Федеральное государственное бюджетное учреждение науки
Институт проблем управления им. В.А. Трапезникова
РОССИЙСКОЙ АКАДЕМИИ НАУК

Бывайков М.Е.

ЯЗЫК ABIS. ОПИСАНИЕ ЯЗЫКА

Москва
ИПУ РАН
2013

УДК-004.432:004.434:004.8
ББК 32.973.26-018.1
Б 95

Бывайков М.Е. Язык ABIS. Описание языка [Электронный ресурс]: монография.− Электрон. текстовые и граф. дан. (0,6 Мб).− М.: ИПУ РАН, 2013.– 1 электрон. опт. диск (CD-R).− Систем. требования: IBM PC, Internet Explorer, Acrobat reader 3.0 и выше.−
ISBN 978-5-91450-128-7.

Монография представляет собой электронное издание комбинированного распространения и содержит полное описание языка ABIS, включая теоретические основы, синтаксис, семантику, правила разработки и отладки. Предназначена для широкого круга специалистов по программированию в качестве справочного руководства и учебного пособия.

Рецензенты: д.т.н. Р.Р. Бабаян, к.т.н. И.А. Степановская

Утверждено к печати Редакционным советом Института

Текст воспроизводится в виде, утвержденном
Редакционным советом Института

ISBN 978-5-91450-128-7

ОГЛАВЛЕНИЕ

1. ОБЩИЕ СВЕДЕНИЯ

Язык ABIS был разработан в ориентации на написание дедуктивных систем, в основу базы данных которых положена расширенная реляционная модель данных. Этим объясняются особенности и набор базовых объектов описываемого языка.

В разделе 2 дается краткая сводка понятий, относящихся к рассматриваемой области, и механизмов их реализации в языке ABIS.

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

2.1. Дедуктивная система

Основное назначение языка ABIS заключается в реализации дедуктивной системы. Опишем кратко, что она из себя представляет.

Дедуктивная система состоит из базы знаний (БЗ) и базы данных (БД). БЗ состоит из множества правил вида:

Если: < условие 1 >, ..., < условие n >

то: < следствие 1 >, ..., < следствие k >; достоверность

БД состоит из множества фактов, наделенных достоверностью:

(факт 1, достоверность 1),...,(факт l, достоверность l)

Основной процесс, происходящий в дедуктивной системе (именуемый циклом вывода), - это сопоставление фактов, имеющихся в БД, с некими образцами или шаблонами сопоставления, присутствующими в условиях правил БЗ, выявление условий, при которых наблюдается соответствие выставленным образцам и получение на их основе в следствиях правил соответствующих выводов, в результате чего происходит пополнение (модификация) БД. Описанный цикл вывода повторяется до тех пор, пока в БД наблюдаются изменения.

Рассмотрим следующий простой пример, который позволяет лучше понять принципы работы дедуктивной системы. На рис. 1 представлена схема технологического оборудования. Ниже приведена база знаний, состоящая из шести правил П1-П6 и база данных, содержащая первоначально три факта. При этом один из этих фактов является так называемым исходным событием, обнаружение которого в БД инициирует работу дедуктивной системы.

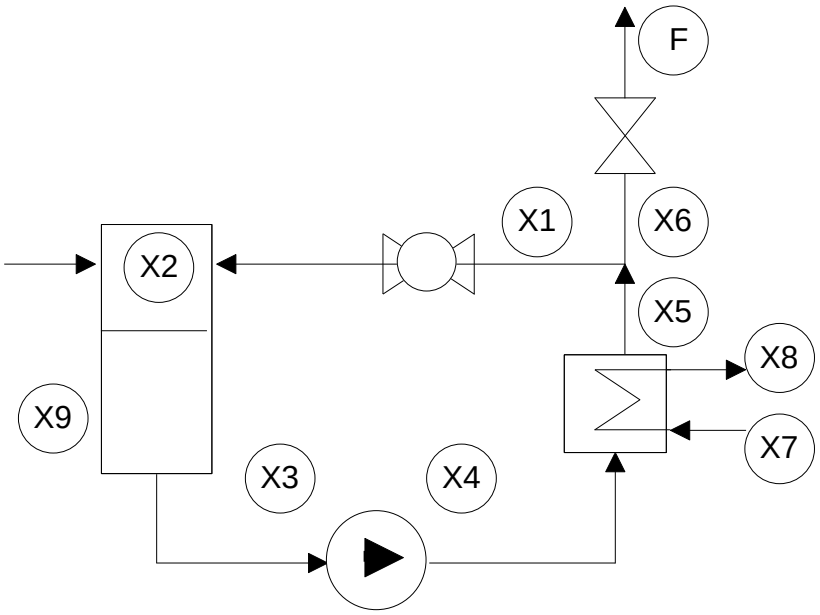

Рис. 1

База знаний

П1:	Если:	Насос отключен
	То :	Нет расхода на участке трубопровода x3,
		Нет расхода на участке трубопровода x4; 1.0
П2:	Если:	Нет уровня в резервуаре
	То :	Нет расхода на участке трубопровода x3; 1.0
П3:	Если:	Нет расхода на входе в теплообменник x4
	То :	Нет расхода на выходе из теