следующие объекты: атом, корень упакованной группы, внешняя структура и таблица организации, соответствующая вершине организации в дереве описания записи.

Физическое представление неупакованного атома зависит от его длины. Если эта длина не превосходит 7 байтов, то значение атома размещается в кодослове типа "ъ" (см. рис. 4). Например, если длина имени школьника установлена в легенде равной 6 байтам, то соответствующее поле памяти может выглядеть следующим образом:

.

Атомы длиной выше 7 байтов представляются при помощи кодослова типа "а". Например, если длина имени установлена в 10 байтов, то конкретное имя предыдущего примера представляется так:



.

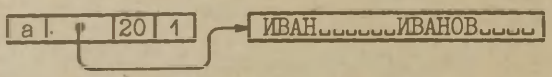
Если атом имеет неопределенную длину, то он представляется либо кодословом типа "ъ", либо типа "а". В последнем случае в поле атома нет неиспользованных байтов.

На поле данных упакованной группы ссылает всегда кодослово типа "а". Если группа является неповторяющейся, то значение подполя Q в кодослове (см. рис. 4) равно 1, а значение подполя P задает длину соответствующего поля данных. Напри-

мер, если в легенде определено поддерево

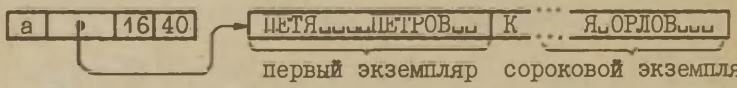
```
* 1 ДИРЕКТОР TEXT PICT=10 PACK
  * 2 ИМЯ
  * 2 ФАМИЛИЯ
```

то ему может соответствовать следующий фрагмент супермассива:



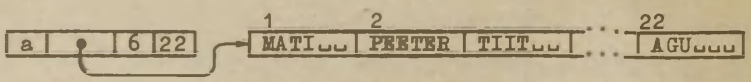
В случае повторяющейся упакованной группы значение P также задает длину экземпляра в байтах, а значение O - количество экземпляров. Такой вариант иллюстрируется следующим примером:

```
* 2 УЧЕНИКИ TEXT REP PICT=8 PACK
  * 3 ИМЯ
  * 3 ФАМИЛИЯ
```



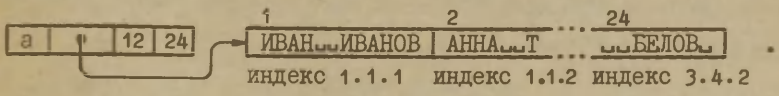
Таким же образом представляется и множественный атом, если предусмотрена упаковка его значений. Экземпляром теперь является значение атома, например:

```
* 7 ДЕТИ PICT=6 REP PACK TEXT
```



Атрибуты P и Q в кодослове упакованного массива соответственно показывают длину элемента и их количество, например:

```
* 3 СОТРУДН ARRAY [3,4,2] PICT=6 РАСК
  * 4 ИМЯ TEXT
  * 4 ФАМИЛИЯ TEXT
```



Ссылка на внешнюю структуру также находится в кодослове самого низкого уровня, причем типом кодослова может быть либо "a", либо "c". Признаком тупика на пути движения по супермассиву служит 1 в первом бите подполя DYN (см. табл. 2). Если внешняя структура представляется в виде отдельного супермассива, то данное кодослово является его корнем, а пути движения в ссылаемой структуре начинаются с этого корня и определяются в описании внешней структуры.

Каждой вершине организации дерева описания записи соответствует по одному кодослову, в котором дается ссылка на таблицу организации. Такое кодослово принадлежит всегда типу "a", причем значение подполя P задает длину таблицы в байтах, а Q=1. Сама таблица состоит в общем случае из заголовка (который соответствует макету, связанному с вершиной организации в дереве описания) и поля для цепей ссылок (см. [2]), хотя некоторые таблицы состоят только из заголовка.

4. Невисящие вершины

В виде невисящих вершин супермассива представляются следующие вершины дерева описания записи: корень неупакованной неповторяющейся группы, корень альтернативной группы, корень

повторяющейся неупакованной группы с организацией REP вместе с подчиняющейся ему вершиной промежуточного уровня, корень и вершины промежуточного уровня массива, а также корень любой (упакованной или неупакованной) повторяющейся группы с организацией LIST. Соответствующее кодослово, как правило, принадлежит типу "c", исключение представляет лишь упакованная LIST-группа, корень которой изображается кодословом типа "a". В кодослове типа "c" ссылаемое поле памяти содержит только кодослова и разделяется на блоки одинаковой длины. Значение подполя P дает количество кодослов в блоке, а значение Q - количество блоков.

Неповторяющейся группе в физической записи соответствует один блок, длина которого - количество элементов группы. Например, на рис. 5 показан участок супермассива, который соответствует отрывку легенды:

```
* 1 ДИРЕКТОР TEXT
  * 2 ИМЯ PICT=4
  * 2 ФАМИЛИЯ PICT=10
```

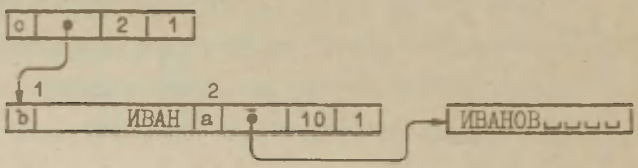


Рис. 5.

Кодослово корня повторяющейся группы с организацией REP ссылает на один или несколько блоков. Если количество экземпляров группы определено, то оно является значением подполя

P, а Q=1. Если же предусматривается неопределенное количество экземпляров, то длиной блока считается 16 кодослов. Ссылаемое таким кодословом поле памяти соответствует вершине промежуточного уровня в дереве описания записи. Таким образом, в этом случае одной вершине описания записи соответствует множество вершин физической записи. Все кодослова в одном поле промежуточного уровня имеют одинаковый тип, т.е. они могут представлять либо атомы типа "a" или "b" (если повторяющаяся группа организована как множественный атом), либо корни поддеревьев типа "c".

В качестве примера допустим, что в отрывках легенд из предыдущей главы упаковка данных не требуется. Модифицированные таким образом легенды и соответствующие им отрывки супермассивов выглядят теперь следующим образом. Отрывку легенды

```
* 2 УЧЕНИКИ REP=20 TEXT
  * 3 ИМЯ PICT=7
  * 3 ФАМИЛИЯ PICT=8
```

соответствует отрывок супермассива, показанный на рис. 6.

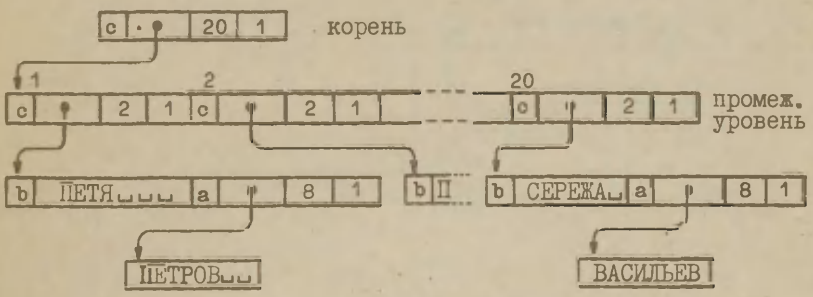


Рис. 6.

Физическое представление отрывка легенды

```
* 7 ДЕТИ PICT=6 REP=40
```

приведено на рис. 7 (данный пример представляет единственный случай, когда вершине промежуточного уровня соответствует вектор кодослов конечного уровня), а отрывку легенды

```
* 3 СОТРУДН ARRAY [3,4,2] PICT=6 TEXT
  * 4 ИМЯ
  * 4 ФАМИЛИЯ
```

соответствует рис. 8.

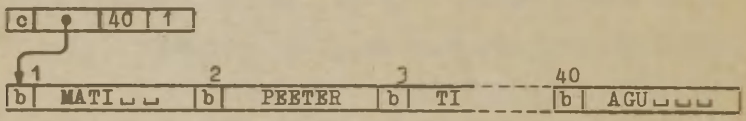


Рис. 7.

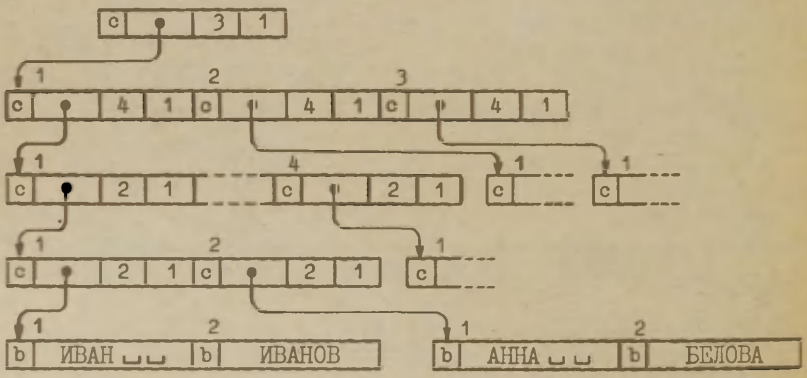


Рис. 8.

Корень альтернативной группы представляется так же, как

и корень неповторяющейся группы. Количество кодослов в подчиненном блоке соответствует количеству альтернативов, но в каждом конкретном случае среди этих кодослов может быть только одно непустое кодослово (пустое кодослово имеет значение 0). Например, в [1], на странице 29 приведен участок легенды:

```
* 2 МАЛЫШ
  * 3 A SCOPE=[ДОМА, ЯСЛИ, САД]
  * 3 B CASE=A NAT
    * 4 ДОМАШНИЙ NIL
    * 4 НОМЕР 'НОМЕР ЯСЛЕЙ'
    * 4 САД
      * 5 АДРЕС TEXT PICT=40
      * 5 НОМЕР 'НОМЕР САДА'
      * 5 ГРУППА MAX=15 'НОМЕР ГРУППЫ'
```

Если атом A имеет значение САД, то соответствующий отрывок супермассива может быть таким, как показано на рис. 9.

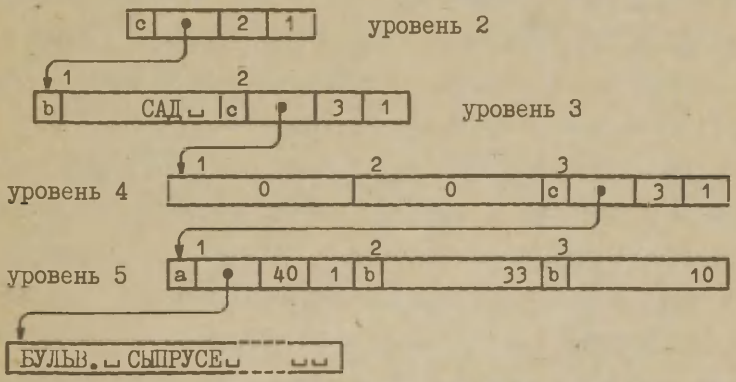
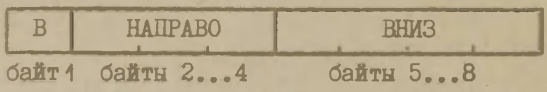


Рис. 9.

Если организацией повторяющейся группы является LIST, то вершине промежуточного уровня в дереве описания не соответствует ни одной вершины в супермассиве. Каждый экземпляр группы представляется полем памяти следующего формата:

Поле ссылок занимает при этом 8 байтов и разделяется на три подполя:



Значение подполя ВНИЗ дает относительную ссылку на звено списка более низкого уровня (если такого нет, то значением ссылки является 0). Значение байта B определяет семантику подполя НАПРАВО: если B = 0, то НАПРАВО содержит относительную ссылку на следующее звено того же уровня, если же B = 1, то ссылается на то звено предыдущего уровня, в котором ссылка ВНИЗ указывает на начало данного подсписка.

Поле данных содержит либо кодослова (если группа не упакована), либо сами данные. Корень повторяющейся LIST-группы представляется как кодослово типа "a" или "c" в зависимости от упакованности. Признаком свойства LIST служит 1 во втором бите поля DYN; поле LOC ссылает на первое звено списка, значение подполя Q всегда 1, а значение P дает длину экземпляра в байтах (тип "a") или в двойных словах (тип "c").

Например, если в легенде имеется отрывок

```
* 3 ЗВЕНО TEXT PICT=7 LIST
  * 4 ИМЯ
  * 4 ФАМИЛИЯ
```

то соответствующая часть супермассива принимает вид, указанный на рис. 10. Если же в корень этой группы добавить свойство РАСК, то соответствующее физическое представление приведено на рис. 11 .

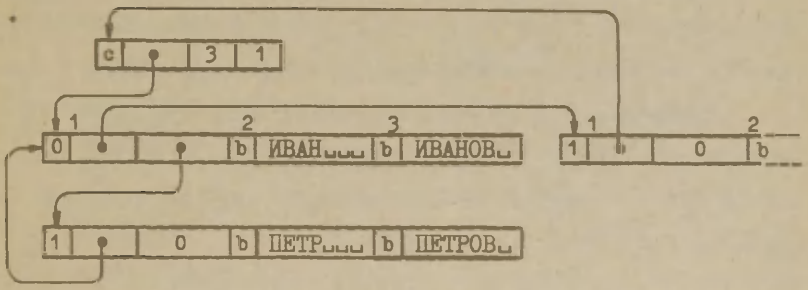


Рис. 10.

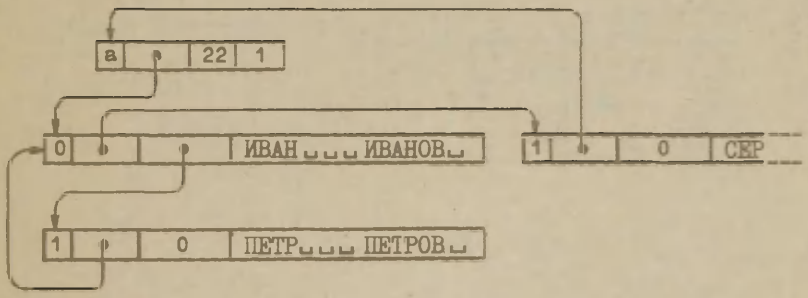


Рис. 11.

5. Представление доступа

Физическое представление вариантов доступа к экземплярам повторяющейся группы осуществляется следующим образом.

1. Если в легенде не определяется свойство "доступ", то экземпляры группы (не смотря на организацию REP или LIST) размещаются в супермассиве в порядке их поступления.

2. Если вариантом основного доступа является SORT или SORTDOWN, то в случае REP-организации кодослова промежуточного уровня поместятся в порядке возрастания или убывания значений ключей доступа соответствующих экземпляров, а при LIST-организации таким образом поместятся сами звенья в каждой подцепи в отдельности. Например, если значения ключа в экземплярах соответственно A, B, C и D, то повторяющаяся REP-группа с доступом SORT выглядит так, как показано на рис. 12.

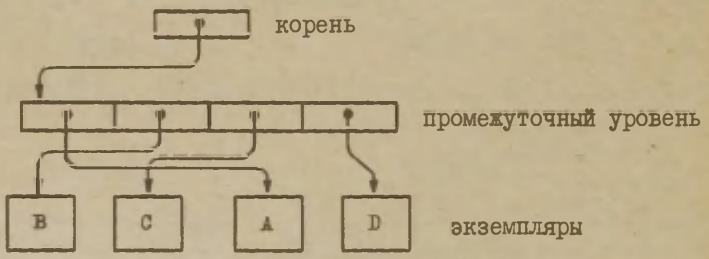


Рис. 12.

Если же организацией предусмотрен LIST, то картина может быть такой, как представлено на рис. 13.

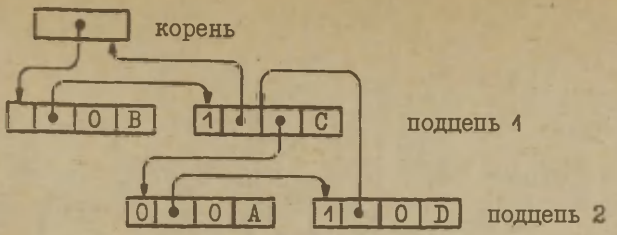


Рис. 13.

3. Если вариантом основного доступа служит HASH, то повторяющаяся группа представляется физически таким же образом, как и группа без доступа, а таблицей организации является хэш-таблица, содержащая ссылки на экземпляры группы. Кодослово той таблицы находится рядом с кодословом корня группы.

4. Все дополнительные доступы без исключения не повлияют на расположение кодослов в блоке промежуточного уровня REP-группы или звеньев списка; они реализуются аналогично основному доступу HASH при помощи таблиц организации.

Литература

1. Изотамм А., Каазик Ю., Томбак М., Язык определения записи. Труды ВЦ ТГУ, 1978, № 41, 7-64.
2. Изотамм А., Дерево описания данных в системе РАМА. Наст. сборник, 3-35.
3. Нигуль А., Система динамического распределения памяти для иерархических структур данных. Наст. сборник, 55-73.
4. Томбак М.О., Микли Т.И., Аллик К.К.-Э., Об отношении субординации. Труды ВЦ ТГУ, 1974, № 30, 8-22.
5. Выханду Л.К., Об интегрированных системах обработки дискретной информации. Труды Таллинского Политехнического Института, 1971, серия А, № 313, 3-14.
6. Аус Т.А., Рябовыйтра М.Г., Томбак М.О., Генератор матричных отчетов. Труды ВЦ ТГУ, 1974, № 30, 23-29.
7. Виллемс А.Л., Изотамм А.А., Томбак М.О., Генератор отчетов "VILLIS". Труды ВЦ ТГУ, 1974, № 30, 49-66.
8. Система математического обеспечения ЕС ЭВМ. Под ред. А.М. Ларионова, "Статистика", Москва, 1974.
9. Каазик Ю., Образование файлов. Труды ВЦ ТГУ, 1978, № 41, 65-74.
10. Каазик Ю., Рауп А., Язык манипулирования данными. Труды ВЦ ТГУ, 1978, № 41, 97-140.

СИСТЕМА ДИНАМИЧЕСКОГО РАСПРЕДЕЛЕНИЯ ПАМЯТИ ДЛЯ ИЕРАРХИЧЕСКИХ СТРУКТУР ДАННЫХ

А. Нигуль

Для описания и обработки структур данных некоторого определенного вида полезно иметь соответствующие базовые средства, при помощи которых элементарные операции отображения и модификации структур данных осуществлялись бы достаточно просто и эффективно. Так как при разработке системы РАМА для управления базой данных [4] выяснилось, что средства операционной системы ОС ЕС не всегда удовлетворяют требованиям системы РАМА, возникла необходимость создания соответствующих базовых средств. Рассматриваемая в настоящей статье система DYN и является средством отображения и обработки структур данных, ориентированным потребностям системы РАМА.

Прообразом при создании системы DYN служила система представления структуры данных, разработанная для ЭВМ "Минск -32" [2,3]. Особенностью нашей системы является то, что оперативная память, контролируемая системой, разделяется на участок, предназначенный для построения структур данных, и на участки для сохранения указателей на элементы структур данных. Процедуры системы DYN, описанные в настоящей статье, реализованы на ЭВМ ЕС-1022 и объединены в систему программ.

1. Представление структур данных

Элементы структур данных и их расположение относительно друг-друга представляют в системе DYN при помощи т.н. кодослов [1,2], каждое из которых имеет имя. Кодослово в нашей системе - это четверка вида

$$(\mathrm{TYPE},P,Q,\mathrm{LOC}) ,$$

где компонента TYPE называется типом данного кодослова. Остальные компоненты описывают некоторое поле памяти, которое мы будем называть набором этого кодослова. В системе DYN наборы кодослов могут содержать данные и/или другие кодослова. Компоненты P и Q кодослова характеризуют длину соответствующего набора, а компонента LOC дает адрес набора.

Если имя кодослова обозначить через K и набор этого кодослова содержит некоторое непустое множество кодослов, то через K.i обозначим имя i-того кодослова в наборе K. Ясно, что запись K.i имеет смысл только в том случае, когда набор кодослова K содержит по меньшей мере i кодослов. Если теперь набор кодослова K.i содержит в свою очередь некоторые кодослова, то имя j-того кодослова в наборе K.i можем обозначать через K.i.j . В общем случае, выражение вида

$$K.i_1.i_2.\ \cdots\ .i_n ,$$

где K - имя кодослова и $i_1,i_2,\ldots,i_n$ - натуральные числа, будем называть составным именем с базой K.

Будем говорить, что кодослово с именем L достижимо из кодослова K, если существует конечная последовательность на-

туральных чисел $i_1, i_2, \ldots, i_n$ такая, что $L = K.i_1.i_2. \cdots .i_n$.

В системе DYN выделено т.н. главное кодослово ROOT, набор которого состоит только из кодослов (в дальнейшем этот набор будет называться полем BASE или базисным полем). Структура всех других кодослов в системе DYN удовлетворяет следующему требованию. Для произвольного кодослова с именем K, отличного от ROOT:

(1) кодослово K содержится в наборе хотя бы одного кодослова структуры;

(2) кодослово K достижимо из кодослова ROOT;

(3) кодослово K не достижимо из кодослова K.

Используя систему DYN для построения или модификации структур данных, пользователь не имеет прямого доступа к элементам структур данных. Доступ (ссылка) к произвольному кодослову возможен лишь составным именем с базой ROOT, т.е. выражением вида $ROOT.i_1.i_2.\cdots.i_n$. Это значит, что для ссылки на некоторое кодослово достаточно иметь выражение вида $(i_1, i_2, \ldots, i_n)$ - последовательность натуральных чисел, которую назовем меткой и будем рассматривать вместо имени соответствующего кодослова. Если в дальнейшем будем говорить, что у нас имеется некоторое кодослово, то имеем в виду, что у нас имеется метка этого кодослова. Отметим, что главное кодослово ROOT для пользователя недоступно.

2. Классификация кодослов

По своим функциям при описании структур данных кодослова разделяются в следующие классы:

(1) кодослова терминального типа – кодослова, наборы которых содержат только данные и ни одного кодослова;

(2) кодослова нетерминального типа – кодослова, наборы которых состоят только из кодослов;

(3) кодослова смешанного типа – кодослова, наборы которых содержат как кодослова, так и данные.

Рассматриваются еще т.н. пустые кодослова (или кодослова типа NIL), которые вообще не имеют набора.

Подробная классификация кодослов определяется на основе их типов. Тип кодослова (TYPE, P, Q, LOC) рассматривается как пятерка

$$\text{TYPE} = (\alpha, \beta, \gamma, \delta, \varepsilon) ,$$

где в качестве значений компонент участвуют неотрицательные целые числа 0,1,2. Для пустого кодослова принимается TYPE = = (0,0,0,0,0). Ниже при классификации будем рассматривать только непустые кодослова.

Компонента α в типе кодослова называется признаком указателя. Если $\alpha = 0$, то LOC дает начальный адрес набора кодослова. Если же $\alpha = 1$, то LOC является ссылкой на начальный адрес набора кодослова. В этом случае говорим, что начальный адрес кодослова – указатель, а LOC – ссылка на указатель. Требуется, чтобы указатель находился вне того участка памяти, который подчиняется системе DYN (например, указателем может быть некоторое поле в программе пользователя). Сам указатель находится под контролем системы, которая гарантирует пользователю наличие там правильного начального адреса набора.

Компонента ε в типе кодослова называется признаком копирования. Если $\varepsilon = 0$, то набор кодослова уникален. Если же $\varepsilon = 1$, то будем говорить, что данное кодослово $((\alpha,\beta,\gamma,\delta,1), P,Q,LOC)$ является копированным, т.е. в структуре кодослов существует по крайней мере одно отличное от данного кодослова кодослово $((\alpha_1,\beta,\gamma,\delta,1),P,Q,LOC_1)$, которое имеет тот же набор. Ясно, что в случае $\alpha \neq \alpha_1$ должно быть $LOC \neq LOC_1$ (см. рис. 1).

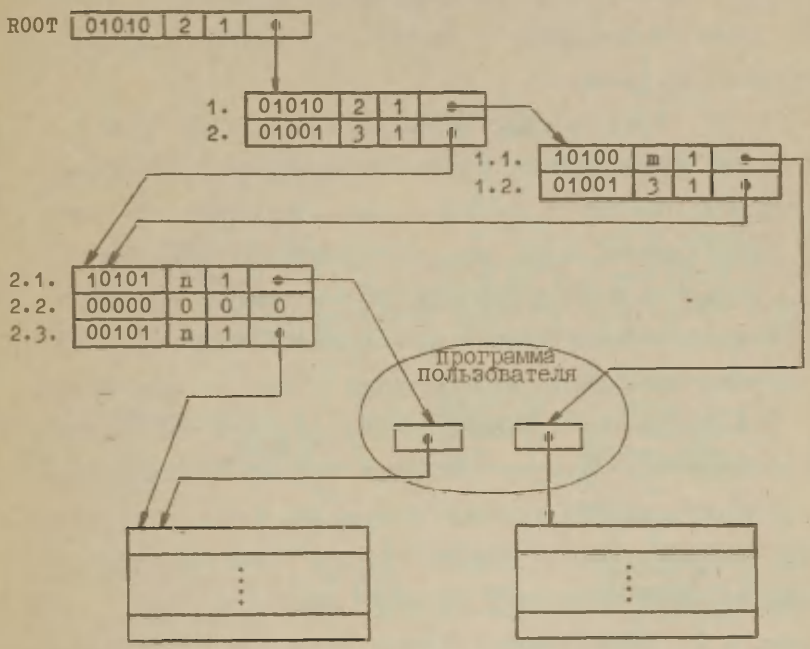


Рис. 1.

Система DYN дает пользователю возможность удлинять и сокращать наборы всех кодослов за исключением сокращения по-

ля BASE. В конкретной реализации системы длины наборов указываются в двойных словах, а в качестве длины кодослова зафиксировано двойное слово (8 байтов).

Рассмотрим теперь подробнее кодослова терминального типа, в которых $\beta = 0$. Если при этом $\gamma = 1$, то такое кодослово определяет т.н. атом, структурой которого система DYN не интересуется. Если $\delta = 0$, то компонента P кодослова указывает длину набора и $Q = 1$. В случае $\delta = 1$ набор кодослова разделен на некоторые блоки: P указывает длину одного блока и Q – количество блоков в наборе ($Q \geq 1$). Изменять длину такого набора можно только по блокам.

Если в кодослове терминального типа $\gamma = 2$, то будем говорить, что данное кодослово определяет атомарную повторяющуюся группу. Это значит, что набор кодослова разделен на участки равной длины – экземпляры повторяющейся группы. Компонента P указывает длину экземпляра (в конкретной реализации эта длина указывается в байтах), а Q – число экземпляров. Удлинять или сокращать такой набор можно только по экземплярам.

В кодословах нетерминального типа $\beta = 1$, а компонента γ не учитывается. Если $\delta = 0$, то компонента P кодослова указывает число кодослов в наборе данного кодослова (т.е. длину этого набора) и $Q = 1$. Если же $\delta = 1$, то набор кодослова разделен на блоки кодослов, P указывает число кодослов в одном блоке, а Q – число блоков. В этом случае длину набора можно изменять только по блокам.

В случае $\beta = 2$ мы имеем дело с кодословом смешанного типа, т.е. соответствующий набор содержит как кодослова, так и атом. Предполагается, что атом и множество всех кодослов на-

бора размещаются компактно (в конкретной реализации, например, все кодослова расположены "перед атомом"). Изменять в таком наборе можно как длину атома, так и число кодослов – в частном случае кодослово смешанного типа может переходить в кодослово терминального или нетерминального типа (обратное невозможно).

Так как пустое кодослово можно рассматривать как частный случай кодослова терминального типа, то классификация кодослов описывается схемой, приведенной на рис. 2.

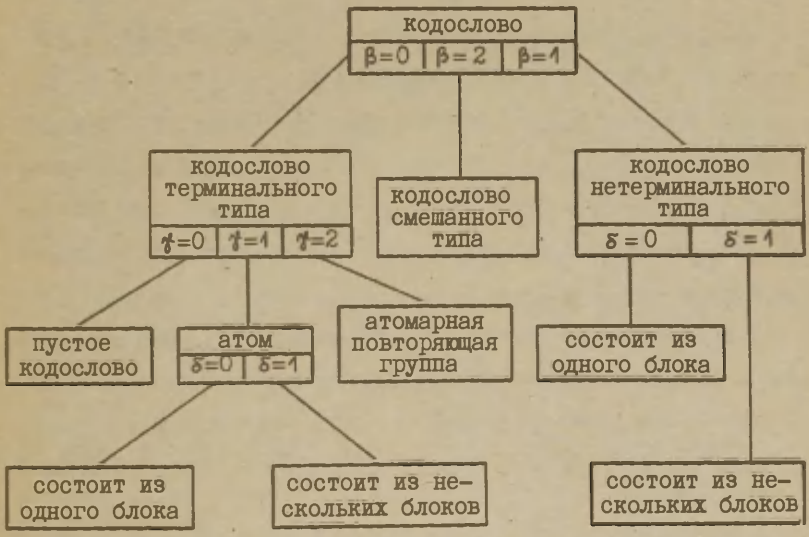


Рис. 2.

3. Описание системы

Распределение памяти организуется в системе DYN при помощи т.н. кусочно-свободной структуры [3]. Это значит, что подчиненная системе память разделяется на некоторое число активных (занятых) участков и на некоторое число неактивных (свободных) участков. Последние связываются в список свободной памяти. Проблемы, связанные с такой структурой оперативной памяти, подробно изложены в монографии [5].

Рассмотрим теперь коротко главные внутренние процедуры системы DYN.

Процедура CUT(n,s,l,A) служит для резервирования участка памяти. В случае n > 0 активизируется участок памяти длиной в n двойных слов. При нехватке свободной памяти проверяется значение параметра l: в случае l = 1 возникнет ситуация ошибки, а в случае l = 0 применяется процедура GARCOL для "сборки мусора" (если и после этого памяти не хватает, то возникает ситуация ошибки).

В случае n = 0 необходимо активизировать максимальный возможный участок памяти. Если при этом l = 0, то прежде всего применяется процедура GARCOL. После резервирования участка его длина (в двойных словах) засылается на место параметра n.

Начальный адрес зарезервированного участка памяти во всех случаях засылается на место параметра A, этот участок заполняется символами s и корректируется список свободной памяти.

Процедура FREE(A,n) служит для освобождения участка памяти. Работа этой процедуры заключается в освобождении активного участка памяти длиной в n двойных слов с начальным адресом A и включении этого участка в список свободной памяти.

Процедура GARCOL осуществляет "сборку мусора" в подчиненной системе DYN части памяти, т.е. минимизирует число элементов списка свободной памяти. В результате применения этой процедуры подчиненная системе память состоит из одного активного и одного свободного участка.

Процедура FORK(A,B) предназначена для просмотра всех кодослов, достижимых из кодослова с адресом A. При первом обращении к этой процедуре параметр B имеет значение 1, при последующих обращениях - значение 0. Процедура найдет адрес первого еще не выданного кодослова, достижимого из данного кодослова и засылает этот адрес на место параметра B. Если все достижимые кодослова просмотрены, значение B не изменяется. В процедуре предусмотрена защита от зацикливания просмотра, каждое кодослово выдается однократно.

Процедура SHIFT(A,s,n) осуществляет удлинение набора кодослова с адресом A на n двойных слов. Если это оказывается необходимым, то будут перемещаться некоторые активные участки памяти, а соответствующие кодослова корректируются. Добавленная часть набора заполняется символами s. В случае недостатка свободной памяти возникнет ситуация ошибки.

Процедура LEVEL($(i_1,i_2,\ldots,i_n)$,A) - это инструмент для доступа к кодослову. Список параметров $(i_1,i_2,\ldots,i_n)$ рассматривается как метка соответствующего кодослова. Если $i_n \neq 0$, то проходится путь, соответствующий данному составному имени и в результате на место параметра A выдается адрес кодослова ROOT.$i_1.i_2. \cdots .i_n$. Ситуация ошибки возникает тогда, когда данной метке не соответствует кодослово.

В случае $i_n = 0$ процедура ищет первое пустое кодослово в наборе кодослова K с меткой $(i_1,\ldots,i_{n-1})$, пусть меткой такого кодослова является $(i_1,\ldots,i_{n-1},j)$. Тогда на место параметра i_n засылается значение j, а на место A - адрес кодослова ROOT.$i_1.i_2. \cdots .i_n$. В случае, когда в рассматриваемом наборе нет пустых кодослов, проверяется, будет ли K вообще кодословом нетерминального типа и чему равен $\delta(K)$. Если $\delta(K) = 0$ или $\beta(K) \neq 1$, то возникнет ситуация ошибки. В противном случае ($\beta(K) = \delta(K) = 1$) применяется процедура SHIFT для удлинения набора K на один блок. Теперь требуемое пустое кодослово с меткой $(i_1,\ldots,i_{n-1},j)$ существует, выдается его адрес и $i_n := j$.

Отметим еще, что реализация системы допускает довольно легко заменить стратегии как резервирования и удлинения участков памяти, так и сборки мусора.

4. Пользование системой DYN

Рассмотрим теперь основные средства системы DYN, предназначенные для пользователя системой. Таковыми являются процедуры START и RESTART для инициализации системы и изме-

нения ее состояния, процедуры DECL и COPY для создания структуры кодослов (структуры данных), процедуры LONG и SHORT для изменения длин наборов (занятых участков памяти) и процедура DELETE для гашения подструктур. Доступ к кодословам (к элементам структуры данных) осуществляется при помощи описанной выше процедуры LEVEL.

Процедура START(A,B,P,Q) предназначена для инициализации системы DYN. Параметры P и Q описывают создаваемое поле BASE, остальные задают участок памяти, переходящий в подчинение системы. Предусмотрены следующие варианты:

(1) в случае B = 0 параметр A указывает длину участка в процентах от максимальной возможной длины, которую допускает операционная система (память резервируется системой DYN);

(2) в случае B = 1 параметр A указывает длину участка в K байтах (память резервируется системой);

(3) в случае B = 2 параметр A указывает длину в K байтах той части памяти, которая не переходит в распоряжение системы DYN (остальная часть памяти резервируется системой);

(4) во всех других случаях A и B задают начальный и конечный адрес того участка памяти, который переходит в подчинение системы, но резервирован пользователем.

Работа процедуры состоит в резервировании памяти (если это требовалось), образовании базисного поля и формировании системных параметров. Если $P \neq 0$ и $Q \neq 0$, то они рассматриваются как требуемые компоненты кодослова ROOT; в противном случае параметрам P и Q присваиваются значения по умолчанию. Для образования базисного поля на место кодослова

ROOT засылается пустое кодослово и к нему применяется процедура DECL (при TYPE = (0,1,0,1,0), $P_1 = P$, $Q_1 = Q$, s = 0, l = 0, C = 0). В результате получается поле BASE, заполненное кодословами типа NIL.

Отметим, что повторное применение процедуры START без применения процедуры RESTART вызывает ошибку.

Процедура RESTART(t,P,Q) служит для изменения состояния системы DYN. Если t = 0, то система переходит в пассивное состояние: аннулируются все системные параметры, все указатели заполняются нулями и освобождается подчиняющаяся системе память (последнее в том случае, если память резервировалась системой). Отметим, что пассивной системой работать невозможно: следует применить процедуру START.

Если t = 1, то аннулируется вся существующая структура кодослов, все указатели заполняются нулями и производится новая инициализация системы (при этом участок памяти, подчиненный системе, остается прежним). Как и в процедуре START, при $P \neq 0$ и $Q \neq 0$ эти параметры интерпретируются в качестве требуемых компонент кодослова ROOT. В противном случае (P = 0 или Q = 0) значения P и Q определяются по умолчанию. Затем на место кодослова ROOT засылается пустое кодослово и к нему применяется процедура DECL с теми же значениями параметров, что и при процедуре START. В результате получается базисное поле, состоящее из $P \cdot Q$ пустых кодослов.

Процедура DECL($(i_1,\ldots,i_n)$,TYPE,P_1,Q_1,s,l,C) предназначена для резервирования участка памяти, причем $(i_1,\ldots,i_n)$ является меткой соответствующего кодослова. Так как в систе-

ме DYN принято, что структура может продолжать свой "рост" только начиная с кодослов типа NIL, то кодослово K с меткой $(i_1,\ldots,i_n)$ должно быть пустым. Если это не так, то возникнет ситуация ошибки. Параметры TYPE, P_1, Q_1 рассматриваются как требуемые компоненты составляемого на место K нового кодослова и проверяется их согласованность (в случае несогласованности могут быть применены некоторые процедуры изменения значений параметров по умолчанию). Затем применяется процедура CUT(n,s,1,LOC), где значение n вычисляется по значениям TYPE, P_1, Q_1. В случае C = 0 на место кодослова K формируется кодослово (TYPE,P_1,Q_1,LOC) и если набор этого кодослова содержит кодослова, то они заменяются кодословами типа NIL.

Если $C \neq 0$, то значение C трактуется как адрес указателя и вместо кодослова (TYPE,P_1,Q_1,LOC) формируется кодослово (TYPE,P_1,Q_1,C). После этого на место указателя засылается значение LOC.

Процедура COPY($(i_1,\ldots,i_n)$,$(j_1,\ldots,j_k)$,C) служит для копирования кодослов. Здесь метка $(i_1,\ldots,i_n)$ должна определить кодослово типа NIL, а метка $(j_1,\ldots,j_k)$ - некоторое непустое кодослово (TYPE,P,Q,LOC) (иначе возникнет ситуация ошибки). Так как адреса этих кодослов получаются при помощи процедуры LEVEL, то разрешается и случай $i_n = 0$ (то же относится к процедуре DECL).

В случае C = 0 сначала проверяется, является ли кодослово K_1 с меткой $(i_1,\ldots,i_n)$ достижимым из кодослова K_2 с меткой $(j_1,\ldots,j_k)$. Если это так, то возникнет ситуация ошибки. В

противном случае при $\alpha(K_2) = 0$ принимают $\varepsilon(K_2) := 1$ и затем на место кодослова K_1 формируется полученное кодослово (TYPE,P,Q,LOC). Если же $\alpha(K_2) = 1$, то LOC означает ссылку на указатель и после засылки $\varepsilon(K_2) := 1$ на место K_1 формируется кодослсво ($TYPE_1$,P,Q,LOC_1), где $TYPE_1$ отличается от TYPE лишь значением компоненты α, а LOC_1 – адрес набора, полученный из указателя по адресу LOC.

Если $C \neq 0$, то значение C дает адрес нового указателя. В этом случае при $\alpha(K_2) = 1$ на место K_1 формируется кодослово (TYPE,P,Q,C), а при $\alpha(K_2) = 0$ кодослово ($TYPE_1$,P,Q,C), причем указатель заполняется адресом набора кодослова K_2.

Рассмотрим для примера структуру кодослов, изображенную на рис. 1. Такую структуру можно получить последовательным применением процедур START, DECL и COPY следующим образом (здесь A_1 и A_2 обозначают некоторые адреса указателей, а n и m – натуральные числа):

```
START(5,1,2,1);
DECL((0),(0,1,0,1,0),2,1,0,0,0);
DECL((1,1),(1,0,1,0,0),m,1,0,0,A2);
DECL((0),(0,1,0,0,0),3,1,0,0,0);
COPY((1,0),(2),0);
DECL((2,3),(0,0,1,0,0),n,1,0,0,0);
COPY((2,0),(2,3),A1).
```

Процедура DELETE($i_1,\ldots,i_n$) предназначена для гашения подструктуры кодослов. Если кодослово K с меткой ($i_1,\ldots,i_n$) является пустым, то никаких действий не производится. В про-

тивном случае работа процедуры зависит от значения $\varepsilon(K)$.

Если $\varepsilon(K) = 1$, т.е. кодослово K является копированным, то освобождается память из-под всех кодослов и полей данных, достижимых из кодослова K; при этом соответствующие указатели заполняются нулями (для освобождения памяти пользуются процедурами FORK и FREE). Одновременно с гашением во вспомогательный список N собирают все такие кодослова L, для которых $\varepsilon(L) = 1$ (в список N включается и K). После полного гашения при помощи процедуры FORK просматриваются все сохранившиеся кодослова и каждое такое из них, которое совпадает с некоторым кодословом списка N (с точностью до указателя), заменяется пустым кодословом.

Если же $\varepsilon(K) = 0$, то поступают следующим образом:

(1) в список M собирают все достижимые из кодослова K кодослова $L = K.j_1. \cdots .j_k$, для которых $\varepsilon(L) = 1$, а все $\varepsilon(K.j_1. \cdots .j_m) = 0$ (при $m=1,2,\ldots,k-1$); после этого принимают TYPE(L) := (0,0,0,0,0);

(2) освобождается память из-под всех достижимых из кодослова K кодослов и полей данных;

(3) просматриваются все сохранившиеся кодослова, пересчитывая числа вхождений кодослов, идентичных (с точностью до указателя) с кодословами списка M; составляется список N такой, что N[i] (i-тый элемент списка N) равен числу вхождений кодослова M[i] в структуру;

(4) просматривается список N: если N[i] = 1, то принимают ε(M[i]) := 0, а к кодословам M[i], для которых N[i] = 0, применяют процедуру DELETE; в заключение на место кодослова K засылается пустое кодослово.

Рассмотрим для примера опять структуру, представленную на рис. 1. Результатом выполнения процедуры DELETE(1) теперь будет структура, изображенная на рис. 3. Если же применить процедуру DELETE к кодослову с меткой (2,1), то получится структура, приведенная на рис. 4.

Процедура LONG$((i_1,\ldots,i_n),n,s,t)$ осуществляет удлинение набора кодослова K с меткой $(i_1,\ldots,i_n)$. Если это кодослово является пустым, то возникнет ситуация ошибки. В противном случае действие процедуры зависит от типа кодослова K. Если $\beta(K) = 0$ и $\gamma(K) = 2$ (т.е. кодослово K определяет атомарную повторяющуюся группу), то набор кодослова K удлиняется на n экземпляров. В случае $\beta(K) = 2$ K является кодословом смешанного типа и его набор удлиняется на n двойных слов, причем при $t = 0$ добавляются кодослова, а при $t = 1$ удлиняется атом.

Во всех остальных случаях набор кодослова K удлиняется на n двойных слов (при $\delta(K) = 0$) или на n блоков набора (при $\delta(K) = 1$).

Во всех случаях добавленные поля данных заполняются символами s, добавленные кодослова заменяются пустыми кодословами и корректируются компоненты кодослова K. Если $\varepsilon(K) = 1$, то корректируются и все соответствующие копированные кодослова структуры.

Процедура SHORT$((i_1,\ldots,i_n),n,t)$ предназначена для сокращения набора непустого кодослово K с меткой $(i_1,\ldots,i_n)$. Здесь параметр n характеризует "новую" длину набора ($n \geq 0$). Если $\beta(K) = 0$ и $\gamma(K) = 2$, то параметр n указывает новое число экземпляров повторяющейся группы. В случае $\beta(K) = 2$ при

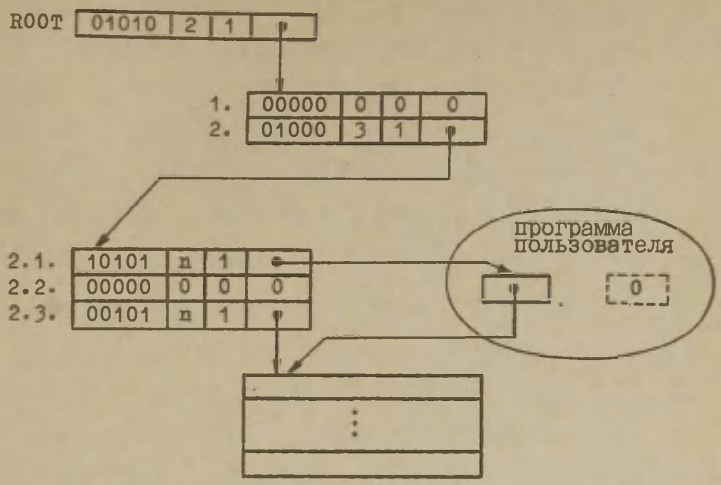

ROOT
01010
2
1
1.
00000
0
0
0
2.
01000
3
1
2.1.
10101
n
1
2.2.
00000
0
0
0
2.3.
00101
n
1
программа
пользователя
0

Рис. 3.

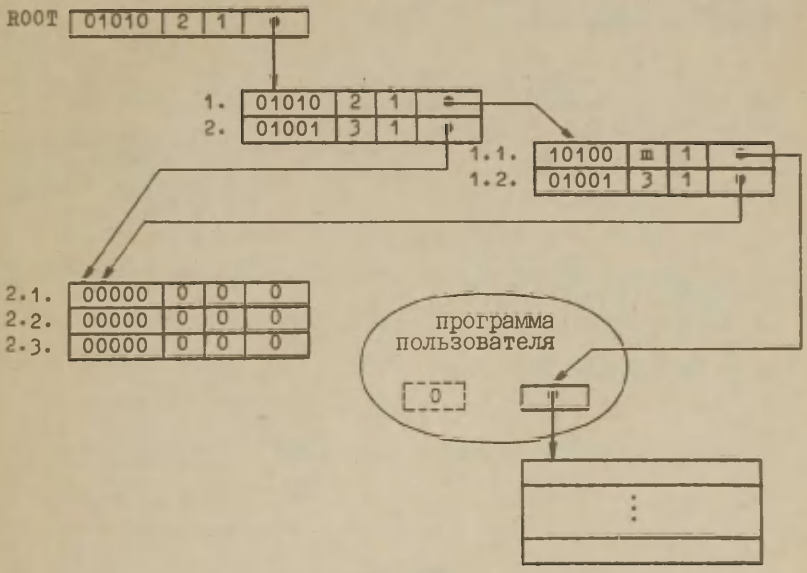

ROOT
01010
2
1
1.
01010
2
1
2.
01001
3
1
1.1.
10100
m
1
1.2.
01001
3
1
2.1.
00000
0
0
0
2.2.
00000
0
0
0
2.3.
00000
0
0
0
программа
пользователя
0

Рис. 4.

t = 0 параметр n указывает новое число кодослов, а при t = 1 – новую длину атома, причем только здесь разрешается и случай n = 0 (последнее означает, что изменяется тип кодослова K).

Во всех остальных случаях параметр n задает новую длину набора K в двойных словах (при $\delta(K) = 0$) или в блоках набора (при $\delta(K) = 1$). Если требуемая длина больше старой длины набора, возникнет ситуация ошибки. В случае, когда удаляемая часть набора содержит кодослова, к этим кодословам применяется процедура DELETE.

Для примера рассмотрим структуру кодослов, приведенную на рис. 4. Результатом выполнения процедур LONG((1),1,0,0) и SHORT((2),1,0,0) теперь будет структура, представленная на рис. 5.

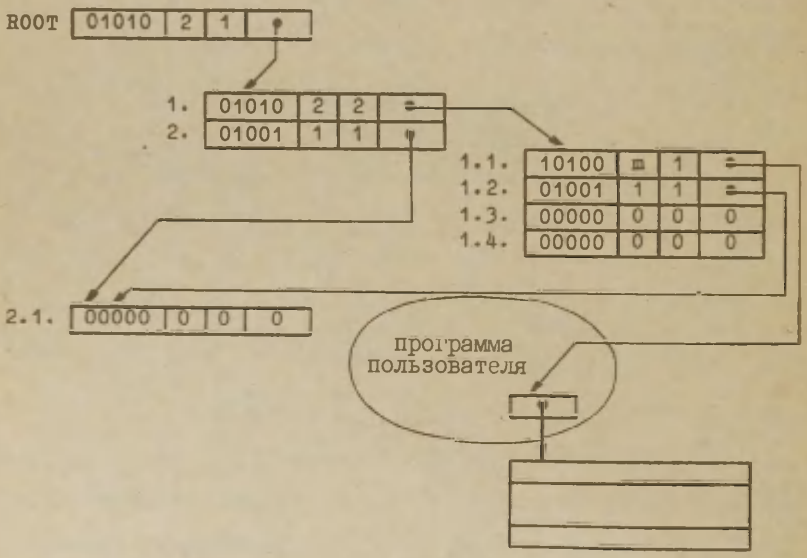


Рис. 5.

Отметим еще, что система DYN все время ведет счет за количеством копированных кодослов в структуре всех кодослов. В случае отсутствия копированных кодослов почти во всех процедурах применяются другие алгоритмы, которые значительно проще и эффективнее вышеописанных.

Следует еще отметить, что пользователю предоставляется возможность создания собственных процедур для отображения и модификации структур данных, используя существующие внешние и внутренние процедуры. Более того, основную часть системы можно использовать и при изменении структуры самих кодослов.

Литература

1. Аилиф Дж., Принципы построения базовой машины. Москва, "Мир", 1973.
2. Аллик К.К.-Э., Микли Т.Й., Отображение структур данных в памяти ЭВМ. Труды Таллинского политехн. ин-та, 1974, № 366, 23-30.
3. Аллик К.К.-Э., Микли Т.Й., Тедерсоо К.Й., Схема немагазинного распределения оперативной памяти ЭВМ "МИНСК-32". Труды Таллинского политехн. ин-та, 1974, № 366, 31-35.
4. Каазик Ю., Томбак М., Система РАМА для управления базой данных. Труды ВЦ ТГУ, 1978, № 41, 3-6.
5. Кнут Д., Искусство программирования для ЭВМ. Т.1, Москва, "Мир", 1976.

ВЫДЕЛЕНИЕ НАИБОЛЕЕ СУЩЕСТВЕННЫХ КЛАССОВ ДАННЫХ

К. Ээремаа

1. База данных, как правило, предназначена для хранения большого количества сведений о некоторых объектах и для представления связей между этими объектами. В некоторых случаях исследование всего набора объектов можно заменить исследованием его некоторого, наиболее существенного подмножества. В настоящей статье и рассматривается вопрос о выделении таких наиболее существенных подмножеств объектов. Для этого конструируется система, элементами которой являются рассматриваемые объекты, внутренние связи между элементами выбраны в соответствии с заданными отношениями между объектами, а значимость (вес) каждого элемента оценивается некоторым числом. Если в такой системе выполняется принцип монотонности в определенном ниже смысле, то по теории выделения экстремальных подсистем монотонной системы [2,3] можно найти наибольшее ядро системы, задающее искомое подмножество.

Описываемый ниже метод может быть рассмотрен частным случаем классифицирования объектов, в котором наряду с разбиением на классы искомым является и максимально возможный уровень. В конце статьи показывается, что ядро системы, при соответствующем задании весов элементов можно рассматривать

как k-кластер в смысле Р.Ф. Линга [1], где k максимальное число, при котором образуются кластеры.

Последующее изложение, в основном, построено на примерах. Анализируемый пример выбран из области информационного поиска - рассматриваются способы задания связей между документами и индексами и выделение на их основе наиболее существенных классов. Изложение метода на конкретных примерах не ограничивает общности - метод применим для анализа разных структур данных.

2. Прежде всего коротко приводим определения монотонной системы и ядра монотонной системы, а также описываем алгоритм вычисления наибольшего ядра. Более подробное изложение этих вопросов можно найти в работах [2, 3, 4, 5].

Пусть задано некоторое конечное множество элементов $W = \{x_1, x_2, \ldots, x_n\}$, на котором определена весовая функция $g \equiv g_W$. Системой S называется пара $S = (W, g)$, а значение $g_W(x)$ называется весом элемента $x \in W$ в системе S. Допустим, что для любого подмножества $W' \subseteq W$ определено сужение $g_{W'}$ весовой функции g. Тогда система $S' = (W', g_{W'})$ считается подсистемой системы S. Так как весовая функция фиксирована для рассматриваемой системы S, то любая подсистема S' определена множеством ее элементов W'.

Определение 1. Система $S = (W, g_W)$ называется монотонной, если для любых двух ее подсистем $S_1 = (W_1, g_{W_1})$ и $S_2 = (W_2, g_{W_2})$ при $x \in W_2$ и $W_2 \subset W_1$ имеет место $g_{W_2}(x) \geq g_{W_1}(x) \geq g_W(x)$ или же $g_{W_2}(x) \leq g_{W_1}(x) \leq g_W(x)$. В первом случае обозначаем систему через S^+, во втором через S^-.

Легко доказывается, что подсистема монотонной системы является монотонной в том же направлении. Среди всевозможных подсистем монотонной системы особый интерес представляют такие, на которых весовая функция g принимает экстремальные значения в определенном ниже смысле. Такие подсистемы называются ядрами системы. Ядро системы можно считать ее самой существенной частью.

Определение 2. Ядром системы $S^+ = (W, g_W)$ называется ее подсистема H^+, при которой функция $F(H) = \max\limits_{x \in W'} g_{W'}(x)$, определенная на подсистемах $H = (W', g_{W'})$, достигает минимум. Аналогично, ядром системы $S^- = (W, g_W)$ называется ее подсистема H^-, при которой функция $F(H) = \min\limits_{x \in W'} g_{W'}(x)$ достигает максимум.

В настоящей статье в дальнейшем рассматриваются только системы $S^- \equiv S = (W, g)$ и соответственно ядра $H^- \equiv H = (W', g)$. В случаях, когда особо подчеркивается минимальный вес элементов ядра $s = \min\limits_{x \in W'} g_{W'}(x)$, ядро обозначается через H^s.

Ядро системы не обязательно определено однозначно: функция F может достигать максимум на нескольких подсистемах. В статье [2] доказывается, что если $H_1 = (W_1, g_{W_1})$ и $H_2 = (W_2, g_{W_2})$ ядра данной системы, то ядром оказывается и подсистема $H = (W', g_{W'})$, где $W' = W_1 \cup W_2$. Объединение всех ядер системы называется наибольшим ядром.

Алгоритм вычисления наибольшего ядра [5] заключается в нахождении такого численного значения $u \in [L, M]$, где

$$L = \min_{x \in W} g_W(x), \qquad M = \max_{x \in W} g_W(x),$$

при котором специальная процедура СЛОЙ выделяет наибольшее

ядро. Работа процедуры СЛОЙ(u,W') при $W' \subseteq W$ заключается в последовательном применении вспомогательной процедуры слой(u,W^i) и описывается следующим образом:

$$\text{СЛОЙ}(u,W') = \text{слой}(u,W^n),$$

где

$$W^i = \text{слой}(u,W^{i-1}) = \left\{ x : x \in W^{i-1},\ g_{W^{i-1}}(x) > u \right\},$$

$i=1,\ldots,n$, $W^0=W'$ и значение n определяется условием $W^n=W^{n+1}$.

Алгоритм вычисления наибольшего ядра описывается теперь следующими шагами:

1. $L := \min\limits_{x \in W} g_W(x)$, $M := \max\limits_{x \in W} g_W(x)$;
2. $u := \varphi(L,M)$, где φ некоторая фиксированная функция вычисления значения $u \in [L,M]$;
3. $W' :=$ СЛОЙ(u,W); если $W' = \emptyset$, то положить $M := u$ и вернуться к шагу 2;
4. $u := \min\limits_{x \in W'} g_{W'}(x)$;
5. $W'' :=$ СЛОЙ(u,W'); если $W'' \neq \emptyset$, то положить $L := u$ и вернуться к шагу 2, в противном случае наибольшее ядро $H^u = (W',g)$ найдено.

Пример вычисления наибольшего ядра приводится ниже.

В настоящей статье выделение "подъядер" наибольшего ядра рассматриваются только в частном случае. Имеет место теорема.

Теорема 1. Если в наибольшем ядре $H^S = (W',g)$ системы $S = (W,g)$ существует подсистема $H_1 = (W_1,g)$, где $W_1 \subset W'$ такая, что для любого элемента $x \in W_1$ имеет место $g_{W_1}(x)=g_{W'}(x)$, то H_1 является ядром системы S.

Для доказательства теоремы следует показать, что H_1 является одной из тех подсистем, на которых $s_1 = \min\limits_{x \in W_1} g_{W_1}(x)$

достигает максимальное значение s. Понятно, что s_1 не может быть больше s, так как в таком случае $H^{=}$ не являлось бы ядром. Покажем, что s_1 не может быть меньше s. Действительно, если $s_1 < s$ и значение s_1 достигается при элементе $\bar{x}$, то по предпосылкам теоремы $s_1 = g_{W_1}(\bar{x}) = g_{W'}(\bar{x}) < s$, т.е. s уже не является минимальным весом. Значит $s = s_1$ и подсистема H_1 является ядром.

3. Пусть заданы множество документов (объектов) $\mathfrak{D} = \{d_1, d_2, \ldots, d_n\}$ и множество индексов (признаков) $\mathfrak{X} = \{x_1, x_2, \ldots, x_m\}$. Каждый документ $d \in \mathfrak{D}$ описывается размытым множеством D в пространстве индексов [6]:

$$D = \{x_1 \mid f(d,x_1), x_2 \mid f(d,x_2), \ldots, x_m \mid f(d,x_m)\},$$

где заданная функция принадлежности f принимает значения на отрезке $[0,1]$.

Аналогично можно каждому индексу $x \in \mathfrak{X}$ сопоставить размытое множество

$$X = \{d_1 \mid f(d_1,x), d_2 \mid f(d_2,x), \ldots, d_n \mid f(d_n,x)\},$$

а тем самым совокупность документов и индексов описывается матрицей $\{D_i\} \times \{X_j\}$, где $i=1,\ldots,n$; $j=1,\ldots,m$. В общем случае такой подход соответствует описанию множества объектов некоторыми признаками, при котором роль объектов и признаков заменима друг на друга.

Одной из наиболее важных задач в области информационного поиска является разбиение множества документов на некоторые тематические классы [7]. Дело проще, когда имеются исходные

соображения, по которым можно определить число классов и тематику каждого из них. В общем же случае требуется найти классификацию документов по заранее неизвестным темам: тема класса определяется в ходе классификации. Последний случай анализируется и в настоящей статье.

Поставим сначала более узкую задачу: найти совокупность наиболее четко выраженных тематических классов. Допустим, что тема документа определяется совокупностью индексов этого документа, т.е. описание документа на множестве индексов является тематическим описанием. Тогда общность тематики двух документов $d_i, d_j \in \mathfrak{D}$ характеризуется связью между этими документами по индексам и можно вычислить по формуле

$$R(d_i, d_k) = |D_i \cap D_k|, \qquad (1)$$

где $D_i \cap D_k$ – пересечение размытых множеств, т.е.

$$D_i \cap D_k = \{x_1 | \min(f(d_i, x_1), f(d_k, x_1)), \ldots, x_m | \min(f(d_i, x_m), f(d_k, x_m))\},$$

а $|D| = \sum_{j=1}^{m} f(d, x_j)$ является мощностью размытого множества D. В частном случае, когда документы описываются неразмытыми множествами, формула (1) дает количество общих индексов в описании документов.

Каждому документу $d_i \in \mathfrak{D}$ поставим в соответствие число

$$g_{\mathfrak{D}}(d_i) = \sum_{j \neq i} R(d_i, d_j), \qquad (2)$$

зависящее от $d_i, \mathfrak{D}$ и характеризующее тематическую связанность документа d_i со всеми другими документами d_j.

Этим, по сути дела, конструирована система $S = (\mathfrak{D}, g_{\mathfrak{D}})$. Легко проверить, что эта система является монотонной. Имея в

виду содержание весовой функции g, наибольшее ядро системы и является искомым множеством наиболее связанных документов.

Рассмотрим пример.

Пусть совокупность документов $\mathfrak{D} = \{d_1, d_2, \ldots, d_8\}$ описывается индексами $\mathfrak{X} = \{x_1, x_2, \ldots, x_8\}$ так, как задано в таблице 1 (в таблице указываются значения функции принадлежности f). Тогда по формуле (1) можно установить связи между от-

$\mathfrak{D}$ \ $\mathfrak{X}$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
d_1	0,8	0,9						
d_2	0,4		1,0					
d_3			0,2	1,0				
d_4			0,8	0,2				
d_5		0,3			0,5	1,0		
d_6					0,3	0,2	1,0	
d_7					0,5			1,0
d_8							0,2	0,2

Таблица 1. Матрица "документ-индекс".

дельными документами (см. рис. 1), а по (2) найти вес каждого документа. Вычисляя наибольшее ядро для полученной системы (см. табл. 2) получаем в качестве ядра множество $H = \{d_2, d_4, d_5, d_6, d_7\}$. Связи между документами наибольшего ядра приведены на рис. 2. Судя по рисунку, наибольшее ядро распадает на две несвязанные между собой части: $W_1 = \{d_2, d_4\}$ и $W_2 = \{d_5, d_6, d_7\}$. Легко убедиться, что выполнены предпосылки

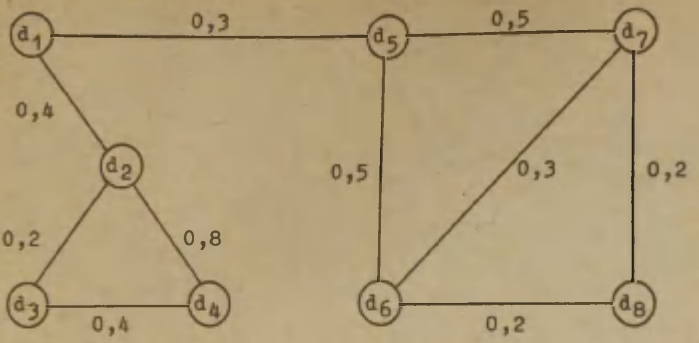


Рис. 1. Граф связей между документами.

теоремы 1: $H_1 = (W_1, g)$ и $H_2 = (W_2, g)$ являются ядрами. Минимальным весом элемента ядра является 0,8 .

u \ g	$g(d_1)$	$g(d_2)$	$g(d_3)$	$g(d_4)$	$g(d_5)$	$g(d_6)$	$g(d_7)$	$g(d_8)$	Примечания
	0,7	1,4	0,6	1,2	1,3	1,0	1,0	0,4	L=0,4;M=1,4;u:=L+M/2
0,9	–	0,8	–	0,8	1,0	0,8	0,8	–	СЛОЙ(0,9;$\mathfrak{D}$)
		–		–	0	–	–		$\mathfrak{D}'=\emptyset$; M:=0,9
0,65	0,7	1,2	–	0,8	1,3	0,8	0,8	–	$\mathfrak{D}'\neq\emptyset$; u:=min $g_{\mathfrak{D}'}(d_1)$
0,7	–	0,8		0,8	1,0	0,8	0,8		СЛОЙ(0,7;$\mathfrak{D}$)$\neq\emptyset$;L:=0,65
0,77	–	0,8	–	0,8	1,0	0,8	0,8	–	СЛОЙ(0,77;$\mathfrak{D}$)$\neq\emptyset$;u:=min
0,8		–		–	0	–	–		$\mathfrak{D}'$ – ядро

Таблица 2. Ход вычисления наибольшего ядра.

По сделанным предпосылкам каждое ядро можно считать тематическим классом документов, характеризуемым применяемыми там индексами. В данном случае ядро $\{d_2, d_4\}$ описывается ин-

дексами x_1, x_3 и x_4, а ядро $\{d_5, d_6, d_7\}$ индексами x_2, x_5, x_6, x_7 и x_8.

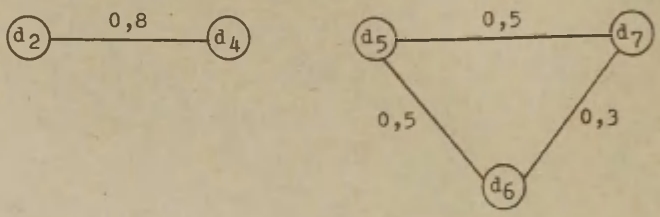


Рис. 2. Наибольшее ядро документов.

Понятно, что среди индексов, описывающих некоторый тематический класс, могут оказаться "более важные" и "менее важные" индексы. Для получения оценки важности индексов, рассмотрим их совместную входимость в документы. В этом случае получаются аналогичные формулам (1) и (2) формулы

$$R(x_k, x_j) = |X_k \cap X_j| \qquad (3)$$

и

$$g_{\mathfrak{X}}(x_j) = \sum_{k \neq j} R(x_k, x_j). \qquad (4)$$

По формулам (3) и (4) можно оценить значимость индексов ядра документов. Однако, простое "отбрасывание" менее важных индексов с целью нахождения основной тематики ядра, повлияет на структуру ядра и рассматриваемая совокупность документов может уже не оказаться ядром в смысле определения 2.

Во избежание такой ситуации построим систему $S = (W, G)$, элементами которой являются как документы так и индексы ($W = \mathfrak{D} \cup \mathfrak{X}$), причем вес элемента определяется либо формулой

(2) либо (4):

$$G_W(y) = \begin{cases} g_{\mathcal{D}}(y), & \text{если } y \in \mathcal{D}, \\ g_{\mathcal{X}}(y), & \text{если } y \in \mathcal{X}. \end{cases}$$

Понятно, что система S = (W,G) распадает на две зависящие друг от друга подсистемы. В случае рассматриваемого примера к структуре документов, изображенной на рис. 1, прибавляется еще структура индексов (см. рис. 3). Используя данные таблицы 1 видим, что наибольшим ядром конструированной системы S = (W,G) оказывается множество $\{d_5, d_6, d_7, x_5, x_6\}$.

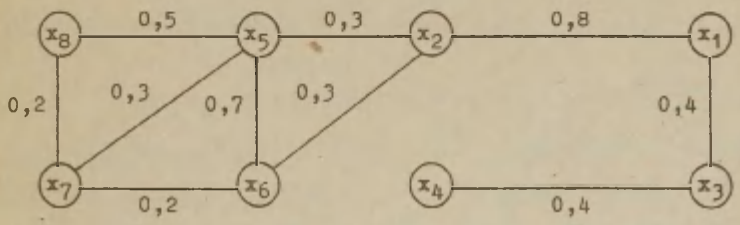


Рис. 3. Граф связей между индексами таблицы 1.

Полученное ядро (с минимальным весом 0,7) можно рассматривать как состоящее из двух частей (см. рис. 4): множества документов $\{d_5, d_6, d_7\}$ и множества характерных для этих документов индексов $\{x_5, x_6\}$. Подчеркиваем, что в данном случае ни совокупность документов $\{d_5, d_6, d_7\}$, ни совокупность индексов $\{x_5, x_6\}$, отдельно взятыми, не являются ядрами. Ядро системы S = (W,G) следует трактовать наиболее четко выраженным тематическим классом $\{d_5, d_6, d_7\}$ с основной тематикой $\{x_5, x_6\}$.

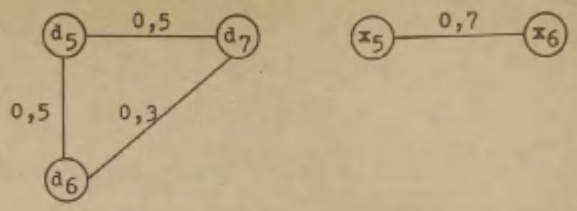


Рис. 4. Наибольшее ядро системы $S = (W,G)$.

Полученные при анализе системы $S = (W,G)$ классы основной тематики можно в информационно-поисковых системах рассматривать как ассоциативные классы индексов. Очевидно, аналогичные классы индексов можно получить и путем анализа системы $S = (X,g_X)$.

4. Пусть для системы $S = (W,g)$ найдено наибольшее ядро $H^{s_1} = (W^1,g)$. Тогда процесс вычисления ядра можно повторить для подсистемы $S' = (W \setminus W^1,g)$, так как по определению весовая функция g определена на любом подмножестве и подсистема монотонной системы является монотонной. Минимальный вес s_2 элементов наибольшего ядра H^{s_2} подсистемы S' будет, конечно, меньше чем s_1.

В результате последовательного применения такого процесса выделения наибольшего ядра получается последовательность ядер $H^{s_1},H^{s_2},\ldots,H^{s_m}$, которой соответствуют последовательности множеств $W^1,W^2,\ldots,W^m$ и весов $s_1 > s_2 > \ldots > s_m$. Если зафиксировано некоторое значение s, считаемое уровнем классификации, то процесс естественно оборвать на таком шаге k, для которого $s_k \geq s > s_{k+1}$. В этом случае множество $W^1 \cup W^2 \cup \ldots$

$\ldots \cup W^k$ является множеством минимально "k-существенных" элементов, на котором задано разбиение по "степеням существенности".

Обращаясь снова к рассмотренному выше примеру, найдем для системы $S = (\mathcal{D} \cup \mathcal{X}, G)$ второе по важности ядро. Этим оказывается множество элементов $\{d_1, d_2, d_3, d_4, x_1, x_2, x_3, x_4\}$ с минимальным весом 0,4. Удовлетворяясь значением уровня 0,4, из множества $\{\mathcal{D} \cup \mathcal{X}\}$ выделялось множество $\{d_1, d_2, \ldots, d_7, x_1, x_2, \ldots, x_6\}$ с внутренним расслоением на уровне 0,7. Оставшееся множество $\{d_8, x_7, x_8\}$ можно рассматривать как множество наименее существенных элементов.

5. В приведенном выше изложении мы исходили из конкретного примера, чтобы демонстрировать возможности применения теории монотонных систем для классифицирования объектов. Данную методику, однако, можно обобщить на любые исследуемые объекты, для которых удается построить монотонную систему. Ниже в общих чертах рассматривается возможность такого построения на основе матрицы различия и показывается отождествимость ядра с кластерами в смысле Р.Ф. Линга [1].

Пусть заданы конечное множество объектов $W = \{x_1, x_2, \ldots, x_n\}$ и матрица различий $R = (r(x_i, x_j)) = (r_{ij})$, где $i, j = 1, \ldots, n$. Сопоставим каждому элементу $x_i \in W$ его вес $g_W(x_i) = F(x_i, R)$ как функцию от матрицы R. Если при этом F определена таким образом, что для любых $W' \subset W'' \subseteq W$ с матрицами различий R' и R'' соответственно при всех $x \in W'$ имеет место

$$g_{W'}(x) = F(x, R') \leq F(x, R'') = g_{W''}(x),$$

то пара $S = (W,F)$ определяет монотонную систему. Содержательное значение наибольшего ядра системы S определяется семантикой функции F. Например, если положить

$$g_W^1(x_i) = \sum_{j=1}^{n} r_{ij} ,$$

то элементами ядра системы являются те объекты, при которых минимальное суммарное различие от других объектов достигает свое максимальное значение. Таким образом - ядром является множество наиболее удаленных объектов.

С другой стороны, если при фиксированном r поставить

$$g_W^2(x_i) = \sum_{j=1}^{n} r_{ij}^* , \qquad (5)$$

где

$$r_{ij}^* = \begin{cases} 1, \text{ если при } i \neq j \quad r_{ij} \leq r, \\ 0, \text{ если } r_{ij} > r, \quad \text{или} \quad i = j, \end{cases} \qquad (6)$$

то вес объекта x в системе $S = (W,g^2)$ означает число "подобных ему", различия с которыми не превышают заданного предела. В этом случае некоторое ядро $H_i^k = (W_i,g^2)$ дает подмножество объектов W_i, имеющие в W_i по меньшей мере k подобных. При этом по определению весовой функции k является целым числом и по определению ядра принимает на W_i максимальное значение.

По своей внутренней структуре ядро системы $S = (W,g^2)$ напоминает k-кластер, построенный на фиксированном уровне различия r с максимально возможным числом связей k. Для того, чтобы показать, что ядро действительно является k-клас-

тером, напомним сначала определение понятия кластера.

Пусть заданы множество объектов $W = \{x_1, x_2, \ldots, x_n\}$ и матрица различий $R = (r(x_i, x_j))$, где $i, j = 1, \ldots, n$. Тогда подмножество $W' \subseteq W$ называется k-кластером при заданном значении r, если:

1^0 для любых $x, y \in W'$ существует цепь $x = x_1, x_2, \ldots, x_m = y$ такая, что $r(x_i, x_{i+1}) \leq r$, где $i = 1, \ldots, m-1$;

2^0 для любого $x \in W'$ найдется по меньшей мере k-элементное подмножество $W^x \subset W'$ $(x \notin W^x)$ такое, что $r(x,y) \leq r$ при $y \in W^x$;

3^0 подмножество W' является максимальным в том смысле, что не найдется множества $W'' \supset W'$ такого, что условия 1^0 и 2^0 выполняются на W''.

Приведенное определение легко переформулировать для матрицы $R^* = (r^*_{ij})$, полученной из матрицы различий R с помощью формулы (6). Действительно, в первом условии следует лишь писать равносильное неравенству $r(x_i, x_{i+1}) \leq r$ равенство $r^*(x_i, x_{i+1}) = 1$, а во втором $r^*(x,y) = 1$ вместо $r(x,y) \leq r$. Таким образом, за основу выделения кластеров и наибольшего ядра принимаются одни и те же исходные данные.

Понятно, что в общем случае наибольшее ядро не является k-кластером, так как ничем не гарантируется выполненность условия 1^0. Допустим сначала, что в частном случае наибольшее ядро $H^k = (W', g^2)$ является связным множеством в смысле условия 1^0 определения кластера. Тогда по определению ядра для любого элемента $x \in W'$ имеет место

$$g^2_{W'}(x) = \sum_{y \in W'} r^*(x,y) \geq k, \qquad (7)$$

т.е. найдется по меньшей мере k-элементное множество W^x такое, что $r^*(x,y) = 1$ при $y \in W^x$. Тем самым условие 2^o в определении кластера выполнено. Выполненность условия 3^o обеспечивается тем, что W' является наибольшим множеством, для которого выполняется неравенство (7).

Значит, если множество W' является связным множеством, то оно является k-кластером. Кроме того, по определению ядра не существует подсистемы $H^{k'} = (W'', g^2)$ такой, что $k' > k$ и этим связное наибольшее ядро является кластером при максимально возможной связанности элементов.

Пусть теперь для наибольшего ядра $H^k = (W', g^2)$ условие 1^o не выполняется. Разобьем множество элементов W' на подмножества $W' = W'_1 \cup \ldots \cup W'_m$ таким образом, что $x, y \in W'_i$, если между ними существует определенная условием 1^o цепь. Понятно, что при таком разбиении $W'_i \cap W'_j = \emptyset$, если $i \neq j$. Оказывается, что в этом случае определенная множеством W'_i система $S_i = (W'_i, g^2)$ является ядром системы $S = (W, g^2)$. Действительно, так как для любого $x \in W'_i$,

$$g^2_{W'_i}(x) = \sum_{i=1}^{m} \Big(\sum_{y \in W'_i} r^*(x,y) \Big) = \sum_{y \in W'_i} r^*(x,y),$$

то на множествах W'_i $(i=1,\ldots,m)$ выполнены предпосылки теоремы 1, т.е. $H^k_i = (W'_i, g^2)$ является ядром, а учитывая конструкцию множества W'_i – связным ядром. Для связного ядра, как было показано выше, выполняется условие 2^o, выполненность же условия 3^o обеспечивается тем, что для любого элемента $x \in W'_i$ не найдется $y \in W' \setminus W'_i$ такого, что $r^*(x,y) = 1$.

Таким образом, разбиение наибольшего ядра $H^k = (W', g^2)$

системы $S = (W, g^2)$, где функция g^2 определена формулой (5), соответствует выделению k-кластеров множества W при заданных r и R. При этом k является максимальным числом подобных элементов, при котором образуются кластеры.

Следует отметить, что для выделения кластеров на более низком уровне $k' < k$ нельзя использовать систему $S' = (W \setminus W', g^2)$, как это делалось выше, так как в системе S' не учитывается подобие элементов из $W \setminus W'$ с элементами из W'. Поэтому, метод выделения ядер нельзя рассматривать как обобщение метода кластеризации. Оба эти метода имеют свою специфику и свою область применения, хотя имеют места стыковки.

Литература

1. Ling, R.F., On the theory and construction of k-clusters. The Computer Journal, 1972, vol. 15, № 4, 326-332.
2. Муллат И.Э., Экстремальные подсистемы монотонных систем. I. Автоматика и телемеханика, 1976, № 5, 130-139.
3. Муллат И.Э., Экстремальные подсистемы монотонных систем. II. Автоматика и телемеханика, 1976, № 8, 169-178.
4. Муллат И.Э., Экстремальные подсистемы монотонных систем. III. Автоматика и телемеханика, 1977, № 1, 109-119.
5. Выханду Л.К., Монотонные методы анализа данных. Труды ТПИ, 1979, № 468, 15-26.
6. Ээремаа К.А., Модель ИПС с тезаурусом на базе теории размытых множеств. Уч. зап. ТГУ. Труды по искусственному интеллекту II. Семантика и представление знаний. Тарту, 1979, 145-155.
7. Солтон Д.Ж., Динамические библиотечно-информационные системы. М., 1979.

ОРТОГОНАЛЬНОЕ РАЗМЕЩЕНИЕ ДРЕВОВИДНЫХ СТРУКТУР

Ю. Кихо

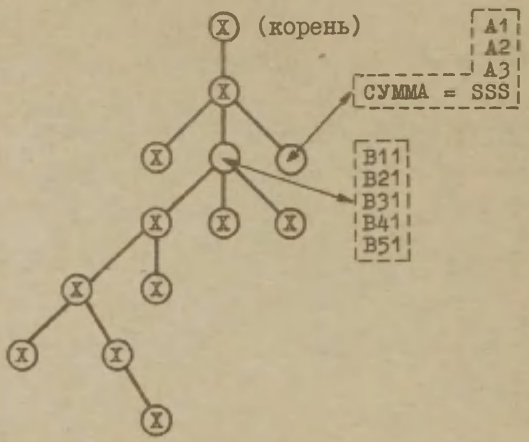
При создании системы управления базами данных возникает проблема визуального представления обрабатываемых структур, в частности деревьев. Скорость отладки системы существенно зависит от наличия удобных средств для наглядной распечатки информации о состоянии структур. В настоящей работе поставлена и решена частичная задача о планировании тренарных деревьев (3-деревьев).

1. Постановка задачи

Под 3-деревом подразумевается ориентированное от корня дерево, каждая вершина которого имеет не более трех преемников (поддеревьев). Предполагается, что с каждой вершиной дерева связана ее картина, т.е. рассматриваются не точечные вершины, а вершины, имеющие реальные размеры (см. рис. 1 сверху, где односимвольные картины изображены непосредственно в вершинах).

Для того, чтобы данное 3-дерево изобразить на плоскости, необходимо расставить на плоскости картины всех вершин (не меняя исходной ориентации картин) и провести линии от карти-

ны каждой вершины к ее преемникам. Размещение 3-дерева на плоскости называется ортогональным, если дуги между картинами вершин изображены в виде непересекающихся отрезков прямых, расположенных либо вертикально, либо горизонтально (см. рис. 1 внизу).



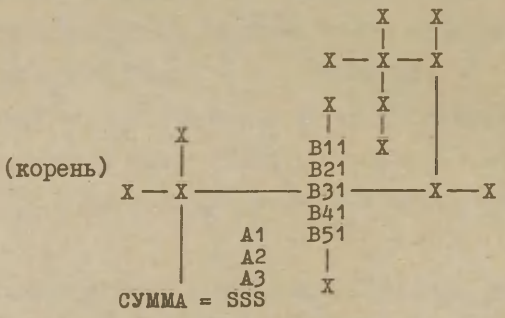


Рис. 1. Пример 3-дерева и его ортогональное размещение.

Основной вопрос при размещении дерева – это вопрос о длине дуг, т.е. о расстояниях между вершиной и ее преемниками. Если заранее правильно определены длины всех дуг, то размещение дерева не представляет особых трудностей. В настоящей работе описывается алгоритм определения длин дуг, необходимых для ортогонального размещения 3-дерева на плоскости. Предполагается, что преемники каждой вершины упорядочены (пронумерованы). Планируется размещение, в котором исходящие из вершины дуги ориентированы по отношению к направлению входящей в эту вершину дуги следующим образом: первая дуга – прямо (т.е. в том же направлении, что и входящая дуга), вторая дуга – влево, третья – вправо. Направление первой дуги, исходящей из корня, можно выбрать произвольным образом; этим выбором полностью определяются ориентации всех остальных дуг.

Предлагаемым алгоритмом определяется лишь одно из возможных ортогональных размещений данного 3-дерева на плоскости. Имеется возможность в некоторой степени управлять размещением, варьируя упорядоченность преемников вершин.

2. Понятия и обозначения

Так как при планировании дерева конкретное содержание картины не имеет значения, то картину можно рассматривать как непустое конечное множество точек плоскости, имеющих целочисленные координаты в некоторой сети декартовых координат. Более того, важны только "крайние" точки картины (т.н. очертание, определяемое ниже) и ее центр. Центр картины –

это особо выделенная точка картины. Планирование дерева происходит с таким расчетом, чтобы продолжения отрезка, изображающего дугу, проходили через центры инцидентных данной дуге вершин.

Естественно предположить, что исходные картины заданы в общей сети координат. Относительно к этой сети подразумеваются и направления дуг (например, "вправо" означает "в направлении оси абсцисс общей сети"). Но поскольку взаимное расположение картин не определено, а является искомым, то в процессе планирования целесообразнее рассматривать каждую картину в собственной, локальной сети координат. Локальная сеть координат картины некоторой вершины T имеет начало в центре картины и ориентирована относительно общей сети так, что направление оси ординат совпадает с направлением входящей в вершину T дуги (см. рис. 2). Определяемые ниже понятия подразумеваются именно в смысле локальных координат.

В случае произвольного непустого множества S точек на плоскости обозначим:

$\mathrm{xmin}(S)$ = минимальная абсцисса точек S,

$\mathrm{ymin}(S)$ = минимальная ордината точек S,

$\mathrm{xmax}(S)$ = максимальная абсцисса точек S,

$\mathrm{ymax}(S)$ = максимальная ордината точек S,

$S_{x=i} = \{(x,y) : (x,y) \in S \ \& \ x=i\}$,

$S_{y=j} = \{(x,y) : (x,y) \in S \ \& \ y=j\}$.

Рассмотрим вершину T некоторого 3-дерева и ее картину K. Обозначим

$$I_T = \{i : \exists j (i,j) \in K\}, \quad J_T = \{j : \exists i (i,j) \in K\};$$

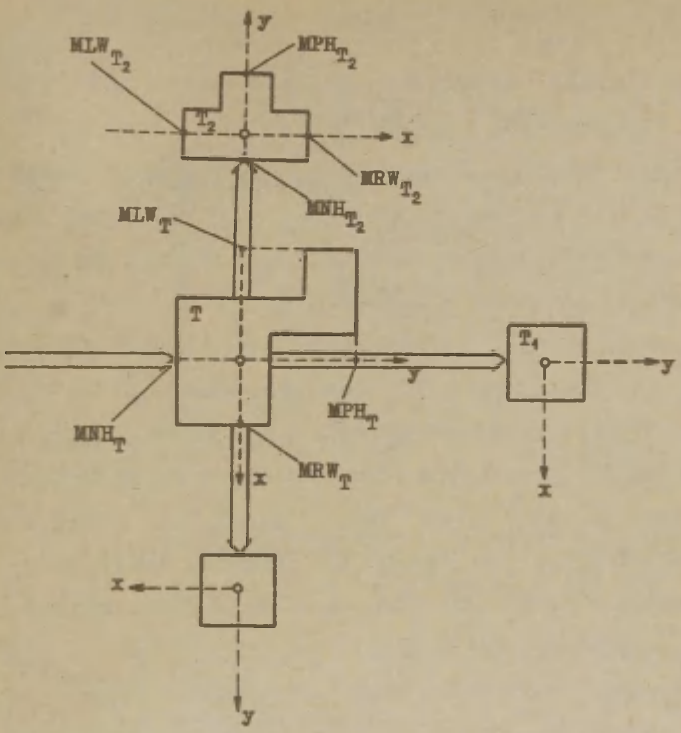


Рис. 2. Локальные сети координат (для вершин T и T_2 указаны экстремальные координаты).

для $i \in I_T$:

$$NH_T(i) = ymin(K_{x=i}), \quad PH_T(i) = ymax(K_{x=i});$$

для $j \in J_T$:

$$LW_T(j) = xmin(K_{y=j}), \quad RW_T(j) = xmax(K_{y=j});$$

$$MNH_T = \min_{i \in I_T} NH_T(i), \quad MPH_T = \max_{i \in I_T} PH_T(i),$$

$$MLW_T = \min_{j \in J_T} LW_T(j), \qquad MRW_T = \max_{j \in J_T} RW_T(j),$$

$$\mathcal{C}_T = \{(i,j) : (i,j) \in K \;\&\; (i = LW_T(j) \vee i = RW_T(j) \vee$$
$$\vee\, j = NH_T(i) \vee j = PH_T(i))\}.$$

Множество $\mathcal{C}_T$ называется очертанием картины вершины T. Очевидно, что $i \in I_T \Rightarrow MLW_T \leq i \leq MRW_T$ и $j \in J_T \Rightarrow MNH \leq j \leq MPH$. Если имеет место $MLW_T \leq i \leq MRW_T \Rightarrow i \in I_T$ и $MNH \leq j \leq MPH \Rightarrow j \in J_T$, то картина называется непрозрачной.

Предполагается, что рассматриваемые картины вершин являются непрозрачными. Это позволяет задавать множества I_T и J_T при помощи их граничных пар (MLW_T, MRW_T) и (MNH_T, MPH_T) соответственно. В ходе работы алгоритма очертание представляется не явным образом (в виде списка точек $\mathcal{C}_T$), а в виде таблиц значений NH(i), PH(i) для $i \in I_T$ и LW(j), RW(j) для $j \in J_T$ (см. таблицу 1, где приведены значения, соответствующие очертанию картины, изображенной на рис. 3).

Деревом T называется поддерево данного дерева, корнем которого является вершина T. Под картиной (спланированного) дерева T подразумевается картина в локальной для корня T сети координат, которая состоит из точек картины корня, картин ортогонально размещенных поддеревьев T и отрезков между корнем T и корнями поддеревьев (если T не имеет поддеревьев, то картина дерева T совпадает с картиной вершины T).

I	-2	-1	0	1	2	3	4
NH	-2	-2	-2	-2	-2	1	4
PH	3	3	3	5	5	5	5

J	-2	-1	0	1	2	3	4	5
LW	-2	-2	-2	-2	-2	-2	1	1
RW	2	2	2	3	3	2	4	4

Табл. 1. Значения NH(i), PH(i), LW(j), RW(j) для картины, изображенной на рис. 3.

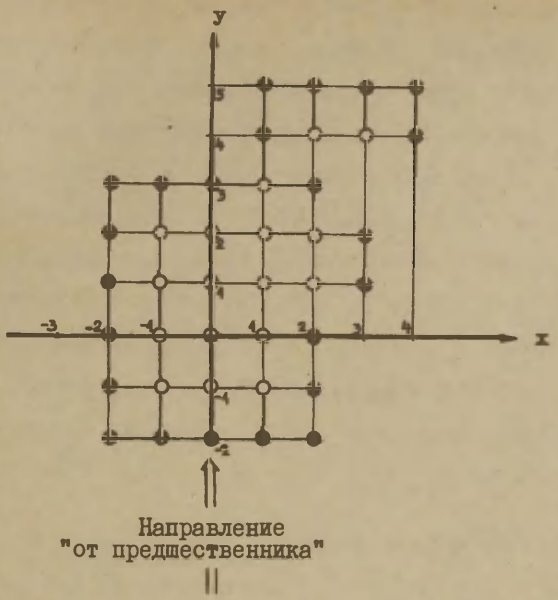


Рис. 3. Картина вершины (точки картины обозначены кружками, причем темные кружки соответствуют точкам очертания).

3. Алгоритм планирования 3-дерева

Идея алгоритма заключается в последовательном вычислении очертаний поддеревьев (важно заметить, что в алгоритме планирования картины поддеревьев вообще не собираются). Пусть вершина T имеет поддеревья T_1, T_2, T_3. После того, когда вычислены очертания для T_1, T_2, T_3, определяется длина a_3 отрезка $T-T_3$, исходя из очертаний картины вершины T и картины поддерева T_3 (см. рис. 4, где кружком обозначен центр карти-

ны, жирной линией - точки очертания, а через (a) (b) (c) показаны состояния очертания T после "слияния" очертаний T_3, T_2, T_1 соответственно). В соответствии с d_3 вычисляется очертание поддерева $T-T_3$ и это очертание заменяет очертание вершины T. Далее определяется длина d_2 отрезка $T-T_2$ учитывая уже новое очертание вершины T и очертание поддерева T_2, а затем обновляется очертание T включением картины T_2 на расстоянии d_2 (см. рис. 4-(b)). Наконец, аналогичным образом определяется d_1 для $T-T_1$ и конечное очертание T, которое и будет очертанием (под)дерева T (см. рис. 4-(c)). Ниже будет более точно описан алгоритм планирования ортогонального размещения 3-дерева.

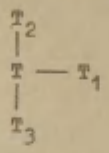
3.1. Основная процедура (ПЛАНДЕР)

Дано: (под)дерево T и направление S (на плоскости), в котором в T входит дуга от предшественника. Возможные направления дуг предполагаются закодированными следующим образом: S=1 - дуга направлена вправо, S=2 - вверх, S=3 - влево, S=4 - вниз.

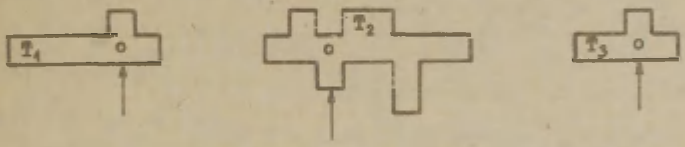
Результат: 1) дугам дерева T присвоены длины, обеспечивающие ортогональное размещение T; 2) с корнем дерева T связано очертание ортогонально размещенного дерева T.

Подпроцедуры: ОЧЕРТ, СБОР3, СБОР2, СБОР1, ПЛАНДЕР (рекурсивно).

Обозначения: через T_1, T_2, T_3 обозначаются поддеревья дерева T, причем $T_i = \Lambda$ означает отсутствие i-ого поддерева; через $\mathcal{O}_{T_1}$ обозначается очертание поддерева T_1 ($\mathcal{O}_{T_i}$ создается в ходе выполнения ПЛАНДЕР).

Дерево: 

Очертания поддеревьев:



Очертание корня T:

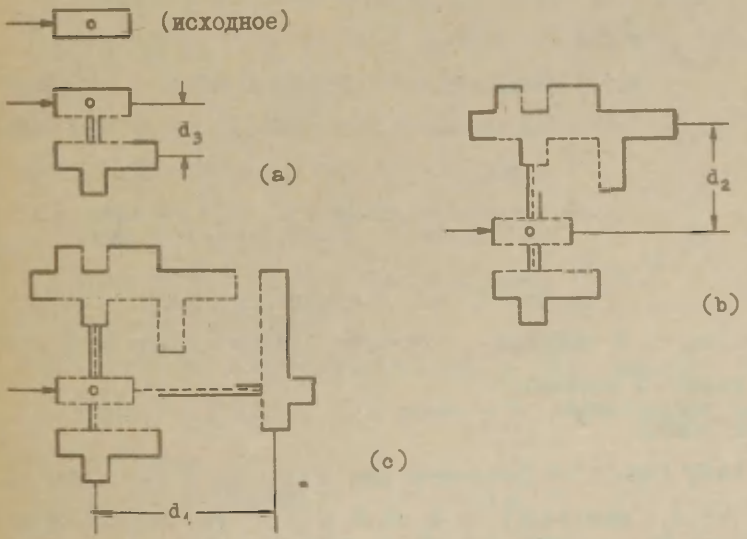


Рис. 4. Определение очертания дерева T.

Алгоритм:

0. Начало.

1. Если $T = \Lambda$, то КОНЕЦ.

2. Спланировать поддеревья:

2.1. ПЛАНДЕР(T_1, S);

2.2. ПЛАНДЕР$(T_2, (S+1)_{mod4})$;

2.3. ПЛАНДЕР$(T_3, (S-1)_{mod4})$.

3. (Процедура ОЧЕРТ) Определить очертание $\mathcal{O}$ исходной картины корня T, учитывая направление S.

4. Определить длины дуг $T \to T_3$, $T \to T_2$, $T \to T_1$ и очертание всего дерева T (спланировать дерево T, см. рис. 4): для i=3,2,1 выполнить 4.1;

4.1. если $T_i \neq \Lambda$, то (4.1.1 - 4.1.3);

4.1.1. $\mathcal{O}' := \mathcal{O}_{T_i}$;

4.1.2. (процедура СБОРi) исходя из очертаний $\mathcal{O}, \mathcal{O}'$ определить длину d_i дуги между картинами вершин T, T_i и новое очертание $\mathcal{O}$, полученное присоединением $\mathcal{O}'$ к $\mathcal{O}$ на расстоянии d_i;

4.1.3. дуге $T \to T_i$ присвоить длину d_i.

5. $\mathcal{O}_T := \mathcal{O}$ (найденное очертание $\mathcal{O}$ спланированного дерева T связывать с корнем).

6. КОНЕЦ.

Чтобы упростить выполнение действий, предусмотренных на шаге 4.1.2, целесообразно функции NH, RH, LW, RW рассматривать всюду определенными. Такое расширенное очертание определяется дополнительными формулами:

$NH_T(i) = +\infty$, если $i \notin I_T$;

$PH_T(i) = -\infty$, если $i \notin I_T$;

$LW_T(j) = +\infty$, если $j \notin J_T$;

$HW_T(j) = -\infty$, если $j \notin J_T$.

Таким образом, на шаге 3 определяется расширенное очертание $\mathcal{O}$, а на шаге 4.1.1 происходит расширение $\mathcal{O}_{T_i}$. На шаге 5 $\mathcal{O}_T$ присваивается ограниченное очертание. Заметим также, что после шага 4.1.2 очертания поддеревьев T_1, T_2, T_3 становятся ненужными.

3.2. Процедура ОЧЕРТ

Дано: картина корня T в общей сети координат и направление (входа в T) S.

Результат: сформировано расширенное очертание $\mathcal{O}$ данной картины в локальной для T сети координат.

Алгоритм прямой (сводится к определению центра, точек очертания и преобразованию координат).

3.3. Процедуры СБОР1, СБОР2, СБОР3

Для процедуры СБОРi (i=1,2,3) дано:

1) очертания $\mathcal{O}$ и $\mathcal{O}'$;

2) l - параметр разреженности; l есть слагаемое, которое прибавляется к минимальной длине дуги, определенной структурой дерева.

Результат процедуры СБОРi (i=1,2,3):

1) d_i - длина дуги между картинами, соответствующими очертаниям $\mathcal{O}$,$\mathcal{O}'$;

2) $\mathcal{O}$ - объединение очертаний $\mathcal{O}$,$\mathcal{O}'$ (на расстоянии d_i).

Обозначения I, J, NH, PH, LW, RW соответствуют очертанию σ, а I', J', NH', PH', LW', RW' – очертанию σ'.

Алгоритм процедуры СБОР1:

1. $d_0 := PH(0)$ (запоминается высота над центром).
2. $I_0 := I \cup I'$.
3. Найти расстояние между центрами:

$$d_1 = \max_{i \in I} (PH(i) - NH'(i)) + I + 1.$$

4. Изменить NH, PH:

для $i \in I' \setminus I$ $\quad NH(i) := NH'(i) + d_1$;

для $i \in I'$ $\quad PH(i) := PH'(i) + d_1$.

5. $J := \{ \min_{i \in I_0} NH(i), \min_{i \in I_0} NH(i)+1, \ldots, \max_{i \in I_0} PH(i) \}$;

$J_0 := \{0,1,\ldots,d_1\}$.

6. Изменить LW, RW:

для $j \in J_0$:
$$\begin{cases} LW(j) := \min(0, LW(j), LW'(k)) \\ RW(j) := \max(0, RW(j), RW'(k)); \end{cases}$$

для $j \in J \setminus J_0$:
$$\begin{cases} LW(j) := \min(LW(j), LW'(k)) \\ RW(j) := \max(RW(j), RW'(k)); \end{cases}$$

где $k := j - d_1$.

7. $I := I_0$ (σ сформировано).
8. Найти расстояние (длину дуги) между картинами:

$$d_1 := d_1 + NH'(0) - d_0 - 1.$$

Алгоритм процедуры СБОР2:

1. $d_0 := LW(0)$ (запоминается ширина, левее от центра).
2. $J_0 := J \cup I'$.
3. Найти расстояние между центрами:

$$d_2 := \max_{j \in J_0} (-L(j) - NH'(j)) + 1 + 1,$$

где

$$L(j) = \begin{cases} LW(j), & \text{если } j \in J \\ 0, & \text{если } j \notin J. \end{cases}$$

4. Изменить LW:

для $j \in I'$ $LW(j) := -(PH'(j) + d_2)$.

5. $l_0 := \{\min_{j \in J_0} LW(j), \min_{j \in J_0} LW(j)+1, \ldots, 0\}$.

6. Изменить NH, PH:

для $i \in I_0$, $i \geq -d_2$: $\begin{cases} NH(i) := \min(0, NH(i), LW'(k)) \\ PH(i) := \max(0, PH(i), RW'(k)); \end{cases}$

для $i \in I_0$, $i < -d_2$: $\begin{cases} NH(i) := \min(NH(i), LW'(k)) \\ PH(i) := \max(PH(i), RW'(k)); \end{cases}$

где $k := -i - d_2$.

7. $I := I \cup I_0$; $J := J_0$ (Θ сформировано).

8. Найти расстояние (длину дуги) между картинами:

$$d_2 := d_2 + NH'(0) + d_0 - 1.$$

Алгоритм процедуры СБОРЗ:

1. $d_0 := RW(0)$ (запоминается ширина, правее от центра).

2. $J_0 := J \cup (-I')$.

(Через -I' обозначено множество чисел, получаемое из I' умножением его элементов на -1.)

3. Найти расстояние между центрами:

$$d_3 := \max_{j \in J_0} (R(j) - NH'(-j)) + 1 + 1,$$

где

$$R(j) = \begin{cases} RW(j), & \text{если } j \in J \\ 0, & \text{если } j \notin J. \end{cases}$$

4. Изменить RW:

для $j \in -I'$ $RW(j) := PH'(-j) + d_3$.

5. $I_0 := \{0,1,\ldots,\max\limits_{j \in J_0} RW(j)\}$.

6. Изменить NH, PH:

для $i \in I_0$, $i \leq d_3$: $\begin{cases} NH(i) := \min(0,NH(i),-RW'(k)) \\ PH(i) := \max(0,PH(i),-LW'(k)); \end{cases}$

для $i \in I_0$, $i > d_3$: $\begin{cases} NH(i) := \min(NH(i),-RW'(k)) \\ PH(i) := \max(PH(i),-LW'(k)); \end{cases}$

где $k := i - d_3$.

7. $I := I \cup I_0$; $J := J_0$ (Ф сформировано).

8. Найти расстояние (длину дуги) между картинами:

$$d_3 := d_3 + NH'(0) - d_0 - 1.$$

ВЫЧИСЛЕНИЕ ОДНОСИМВОЛЬНОГО КОНТЕКСТА В КС-ГРАММАТИКАХ ПРИ ПОМОЩИ БИНАРНЫХ ОТНОШЕНИЙ

М. Томбак

Хорошо известны алгоритмы Мартина для вычисления отношений и векторов предшествования при помощи бинарных отношений (см. [1], [2]). В настоящей статье излагаются и обосновываются аналогичные алгоритмы для вычисления односимвольного контекста нетерминальных символов в контекстно-свободных грамматиках (КС-грамматиках).

Пусть X – некоторое множество, а α, β бинарные отношения на этом множестве. Через $\alpha\beta$ обозначим произведение отношений α и β, через α^+ – транзитивное замыкание отношения α, а через α^* – рефлексивное транзитивное замыкание отношения α. Вводим еще обозначения $a\alpha = \{b : (a,b) \in \alpha\}$ и $\alpha b = \{a : (a,b) \in \alpha\}$ для $a, b \in X$. Вместо $(a,b) \in \alpha$ используем иногда и обозначение $a \alpha b$.

Пусть $G = (V_N, V_T, P, S)$ – КС-грамматика, а $V = V_N \cup V_T$. Обозначим отношение непосредственной выводимости на множестве V^* через $\Longrightarrow$ и отношение непосредственной канонической выводимости через $\underset{r}{\Longrightarrow}$.

Канонический (m,k)-контекст $C_{m,k}(A)$ символа $A \in V_N$ опре-

деляется следующим образом (см. [4]):

$$C_{m,k}(A) = \{(x,y) : |x|=m, |y|=k, *^m S *^k \overset{*}{\underset{r}{\Longrightarrow}} uxAyv, yv \in (V_T \cup \{*\})^*\}.$$

Односимвольный канонический контекст нетерминального символа A – это множество $C_{1,1}(A)$, односимвольный левый канонический контекст – это множество $\mathfrak{L}C(A) = \{X : (X,\Lambda) \in C_{1,0}(A)\}$, а односимвольный правый канонический контекст – $RC(A) = \{T : (\Lambda,T) \in C_{0,1}(A)\}$, где Λ – пустое слово. Общий алгоритм для вычисления $C_{m,k}(A)$ можно найти в [3] (алгоритм 5.16). Более простые алгоритмы для специальных случаев $C_{1,1}(A)$, $\mathfrak{L}C(A)$ и $RC(A)$ заданы в [4]. Цель настоящей статьи – задать алгоритмы для вычисления множеств $\mathfrak{L}C(A)$, $RC(A)$ и $C_{1,1}(A)$ в терминах бинарных отношений.

Определим по КС-грамматике G следующие элементарные бинарные отношения на множестве $V \cup \{*\}$:

$$\varepsilon = \{(X,X) : X \in V \cup \{*\}\},$$
$$\varepsilon_T = \{(T,T) : T \in V_T \cup \{*\}\},$$
$$\alpha = \{(X,Y) : A \to uXYv \in P; u,v \in V^*\} \cup \{(S,*),(*,S)\},$$
$$\lambda = \{(A,X) : A \to Xv \in P; v \in V^*\},$$
$$\rho = \{(X,A) : A \to uX \in P; u \in V^*\},$$
$$\mu = \{(B,A) : B \to A \in P\}.$$

Для каждого нетерминального символа $A \in V_N$ определим еще три элементарных бинарных отношений:

$$\mu_1^A = \{(\bar{X},\bar{Y}) : \bar{B} \to uXAYv \in P; u,v \in V^*\},$$
$$\mu_2^A = \{(\bar{X},\bar{B}) : B \to uXA \in P; u \in V^*\},$$
$$\mu_3^A = \{(\bar{B},\bar{Y}) : \bar{B} \to AYv \in P; v \in V^*\}.$$

"Элементарность" этих отношений заключается в том, что каждое из них можно вычислить за один линейный просмотр КС-грамматики.

В статье [4] для приведенных[1] КС-грамматик доказаны следующие утверждения:

$$\mathfrak{LC}(A) = \{X : X \lessdot A \vee X \doteq A\}, \tag{1}$$

$$RC(A) = \{T : [A \lessdot T \vee A \doteq T \vee A \gtrdot T] \,\&\, T \in V_T \cup \{*\}\}, \tag{2}$$

$$C_{1,1}(A) = \gamma_1(A) \cup \gamma_2(A) \cup \gamma_3(A) \cup \gamma_4(A), \tag{3}$$

где

$$\gamma_1(A) = \{(X,T) : B \to uXAYv \in P \,\&\, T \in H_1(Y)\},$$

$$\gamma_2(A) = \{(X,T) : B \to uXA \in P \,\&\, T \in RC(B)\},$$

$$\gamma_3(A) = \{(X,T) : B \to AYv \in P \,\&\, X \in \mathfrak{LC}(B) \,\&\, T \in H_1(Y)\},$$

$$\gamma_4(A) = \{(X,T) : B \to A \in P \,\&\, (X,T) \in C_{1,1}(B)\},$$

$$H_1(Y) = \{T : Y \overset{*}{\underset{\Gamma}{\Rightarrow}} Tw \,\&\, T \in V_T\},$$

а символы $\lessdot$, $\doteq$ и $\gtrdot$ обозначают отношения предшествования (см. [3]), определяемые следующим образом: $\lessdot = \alpha\lambda^+$, $\doteq = \alpha$, $\gtrdot = \rho^+\alpha\lambda^*$.

Следующие теоремы дают представление алгоритмов вычисления односимвольного канонического контекста в виде выражений от указанных элементарных бинарных отношений. Такое представление позволяет намного облегчить программирование этих алгоритмов.

Теорема 1. Если $G = (V_N, V_T, P, S)$ – приведенная КС-грамматика и $A \in V_N$, то

---

[1] КС-грамматика называется приведенной, если она не содержит бесполезных символов и правил с пустой правой частью.

а) $\mathfrak{LC}(A) = (\alpha\lambda^*)A$,

б) $\mathfrak{RC}(A) = A(\varsigma^*\alpha\lambda^*\varepsilon_T)$.

Доказательство.

а) Используя (1) и определения отношений предшествования, получаем:

$$\begin{aligned}\mathfrak{LC}(A) &= \{X : X \lessdot A \vee X \doteq A\} = \\ &= \{X : (X,A) \in (\lessdot \cup \doteq)\} = \\ &= \{X : (X,A) \in (\alpha\lambda^+ \cup \alpha)\} = \\ &= \{X : (X,A) \in \alpha(\lambda^+ \cup \varepsilon)\} = \\ &= \{X : (X,A) \in \alpha\lambda^*\} = \\ &= (\alpha\lambda^*)A .\end{aligned}$$

б) Используя (2) и определения отношений предшествования, получаем:

$$\begin{aligned}\mathfrak{RC}(A) &= \{T : (A \lessdot T \vee A \doteq T \vee A \gtrdot T) \;\&\; T \in V_T \cup \{*\}\} = \\ &= \{T : [(A,T) \in (\lessdot \cup \doteq \cup \gtrdot)] \;\&\; (T,T) \in \varepsilon_T\} = \\ &= \{T : (A,T) \in (\alpha\lambda^+ \cup \alpha \cup \varsigma^+\alpha\lambda^*)\varepsilon_T\} = \\ &= \{T : (A,T) \in [\alpha(\lambda^+ \cup \varepsilon) \cup \varsigma^+\alpha\lambda^*]\varepsilon_T\} = \\ &= \{T : (A,T) \in (\alpha\lambda^* \cup \varsigma^+\alpha\lambda^*)\varepsilon_T\} = \\ &= \{T : (A,T) \in (\varepsilon \cup \varsigma^+)\alpha\lambda^*\varepsilon_T\} = \\ &= \{T : (A,T) \in \varsigma^*\alpha\lambda^*\varepsilon_T\} = \\ &= A(\varsigma^*\alpha\lambda^*\varepsilon_T).\end{aligned}$$

Лемма 1. Если $G = (V_N, V_T, P, S)$ – приведенная КС-грамматика, $A \in V_N$ и $Y \in V$, то

а) $H_1(Y) = Y(\lambda^*\varepsilon_T)$,

б) $\gamma_1(A) = \mu_1^A\lambda^*\varepsilon_T$,

в) $\gamma_2(A) = \mu_2^A\varsigma^*\alpha\lambda^*\varepsilon_T$,

г) $\gamma_3(A) = \alpha\lambda^*\mu_3^A\lambda^*\varepsilon_T$.

Доказательство.

а) В статье [1] доказано, что $\{(Y,X) : Y \overset{+}{\underset{r}{\Longrightarrow}} Xv\} = \lambda^+$. Используя этот результат, получаем:

$$H_1(Y) = \{T : Y \overset{*}{\underset{r}{\Longrightarrow}} Tw \;\&\; T \in V_T\} =$$

$$= \{T : (Y \overset{+}{\underset{r}{\Longrightarrow}} Tw \vee Y = T) \;\&\; T \in V_T\} =$$

$$= \{T : [(Y,T) \in \lambda^+ \vee (Y,T) \in \varepsilon] \;\&\; (T,T) \in \varepsilon_T\} =$$

$$= \{T : (Y,T) \in \lambda^* \varepsilon_T\} =$$

$$= Y(\lambda^* \varepsilon_T).$$

б) Используя пункт а) настоящей леммы, получаем:

$$\gamma_1(A) = \{(X,T) : B \to uXAv \in P \;\&\; T \in H_1(Y)\} =$$

$$= \{(X,T) : (X,Y) \in \mu_1^A \;\&\; (Y,T) \in \lambda^* \varepsilon_T\} =$$

$$= \{(X,T) : (X,T) \in \mu_1^A \lambda^* \varepsilon_T\} =$$

$$= \mu_1^A \lambda^* \varepsilon_T .$$

в) Используя пункт б) теоремы 1, получаем:

$$\gamma_2(A) = \{(X,T) : B \to uXA \in P \;\&\; T \in BC(B)\} =$$

$$= \{(X,T) : (X,B) \in \mu_2^A \;\&\; (B,T) \in \varrho^* \alpha \lambda^* \varepsilon_T\} =$$

$$= \{(X,T) : (X,T) \in \mu_2^A \varrho^* \alpha \lambda^* \varepsilon_T\} =$$

$$= \mu_2^A \varrho^* \alpha \lambda^* \varepsilon_T .$$

г) Используя пункт а) теоремы 1 и пункт а) настоящей леммы, получаем:

$$\gamma_3(A) = \{(X,T) : B \to AYv \in P \;\&\; X \in \alpha C(B) \;\&\; T \in H_1(Y)\} =$$

$$= \{(X,T) : (B,Y) \in \mu_3^A \;\&\; (X,B) \in \alpha\lambda^* \;\&\; (Y,T) \in \lambda^* \varepsilon_T\} =$$

$$= \{(X,T) : (X,T) \in \alpha \lambda^* \mu_3^A \lambda^* \varepsilon_T\} =$$

$$= \alpha \lambda^* \mu_3^A \lambda^* \varepsilon_T .$$

Теорема 2. Если $G = (V_N, V_T, P, S)$ – приведенная КС-грамматика и $A \in V_N$, то

$$c_{1,1}(A) = \Big[\bigcup_{B \in \mu^* A} (\mu_1^B \cup \mu_2^B \varrho^* \alpha \cup \alpha \lambda^* \mu_3^B) \Big] \lambda^* \varepsilon_T .$$

Доказательство. Легко увидеть, что (3) можно переписать в виде

$$c_{1,1}(A) = \bigcup_{B : B \overset{*}{\Longrightarrow} A} (\gamma_1(B) \cup \gamma_2(B) \cup \gamma_3(B)). \qquad (4)$$

Из определения отношения μ видно, что утверждение $B \overset{*}{\Longrightarrow} A$ равносильно утверждению $B \in \mu^* A$. Таким образом, (4) принимает вид

$$c_{1,1}(A) = \bigcup_{B \in \mu^* A} (\gamma_1(B) \cup \gamma_2(B) \cup \gamma_3(B)),$$

откуда, используя пункты б), в) и г) леммы 1, получаем:

$$c_{1,1}(A) = \bigcup_{B \in \mu^* A} (\mu_1^B \lambda^* \varepsilon_T \cup \mu_2^B \varrho^* \alpha \lambda^* \varepsilon_T \cup \alpha \lambda^* \mu_3^B \lambda^* \varepsilon_T) =$$

$$= \bigcup_{B \in \mu^* A} (\mu_1^B \cup \mu_2^B \varrho^* \alpha \cup \alpha \lambda^* \mu_3^B) \lambda^* \varepsilon_T =$$

$$= \Big[\bigcup_{B \in \mu^* A} (\mu_1^B \cup \mu_2^B \varrho^* \alpha \cup \alpha \lambda^* \mu_3^B) \Big] \lambda^* \varepsilon_T .$$

Литература

1. Martin, D.F., Boolean matrix methods for the detection of simple precedence grammars. Comm. ACM, 1968, 11, № 10, 685-687.
2. Martin, D.F., A Boolean matrix method for the computation of linear precedence functions. Comm. ACM, 1972, 15, № 6, 448-454.
3. Aho, A.V., Ullman, J.B., The Theory of Parsing, Translation and Compiling; vol.1: Parsing. Prentice-Hall, Englewood Cliffs, 1972. (Русский перевод: Ахо А., Ульман Дж., Теория синтаксического анализа, перевода и компиляции, том I: Синтаксический анализ. Изд-во "Мир", М., 1978.)
4. Вооглайд А.О., Томбак М.О., О проблемах редуцирования в грамматиках предшествования. Труды Таллинского Политехнического Института, 1975, № 386, 23-37.

СЕМАНТИКА РЕАЛИЗУЕМОСТИ ДЛЯ ЯЗЫКА С ПЕРЕМЕННЫМИ ПО РЕКУРСИВНО ПЕРЕЧИСЛИМЫМ МНОЖЕСТВАМ

Р. Пранк

1. Введение

Пусть $\mathcal{E}$ – решетка рекурсивно перечислимых подмножеств множества натуральных чисел $\mathbb{N}$, а $\mathcal{A}(\mathcal{E})$ – булева алгебра множеств, порожденная решеткой $\mathcal{E}$. Основные факты о решетке $\mathcal{E}$ можно найти в 12-ой главе монографии Роджерса [2] и в статье Лахлана [4].

В настоящей статье рассматривается семантика реализуемости для теоретико-решеточного языка первого порядка $\mathcal{L}$ с переменными $X_1, X_2, \ldots$ (для рекурсивно перечислимых множеств), с константными символами $\emptyset$ и $\mathbb{N}$, с функциональными символами $\cup$, $\cap$, $'$, и элементарными формулами вида $S = T$, где S и T – термы.

Классическая элементарная теория решетки $\mathcal{E}$ вводится в [4]. В [5] доказана разрешимость Π_2-фрагмента классической теории (добавленный там предикат $L(X)$ выразим в теории с описанным выше языком $\mathcal{L}$, но меняет несколько границы классов Σ_1 и Π_1). Имеется ряд результатов о невыразимости предикатов в классической элементарной теории. Например, в [6] приведена теорема Мартина о невыразимости гиперпростоты. Из

теоремы Соара ([7], теор. 3.2) следует невыразимость как Т-сводимости, так и всякой более сильной сводимости ($\leq_1$, $\leq_m$, $\leq_{tt}$, $\leq_{btt}$, $\leq_w$ и т.д.). Единственным нетривиальным выразимым предикатом, определенным первоначально в неалгебраических терминах, является пока гипергиперпростота ([2], теор.12-XX). Это показывает, что, как правило, теоретико-алгоритмические предикаты не выразимы на алгебраическом языке (с классической семантикой). Но большинство понятий теории рекурсивно перечислимых множеств имеет именно алгоритмический характер. Поэтому представляется естественным пользование здесь конструктивной семантикой, дающей возможность непосредственно выразить некоторые теоретико-алгоритмические предикаты. Такая семантика вводится в пункте 2 статьи. Пункт 3 сравнивает полученное понятие с реализуемостью арифметических формул. В остальной части работы доказывается выразимость ряда предикатов из теории рекурсивно перечислимых множеств.

На протяжении всей статьи $\{\varphi_x\}$ – фиксированная геделева нумерация частично-рекурсивных функций (ЧРФ) одной переменной и $\{W_x\}$ – соответствующая нумерация рекурсивно перечислимых множеств, т.е. $W_x = \{y \mid !\varphi_x(y)\}$. D_u обозначает конечное множество с каноническим индексом u и $D^{-1}(A)$ – канонический индекс конечного множества A. Через $j_i(x)$ обозначим функцию, значением которой является степень i-того простого числа в разложении x на простые множители.

2. Определение реализуемости

Отношение $e \textcircled{r} \mathfrak{A}$ ("число e реализует формулу $\mathfrak{A}$") определим для замкнутых формул языка $\mathcal{L} + \mathcal{W}$, полученного из $\mathcal{L}$ добавлением счетного множества константных символов $\mathbb{W}_0$, $\mathbb{W}_1, \mathbb{W}_2, \ldots$. Ясно, что при интерпретации новых констант как множеств $\mathbb{W}_i$ термы принимают значения из $A(\mathcal{E})$. В дальнейшем будем часто говорить о множестве T, где T – постоянный терм (ПТ) расширенного языка.

Определение. 1. Если S и T – ПТ расширенного языка, то $e \textcircled{r} (S = T) \rightleftharpoons e = 0$ и $S = T$ истинно.

2. Для пропозициональных связок повторяется определение Клини ([1]) для арифметических формул.

3. Пусть $\mathfrak{A}(X)$ – формула расширенного языка с одной свободной переменной X. Тогда

1) $e \textcircled{r} \exists X \mathfrak{A}(X) \rightleftharpoons e = 2^a \cdot 3^b \;\&\; a \textcircled{r} \mathfrak{A}(\mathbb{W}_b)$,

2) $e \textcircled{r} \forall X \mathfrak{A}(X) \rightleftharpoons (\forall x)[\varphi_e(x) \textcircled{r} \mathfrak{A}(\mathbb{W}_x)]$.

Называем формулу $\mathfrak{A}$ реализуемой и пишем $\textcircled{r} \mathfrak{A}$, если существует натуральное число, реализующее $\mathfrak{A}$. В конечном итоге мы не будем интересоваться формулами с вхождениями символов $\mathbb{W}_1$. Они описывают не рекурсивно инвариантные свойства, реализуемость такой формулы зависит от конкретного выбора нумерации $\{\varphi_x\}$ и т.д.. Теорией $\mathfrak{S}_A$ называем множество реализуемых формул языка $\mathcal{L}$.

Из определения реализуемости видно, что реализуемость формулы расширенного языка, полученной из $\mathfrak{A}(X_1, \ldots, X_n)$ под-

становкой замкнутых термов $T_1,\ldots,T_n$ вместо $X_1,\ldots,X_n$, зависит от выбора $T_1,\ldots,T_n$ только с точностью равенства множеств. Формально это выражается следующей леммой.

Лемма 2.1. Пусть $\mathfrak{A}(X_1,\ldots,X_n)$ – формула расширенного языка, e – натуральное число, а для ПТ $S_1,\ldots,S_n$, $T_1,\ldots,T_n$ имеют место равенства множеств $S_1 = T_1$, ... , $S_n = T_n$. Тогда

$$e \textcircled{r} \mathfrak{A}(S_1,\ldots,S_n) \Leftrightarrow e \textcircled{r} \mathfrak{A}(T_1,\ldots,T_n).$$

Это означает, что формулы выражают свойства множеств, а не свойства их записей термами расширенного языка.

3. Погружение теории $\mathfrak{E}_\alpha$ в арифметику

Будем считать, что ЧРФ φ_1 вычисляется машиной Тьюринга с номером 1. Пользуемся языком арифметики с функциональными буквами для всех примитивно-рекурсивных функций. В частности, пусть p – буква для такой примитивно-рекурсивной функции 3 аргументов, что

$$p(z,x,t) = \begin{cases} 0, & \text{если машина Тьюринга с номером } z \text{ работает} \\ & \text{над } x \text{ не более } t \text{ шагов,} \\ 1 & \text{в противном случае.} \end{cases}$$

Пусть $e\ r\ \mathfrak{A}$ означает, что число e реализует в смысле Клини арифметическую формулу $\mathfrak{A}$. Если реализуемость арифметических формул определена прямо для примитивно-рекурсивных термов, то имеем

$$e\ r\ (p(z,x,t) = 0) \Leftrightarrow e = 0\ \&\ p(z,x,t) = 0.$$

Для погружения теории $\mathfrak{E}_\alpha$ в арифметику (с логикой реали-

зуемости) определим оператор Ar. Оказывается, что для всякой формулы $\mathcal{A}$ языка $\mathcal{L} + \mathcal{W}$ будем иметь

$$(\mathrm{r})\,\mathcal{A} \iff r\,Ar[\mathcal{A}].$$

Во-первых определим индукцией по построению T оператор Ar для выражений вида $x \in T$, где x - натуральное число, а T - ПТ языка $\mathcal{L} + \mathcal{W}$.

Определение. 1. $Ar[x \in W_z] \rightleftharpoons (\exists t)[p(z,x,t) = 0]$.

2. $Ar[x \in \emptyset] \rightleftharpoons Ar[x \in W_k]$, где k - фиксированный индекс пустого множества. Аналогично для $x \in N$.

3. Пусть S и T - ПТ языка $\mathcal{L} + \mathcal{W}$. Тогда

a) $Ar[x \in T'] \rightleftharpoons \neg Ar[x \in T]$,

b) $Ar[x \in S \cap T] \rightleftharpoons Ar[x \in S] \,\&\, Ar[x \in T]$,

c) $Ar[x \in S \cup T] \rightleftharpoons \neg(\neg Ar[x \in S] \,\&\, \neg Ar[x \in T])$.

Ясно, что формула $Ar[x \in T]$ выражает в стандартной (классической) интерпретации предикат "$x \in T$". В пункте 3.c выбран вариант с отрицаниями по той причине, что, определив

$$Ar[x \in S \cup T] \rightleftharpoons Ar[x \in S] \vee Ar[x \in T]$$

получили бы для нерекурсивных W_i

$$r(\neg Ar[N = W_i \cup W_i']).$$

Лемма 3.1. (i). Если $x \notin T$, то $Ar[x \in T]$ нереализуема.

(ii). Если $x \in T$, то $r\,Ar[x \in T]$, причем реализующее число можно найти эффективно по x и T.

Доказательство леммы можно провести очевидной индукцией по построению терма T, следуя определению Ar.

Для элементарных формул языка $\mathcal{L} + \mathcal{W}$ определим

$$\mathcal{A}r[S = T] \rightleftharpoons (\forall x)[\mathcal{A}r[x \in S] \equiv \mathcal{A}r[x \in T]].$$

Лемма 3.2. Пусть S и T – ПТ языка $\mathcal{L} + \mathcal{W}$. Тогда

(i). Если $S \neq T$ (в смысле равенства множеств), то формула $\mathcal{A}r(S = T)$ нереализуема.

(ii). Если $S = T$, то $r\,\mathcal{A}r[S = T]$, причем реализующее число можно найти эффективно по S и T.

Доказательство. (i). Пусть например $x \in S$, $x \notin T$. Тогда, по лемме 3.1, $r\,\mathcal{A}r[x \in S]$ и по x и S можно найти число d, реализующее $\mathcal{A}r[x \in S]$. Если теперь $e\ r\,\mathcal{A}r[S = T]$, то по e, x и d можно получить реализующее число для $\mathcal{A}r[x \in T]$. Но это невозможно, так как по этой же лемме $\mathcal{A}r[x \in T]$ не реализуема.

(ii). Пусть $S = T$ и для x имеет место например $r\,\mathcal{A}r[x \in S]$. Тогда, по лемме 3.1, $x \in S$, а следовательно и $x \in T$. По 3.1(ii) получим теперь и реализацию формулы $\mathcal{A}r[x \in T]$. Аналогично рассматриваются $x \notin S$.

Следствие. $r(\neg\neg\mathcal{A}r[S = T] \supset \mathcal{A}r[S = T])$.

Доказательство. Из $r\neg\neg\mathcal{A}r[S = T]$ следует равенство $S = T$, а это значит, что можно эффективно получить и реализацию формулы $\mathcal{A}r[S = T]$.

Следствие показывает, что реализация арифметической формулы, выражающей "естественно" предикат "$S = T$", не содержит дополнительной информации. Этим оправдывается выбор числа 0 в качестве реализации элементарных формул $S = T$ в определении ⓡ.

Остается еще доопределить оператор $\mathcal{A}r$ для неэлементарных формул языка $\mathcal{L} + \mathcal{W}$. Положим

1. $A_n[\neg\mathfrak{A}] \rightleftarrows \neg A_n[\mathfrak{A}]$.

2. Если $\oplus$ – связка $\&$, $\vee$ или $\supset$, то

$A_n[\mathfrak{A} \oplus \mathfrak{B}] \rightleftarrows A_n[\mathfrak{A}] \oplus A_n[\mathfrak{B}]$.

3. $A_n[\forall x\mathfrak{A}(x)] \rightleftarrows (\forall x)A_n[\mathfrak{A}(\mathbb{V}_x)]$,

$A_n[\exists x\mathfrak{A}(x)] \rightleftarrows (\exists x)A_n[\mathfrak{A}(\mathbb{V}_x)]$.

Индукцией по построению формулы $\mathfrak{A}$ можно доказать следующую теорему.

Теорема 3.3. Для каждой формулы $\mathfrak{A}$ языка $\mathfrak{L} + \mathfrak{W}$ имеет место $\textcircled{r}\ \mathfrak{A} \Longleftrightarrow r A_n[\mathfrak{A}]$, причем равномерно по $\mathfrak{A}$ можно по $\textcircled{r}$-реализации формулы $\mathfrak{A}$ эффективно найти r-реализацию формулы $A_n[\mathfrak{A}]$ и наоборот.

С другой стороны, классическая элементарная теория решетки $\mathfrak{E}$ погружается навешиванием двойных отрицаний в теорию $\mathfrak{E}_n$. Это влечет выразимость в $\mathfrak{E}_n$ всех выразимых в классической теории предикатов.

4. Выразимость в $\mathfrak{E}_n$. Продуктивность и конечность

Определение. Называем определенный на элементах $A(\mathfrak{E})$ предикат $P(X_1,\ldots,X_n)$ выразимым в теории $\mathfrak{E}_n$, если существует такая формула $\mathfrak{A}(X_1,\ldots,X_n)$ языка $\mathfrak{L}$, что для любых ПТ расширенного языка $T_1,\ldots,T_n$ имеет место

$$\textcircled{r}\ \mathfrak{A}(T_1,\ldots,T_n) \Longleftrightarrow P(T_1,\ldots,T_n) = t.$$

Аналогично определяется выразимость предикатов, определенных на элементах $\mathfrak{E}$.

Иногда в дальнейшем пишем в формулах вместо подформул выражаемые ими предикаты, например $X \neq Y$ вместо $\neg(X = Y)$,

$X \subseteq Y$ вместо $X \cap Y' = \emptyset$.

Теорема 4.1. Пусть определенный на $\mathcal{A}(\mathcal{E})$ предикат $P(X_1,\ldots,X_n)$ выразим в $\mathcal{E}_n$. Тогда P рекурсивно инвариантен.

Доказательство. Пусть формула $\mathfrak{A}(X_1,\ldots,X_n)$ выражает предикат $P(X_1,\ldots,X_n)$, $T_1,\ldots,T_n$ – ПТ языка $\mathcal{L}+\mathcal{U}$, число e реализует формулу $\mathfrak{A}(T_1,\ldots,T_n)$, а p – рекурсивная перестановка $\mathbb{N}$. Для получения числа, реализующего формулу $\mathfrak{A}(p(T_1),\ldots,p(T_n))$, нужно каждую функцию f типа $\mathcal{E}^k \to \mathcal{E}$, закодированную в числе e, заменить на pfp^{-1}.

В [3] автором дано описание всех выразимых предикатов для элементарной теории рекурсивных множеств с логикой реализуемости. Выразимость там мало отличается от выразимости в классической теории. Оказывается, что для рекурсивно перечислимых множеств конструктивная семантика существенно расширяет класс выразимых предикатов.

Обозначим через $[|X| = 1]$ формулу

$$X \neq \emptyset \;\&\; \neg(\exists Y)[X \cap Y \neq \emptyset \;\&\; X \cap Y' \neq \emptyset].$$

Очевидно имеет место следующая

Лемма 4.2. Пусть S – ПТ языка $\mathcal{L}+\mathcal{U}$. Тогда

$$|S| = 1 \Leftrightarrow \textcircled{r}[|S| = 1] \Leftrightarrow 1\,\textcircled{r}[|S| = 1].$$

Аналогично можно выразить предикат "$|X| = n$" для любого n. В дальнейшем пишем $[|X| = 1]$ часто без квадратных скобок.

Множество A называется продуктивным, если существует такая ЧРФ Ψ, что

$$(\forall x)[W_x \subseteq A \Rightarrow [!\Psi(x) \;\&\; \Psi(x) \in A \setminus W_x]].$$

Для выражения этого предиката положим

$$\mathrm{Prod}(A) \rightleftharpoons (\forall X)[X \subseteq A \supset (\exists Y)[|Y| = 1 \;\&\; (Y \subseteq A \;\&\; Y \cap X = \emptyset)]] .$$

Теорема 4.3. Формула Prod(A) выражает в $\mathfrak{E}_\kappa$ предикат "A – продуктивное множество" для $A \in \mathcal{A}(\mathfrak{E})$.

Теорема является непосредственным следствием леммы 4.4.

Лемма 4.4. (i). Существует такая ОРФ f, что для любых ПТ S языка $\mathfrak{L} + \mathfrak{U}$ и натурального числа z

φ_z – продуктивная функция Ψ для S $\Rightarrow$ $f(z)$ ⓡ Prod(S).

(ii). Существует такая ОРФ g, что для любых ПТ S и натурального числа e

e ⓡ Prod( ) $\Rightarrow$ $\varphi_{g(e)}$ – продуктивная функция для S.

Доказательство. Для получения требуемых переходных функций достаточно расписать выражение e ⓡ Prod(S), учитывая определение ⓡ и лемму 4.2. В настоящей статье такая процедура подробно проделается в доказательстве леммы 4.7.

В классической элементарной теории $\mathfrak{E}$ предикат "A конечно" выражается формулой $(\forall Y)[Y \subseteq A \supset (\exists Z)(Y = Z')]$. Проставляя перед квантором существования $\neg$, можно получить формулу, выражающую предикат конечности в $\mathfrak{E}_\kappa$. Но нас интересует формула, реализация которой закодирует канонический индекс множества. Имитируя [3], положим

$$\mathrm{Fin}(A) \rightleftharpoons (\forall Y_1 Y_2)[Y_1' = Y_2 \supset A \subseteq Y_1 \vee \neg(A \subseteq Y_1)].$$

Замечание. Примененное здесь представление рекурсивного множества парой рекурсивно перечислимых множеств с $Y_1' = Y_2$ дает погружение теории $\mathfrak{R}_\kappa$ статьи [3] в теорию $\mathfrak{E}_\kappa$.

Теорема 4.5. (i). Пусть S – ПТ языка $\mathcal{L} + \mathcal{U}$. Если множество S бесконечно, то формула Fin (S) не реализуема.

(ii). Существует такая ОРФ f, что для любых ПТ S и натурального числа u

$$S = D_u \Rightarrow f(u) \textcircled{r} \mathrm{Fin}(S).$$

(iii). Существует такая ЧРФ g, что для любых ПТ S и натурального числа e

$$e \textcircled{r} \mathrm{Fin}(S) \Rightarrow D_{g(e)} = S.$$

Доказательство теоремы совпадает с доказательством теоремы 3.3 в [3]. "Равноинформативность" канонического индекса конечного множества X и реализации формулы Fin (X) дает возможность оперировать в $\mathcal{E}_4$ каноническими индексами.

ОРФ f называется функцией, показывающей негипериммунность множества B, если

1) $(\forall u)[D_{f(u)} \cap B \neq \emptyset]$,

2) $(\forall uv)[u \neq v \Rightarrow D_{f(u)} \cap D_{f(v)} = \emptyset]$.

Множество B называется гипериммунным, если оно бесконечно и не существует ОРФ f, показывающей негипериммунность B.

Лемма 4.6. Пусть $B \subseteq N$. Тогда

B бесконечно и не гипериммунно $\Leftrightarrow$

$$\Leftrightarrow (\exists \text{ОРФ } g)(\forall w)[D_w \cap D_{g(w)} = \emptyset \;\&\; D_{g(w)} \cap B \neq \emptyset]. \qquad (1)$$

Доказательство. $\Rightarrow$. Пусть функция f показывает негипериммунность множества B. Положим

$$g(w) = f((\mu u)[D_w \cap D_{f(u)} = \emptyset]).$$

Функция g всюду определена, так как с конечным множеством D_w могут пересекаться самое большее $|D_w|$ множеств из непере-

секающихся $D_{f(0)}, D_{f(1)}, \ldots$. Выполнение обоих членов конъюнкции очевидно.

$\Leftarrow$. Пусть g – функция из (1). Ясно, что B – бесконечно. Учитывая, что $D_0 = \emptyset$, положим

$$f(0) = g(0),$$

$$f(u+1) = g(D^{-1}(D_{f(0)} \cup \ldots \cup D_{f(u)})).$$

Ясно, что f – ОРФ. Пусть $u > v$. Тогда

$$D_{f(u)} \cap D_{f(v)} \subseteq D_{f(u)} \cap (D_{f(0)} \cup \ldots \cup D_{f(u-1)}).$$

Но по первой части конъюнкции в (1) и построению f это множество пусто. По второй части (1) имеем $(\forall u)[D_{f(u)} \cap B \neq \emptyset]$. Следовательно, f показывает негипериммунность B. Лемма доказана.

При записи в $\mathcal{E}_4$ предикатов вида $(\exists$ОРФ $f)(\forall x) \ldots$ пользуемся семантикой выражения $\textcircled{r}\forall X \exists Y \mathfrak{A}(X,Y)$. Раскодируя это по определению реализуемости, получим $\exists e \forall x \mathfrak{A}(W_x, W_{\varphi_e(x)})$, т.е. $(\exists$ОРФ $\varphi_e)(\forall x)\mathfrak{A}(W_x, W_{\varphi_e(x)})$. Пусть

$$\text{НН1}(B) \rightleftharpoons (\forall X)[\text{Fin}(X) \supset (\exists Y)[\text{Fin}(Y) \;\&\; (X \cap Y = \emptyset \;\&\; Y \cap B \neq \emptyset)]].$$

Лемма 4.7. Формула НН1(B) выражает для $B \in \mathcal{A}(\mathcal{E})$ предикат "B бесконечно и не гипериммунно".

Доказательство. Пусть число e реализует формулу НН1(S) для некоторого ПТ S. Для доказательства негипериммунности множества S покажем, как вычислить функцию g из леммы 4.6. Пусть $w \in N$. Можно найти такое число x, что $W_x = D_w$. По утверждению 4.5(11) можно также эффективно по w получить чис-

ло d, реализующее Fin (W_x). Тогда по определению ⓡ имеем

$$\varphi_{\varphi_e(x)}(d) \text{ ⓡ } (\exists Y)[\text{Fin } (Y) \,\&\, (W_x \cap Y = \emptyset \,\&\, Y \cap S \neq \emptyset)]. \qquad (2)$$

Обозначим до конца доказательства леммы $j_1(\varphi_{\varphi_e(x)}(d))$ для i=0,1 через c_i. Получим

$$c_0 \text{ ⓡ } [\text{Fin } (W_{c_1}) \,\&\, (W_x \cap W_{c_1} = \emptyset \,\&\, W_{c_1} \cap S \neq \emptyset)].$$

Из числа $j_0(c_0)$, реализующего Fin (W_{c_1}), можно по 4.5(iii) получить канонический индекс v множества W_{c_1}. Положив g(w) = v, имеем $D_{g(w)} = W_{c_1}$ и

$$D_w \cap D_{g(w)} = \emptyset \,\&\, D_{g(w)} \cap S \neq \emptyset.$$

Это значит, что по лемме 4.6 множество S не гипериммунно.

Наоборот, пусть S не гипериммунно и g – функция из леммы 4.6 для S. Опишем построение числа e, реализующего формулу $\Sigma^0_1(S)$. Имеем

$$e \text{ ⓡ } \Sigma^0_1(S) \Leftrightarrow (\forall x)(\forall d)[d \text{ ⓡ } \text{Fin } (W_x) \Rightarrow (2)].$$

Покажем, как вычисляется функция a(e,x,d) = $\varphi_{\varphi_e(x)}(d)$, предполагая d ⓡ Fin($W_x$). По утверждению 4.5(iii) из d можно получить канонический индекс $D^{-1}(W_x)$. Взяв $W_{c_1} = D_{g(D^{-1}(W_x))}$, получим по лемме 4.6

$$W_x \cap W_{c_1} = \emptyset \,\&\, W_{c_1} \cap S \neq \emptyset.$$

Реализация этой формулы вычисляется по определению ⓡ для &, ¬ и элементарных формул. Реализацию формулы Fin(W_{c_1}) можно по 4.5(ii) получить из канонического индекса $g(D^{-1}(W_x))$. Пусть c_0 – реализация формулы

$$\mathrm{Fin}\,(W_{c_1}) \;\&\; (W_x \cap W_{c_1} = \emptyset \;\&\; W_{c_1} \cap S \neq \emptyset).$$

Положим $a(e,x,d) = 2^{c_0} \cdot 3^{c_1}$,

$$e = \lambda x.\lambda d.a(e,x,d).$$

Ясно, что $e \in \mathrm{HH1}(S)$. Лемма доказана.

Пусть

$$\mathrm{HS}(A) \rightleftharpoons (\exists X)[X = A] \;\&\; \mathrm{Fin}\,(A') \;\&\; \neg\mathrm{HH1}(A').$$

Из теоремы 4.5 и леммы 4.7 непосредственно следует

Теорема 4.8. Формула $\mathrm{HS}(A)$ выражает в $\mathfrak{S}_A$ предикат "A гиперпросто" для $A \in \mathcal{A}(\mathfrak{S})$.

5. Предикаты сводимости

Будут рассмотрены следующие сводимости.

1. $A \leq_m B \rightleftharpoons (\exists \text{ОРФ } f)(\forall x)[x \in A \Leftrightarrow f(x) \in B]$.
2. $A \leq_q B \rightleftharpoons (\exists \text{ОРФ } f)(\forall x)[x \in A \Leftrightarrow D_{f(x)} \cap B \neq \emptyset]$.
3. $A \leq_c B \rightleftharpoons (\exists \text{ОРФ } f)(\forall x)[x \in A \Leftrightarrow D_{f(x)} \subseteq B]$.
4. Табличная сводимость. tt-условиями порядка $n > 0$ называются упорядоченные пары $\langle\langle x_1,\ldots,x_n\rangle,\alpha\rangle$, состоящие из n-ки чисел и n-арной булевой функции. tt-условие выполняется на множестве B, если для характеристической функции c_B множества B выполняется $\alpha(c_B(x_1),\ldots,c_B(x_n)) = 1$. Для tt-условий (сразу всех порядков) фиксируется эффективная нумерация и определяется

$$A \leq_{tt} B \rightleftharpoons (\exists \text{ОРФ } f)(\forall x)[x \in A \Leftrightarrow \text{tt-условие с номером } f(x) \text{ выполняется на } B].$$

5. Слабая табличная сводимость.

$A \leq_w B \Leftrightarrow (\exists z)(\exists \text{ОРФ } h)[c_A = \varphi_z^B \ \& \ (\forall x)[D_{h(x)}$ содержит все числа, о принадлежности которых множеству B задаются вопросы при вычислении $\varphi_z^B(x)]]$.

6. Ограниченно-табличная сводимость.

$A \leq_{btt} B \Leftrightarrow (\exists \text{ОРФ } f)(\exists m)[[f$ tt-сводит A к B] & $(\forall x)[$tt-условие с номером $f(x)$ имеет порядок $\leq m]]$.

Для выражения первых пяти предикатов введем формулы

$$\mathrm{Red}_m(A,B) \Leftrightarrow (\forall X)[|X|=1 \supset (\exists Y)[|Y|=1 \ \& \ (X \subseteq A \equiv Y \subseteq B)]],$$

$$\mathrm{Red}_q(A,B) \Leftrightarrow (\forall X)[|X|=1 \supset (\exists Y)[\mathrm{Fin}(Y) \ \& \ (X \subseteq A \equiv Y \cap B \neq \emptyset)]],$$

$$\mathrm{Red}_c(A,B) \Leftrightarrow (\forall X)[|X|=1 \supset (\exists Y)[\mathrm{Fin}(Y) \ \& \ (X \subseteq A \equiv Y \subseteq B)]],$$

$$\mathrm{Red}_{tt}(A,B) \Leftrightarrow (\forall X)[|X|=1 \supset (\exists Y)[(\mathrm{Fin}(Y) \& Y \neq \emptyset) \ \& \ (\forall D)[\mathrm{Fin}(D) \ \&$$

$$\& \ B \subseteq Y \supset (\exists V)[\mathrm{Fin}(V) \ \& (B = B \cap Y \supset (X \subseteq A \equiv V \neq \emptyset)]]]]],$$

$$\mathrm{Red}_w(A,B) \Leftrightarrow (\forall X)[|X|=1 \supset (\exists Y)[\mathrm{Fin}(Y) \ \& \ \{(\forall Z)[|Z|=1 \ \& \ Z \subseteq Y \supset$$

$$\supset (Z \subseteq B \vee \neg Z \subseteq B)] \supset (X \subseteq A \vee \neg X \subseteq A)\}]] \ .$$

Следующая теорема является непосредственным следствием лемм 5.2 и 5.3.

Теорема 5.1. Формулы $\mathrm{Red}_m(A,B),\ldots,\mathrm{Red}_w(A,B)$ выражают в $\mathcal{E}_\kappa$ предикаты $A \leq_m B,\ldots,A \leq_w B$ для $A,B \in \mathcal{A}(\mathcal{E})$.

Лемма 5.2. Существуют такие ОРФ g_1 и g_2, что для любых ПТ расширенного языка S и T и любых y, e $\in$ N

(i) $S \leq_m T$ функцией $\varphi_y \Rightarrow g_1(y) \textcircled{r} \mathrm{Red}_m(S,T)$,

(ii) $e \textcircled{r} \mathrm{Red}_m(S,T) \Rightarrow S \leq_m T$ функцией $\varphi_{g_2(e)}$.

Аналогично для $\leq_q$, $\leq_c$ и $\leq_{tt}$.

Доказательство. Для $\leq_m$. Пусть функция φ_y m-сводит S к T и пусть $|W_x| = 1$. Перечисляя множество W_x, найдем число

$x_0 \in W_x$. Положим $Y = \{\varphi_y(x_0)\}$. Ясно, что $|Y| = 1$ и $W_x \subseteq S \Leftrightarrow T \subseteq Y$. Формализация предписания получения Y дает реализующее число $g_1(y)$.

Наоборот, пусть $e \textcircled{r} \mathrm{Red}_m(S,T)$. Определим

$\varphi_{g_2(e)}(x)$ = единственному элементу множества Y, где Y получено при помощи реализации e из $X=\{x\}$.

Доказательство для $\leq_q$ и $\leq_o$ аналогично.

Для $\leq_{tt}$. (1). Пусть функция φ_y tt-сводит S к T и задан x, такой, что $|W_x| = 1$. Найдем такой x_0, что $W_x = \{x_0\}$. Вычислим tt-условие $\varphi_y(x_0) = \langle\langle x_1,\ldots,x_n\rangle,\alpha\rangle$. Положим $Y = \{x_1,\ldots,x_n\}$. Ясно, что можно получить характеристический индекс $D^{-1}(Y)$ и, далее, реализацию формулы

$$\mathrm{Fin}\,(Y) \;\&\; Y \neq \emptyset.$$

Покажем, что по x можно найти и реализацию формулы $(\forall D)[\mathrm{Fin}\,(D) \;\&\; \ldots]$. По каноническому индексу любого множества $D \subseteq Y$ можно получить канонический индекс множества V (ответа на вопрос "$x \in S$?"), где

$$V = \begin{cases} \{0\}, & \text{если } \alpha(c_D(x_1),\ldots,c_D(x_n)) = 1, \\ \emptyset, & \text{если } \alpha(c_D(x_1),\ldots,c_D(x_n)) = 0. \end{cases}$$

Если теперь $D = T \cap Y$ (т.е. 0 реализует эту формулу), то $V \neq \emptyset \Leftrightarrow V = \{0\} \Leftrightarrow \alpha(c_T(x_1),\ldots,c_T(x_n)) = 1 \Leftrightarrow x_0 \in S \Leftrightarrow X \subseteq S$.

(1●). Пусть $e \textcircled{r} \mathrm{Red}_{tt}(S,T)$. Покажем, как по числу x_0 найти tt-условие $\varphi_{g_2(e)}(x_0)$.

Построим индекс x рекурсивно перечислимого множества $X = \{x_0\}$. Знаем, что $1 \textcircled{r} [|X|=1]$. Тогда $\varphi_{\varphi_e(x)}(1) \textcircled{r} (\exists Y)[\ldots]$. Из реализации подформулы Fin (Y) получим канонический индекс

множества Y. Положим

$n = |Y|$,

x_i = i-тому элементу множества Y в порядке возрастания,

$$\alpha(c_1,\dots,c_n) = \begin{cases} 1, & \text{если для } D = \{x_i \mid c_i = 1\} \text{ получим } V \neq \emptyset, \\ 0 & \text{иначе.} \end{cases}$$

Ясно, что при каждом x описанная конструкция дает некоторое tt-условие. Далее имеем

$$x_0 \in S \iff X \subseteq S \iff \text{при } D = T \cap Y \text{ получим } V \neq \emptyset \iff$$
$$\iff \alpha(c_T(x_1),\dots,c_T(x_n)) = 1,$$

что и требуется в определении tt-сводимости.

Лемма 5.3. Существуют такие ОРФ g_1, g_2 и g, что для любых ПТ расширенного языка S и T и $y,z,e \in \mathbb{N}$

(i) [$S \leq_w T$, причем $c_S = \varphi_z^T$ и φ_y играет роль функции h из определения $\leq_w$] $\Rightarrow$ $g_1(y,z) \in \mathrm{Red}_w(S,T)$,

(ii) $e \in \mathrm{Red}_w(S,T) \Rightarrow (\forall x)[c_S(x) = \varphi_{g_2(e)}^T(x)$ и $D_{\varphi_{g(e)}}(x)$ содержит все нужные для вычисления $\varphi_{g_2(e)}^T(x)$ числа].

Доказательство. (i). Пусть выполнены условия и пусть дан такой x, что $|W_x| = 1$.

Найдем $x_0 \in W_x$, положим $Y = D_{\varphi_y(x_0)}$. Ясно, что Fin (Y). Если имеется число, реализующее формулу

$$(\forall Z)[|Z|=1 \;\&\; Z \subseteq Y \supset (Z \subseteq T \vee \neg Z \subseteq T)],$$

т.е. известно, какие числа из Y принадлежат T, то по предположению можно вычислить $\varphi_z^T(x)$. Выберем в дизъюнкции

$$\begin{cases} X \subseteq S, & \text{если } \varphi_z^T(x) = 1, \\ \neg X \subseteq S, & \text{если } \varphi_z^T(x) = 0. \end{cases}$$

Формализация описанного алгоритма дает число $g_1(y,z)$.

(11). Пусть $e \textcircled{r} \mathrm{Bed}_w(S,T)$. Покажем, как вычисляются $\varphi^T_{g_2(e)}(x)$ и $\varphi_{g(e)}(x)$.

По x можно найти индекс такого рекурсивно перечислимого множества X, что $X = \{x\}$. Ясно, что $|X| = 1$ и по лемме 4.2 верно $1 \textcircled{r} [|X| = 1]$. По индексу множества X можно теперь при помощи φ_e найти конечное множество Y, удовлетворяющее условию, заключенному в Bed_w в фигурные скобки, и число c, реализующее формулу $\{...\}$. Положим $D_{\varphi_{g(e)}(x)} = Y$. Для вычисления $\varphi^T_{g_2(e)}(x)$ узнаем при помощи оракула для каждого $z \in Y$ ответ на вопрос "$z \in T$?". По ответам найдем число d, реализующее формулу

$$(\forall Z)[|Z|=1 \;\&\; Z \subseteq Y \supset (Z \subseteq T \vee \neg Z \subseteq T)].$$

Положим

$$\varphi^T_{g_2(e)}(x) = \begin{cases} 1, & \text{если } j_0\varphi_c(d) = 0, \\ 0, & \text{если } j_0\varphi_c(d) = 1. \end{cases}$$

Ясно, что $\varphi^T_{g_2(e)}(x) = c_S(x)$ и при вычислении пользовались только числами из $D_{\varphi_{g(e)}(x)}$. Лемма доказана.

Последним рассмотрим btt-сводимость. Положим

$$\begin{aligned} \mathrm{Red}_{btt}(A,B) \rightleftharpoons\ & (\exists M)\{[\mathrm{Fin}(M) \;\&\; M \neq \emptyset] \;\&\; (\forall X)\{|X|=1 \supset \\ \supset\ & (\exists C)\{[C \subseteq M \;\&\; (\forall Y)[|Y|=1 \;\&\; Y \subseteq M \supset (\exists Z)[|Z|=1 \;\& \\ \&\ & (Y \subseteq C \equiv Z \subseteq B)]]] \;\&\; (\forall B)\{\mathrm{Fin}(B) \;\&\; B \subseteq M \supset \\ \supset\ & (\exists V)\{\mathrm{Fin}(V) \;\&\; (D = C \supset (X \subseteq A \equiv V \neq \emptyset))\}\}\}\}\}. \end{aligned}$$

Лемма 5.4. Существуют такие ОРФ g_1, g_2 и g, что для любых ПТ расширенного языка S, рекурсивно перечислимого мно-

жества W_i и $y,m,e \in \mathbb{N}$

(i) φ_y btt-сводит множество S к W_i с порядком $\leq m \Rightarrow$

$$\Rightarrow g_1(y,m,i) \textcircled{r} \mathrm{Red}_{btt}(S,W_i),$$

(ii) $e \textcircled{r} \mathrm{Red}_{btt}(S,W_i) \Rightarrow \varphi_{g_2(e)}$ btt-сводит S к W_i с порядком $\leq g(e)$.

Доказательство. (i). Если функция f btt-сводит множество A к B, то можно построить такую новую btt-сводящую функцию f_1, что все tt-условия с номерами $f_1(x)$ имеют порядок m и числа $x_1,\dots,x_m$ – попарно различны. Для этого нужно булевым функциям α, полученным по некоторым x, добавить фиктивные аргументы.

Пусть функция φ_y btt-сводит множество S к W_i с порядком всех tt-условий в точности m. Положим $M = \{1,\dots,m\}$ и укажем алгоритмы для всех функций, которые кодирует число, реализующее формулу $\mathrm{Red}_{btt}(S,W_i)$.

Пусть дан x, такой, что $|W_x| = 1$. Вычислим $x_0 \in W_x$ и находим tt-условие $\langle\langle x_1,\dots,x_m\rangle,\alpha\rangle$, имеющее номер $\varphi_y(x_0)$. Положим $C = \{j \mid 1 \leq j \leq m \,\&\, x_j \in W_i\}$. Ясно, что индекс множества C можно найти эффективно по i и tt-условию, а также имеем $0 \textcircled{r} [C \subseteq M]$.

Определяя для $Y \subseteq M \,\&\, |Y| = 1$

$Z = \{x_j\}$, если $Y = \{j\}$, имеем очевидно

$$|Y| = 1 \,\&\, Y \subseteq M \supset (Y \subseteq C \equiv Z \subseteq W_i).$$

Ввиду тривиальности реализации заключения этой импликации можно получить и реализацию подформулы $(\forall Y)[\dots]$ формулы Red_{btt}.

Множество V, кодирующее ответ на вопрос "$x_0 \in S$?", можно по каноническому индексу множества B вычислить, выбирая

$$V = \begin{cases} \{0\}, & \text{если } \alpha(c_D(1),\dots,c_D(m)) = 1, \\ \emptyset, & \text{если } \alpha(c_D(1),\dots,c_D(m)) = 0. \end{cases}$$

Имеем $\mathrm{Fin}(V)$. Пусть $D = 0$. Тогда

$$X = W_x \subseteq S \Leftrightarrow x_0 \in S \Leftrightarrow \alpha(c_{W_1}(x_1),\dots,c_{W_1}(x_m)) = 1 \Leftrightarrow$$

$$\Leftrightarrow \alpha(c_C(1),\dots,c_C(m)) = 1 \Leftrightarrow \alpha(c_D(1),\dots,c_D(m)) = 1 \Leftrightarrow$$

$$\Leftrightarrow V = \{0\} \Leftrightarrow V \neq \emptyset ,$$

что показывает, что конструкция дает реализацию формулы $\mathrm{Red}_{btt}(S, W_1)$.

(ii). Пусть $e \,Ⓡ\, \mathrm{Red}_{btt}(S, W_1)$. Вычислив по e канонический индекс множества M, положим $g(e) = |M|$.

Покажем, как строится по x_0 tt-условие $\varphi_{g_2(e)}(x_0)$. Положим $n = |M|$. По x_0 можно найти x, для которого $W_x = \{x_0\}$. Тогда $\varphi_{j_1 j_0(e)}(x) \,Ⓡ\, [|W_x| = 1 \supset (\exists C)\{\dots\}]$.

Пользуясь определением Ⓡ, получим из этого числа номер функции, сопоставляющей каждому одноэлементному $Y \subseteq M$ одноэлементное множество Z. Положим для $1 \leq j \leq n$ число x_j равным единственному элементу множества Z, полученного по $Y = \{k(j)\}$, где $k(j)$ – элемент множества M с номером j в порядке возрастания. Булеву функцию α определим путем

$$\alpha(c_1,\dots,c_n) = \begin{cases} 1, & \text{если при } D = \{k(j) \mid c_j = 1\} \text{ получается } V \neq \emptyset, \\ 0, & \text{если получается } V = \emptyset. \end{cases}$$

Проверим, что построенное tt-условие $\langle\langle x_1,\dots,x_n\rangle,\alpha\rangle$ удовлетворяет определению btt-сводимости. Имеем

$x_0 \in S \Leftrightarrow X = W_x \subseteq S \Leftrightarrow$ при $D = C$ получим $V \neq \emptyset \Leftrightarrow$

$\Leftrightarrow$ при $[c_j = 1 \Leftrightarrow k(j) \in C]$ получим $\alpha(c_1,\dots,c_n) = 1 \Leftrightarrow$

$\Leftrightarrow \alpha(c_C(k(1)),\dots,c_C(k(n))) = 1 \Leftrightarrow \alpha(c_{W_1}(x_1),\dots,c_{W_1}(x_n)) = 1,$

т.е. $x_0 \in S \Leftrightarrow$ tt-условие $\varphi_{g_2(e)}(x_0)$ выполняется на W_1. Ограниченность порядков tt-условий числом $g(e)$ очевидна. Лемма доказана.

Из леммы следует теорема 5.5.

Теорема 5.5. Формула $\mathrm{Red}_{btt}(A,R)$ выражает в $\mathcal{E}_t$ предикат $A \leq_{btt} B$ для $A \in \mathcal{A}(\mathcal{E})$, $R \in \mathcal{E}$.

Литература

1. Клини С.К., Введение в метаматематику. Москва, 1957.
2. Роджерс Х., Теория рекурсивных функций и эффективная вычислимость. Москва, 1972.
3. Пранк Р., Выразимость в элементарной теории рекурсивных множеств с логикой реализуемости. Уч. зап. ТГУ, 1979, 500, 119–128.
4. Lachlan, A.H., On the lattice of recursively enumerable sets. Trans. Amer. Math. Soc., 1968, 130, № 1, 1–37.
5. Lachlan, A.H., The elementary theory of recursively enumerable sets. Duke Math. J., 1968, 35, № 1, 123–146.
6. Soare, R.I., Automorphisms of the lattice of recursively enumerable sets. Part I: Maximal sets. Ann. Math., 1974, 100, № 1, 80–120.
7. Soare, R.I., Recursively enumerable sets and degrees. Bull. Amer. Math. Soc., 1978, 84, № 6, 1149–1181.

С о д е р ж а н и е

95 коп.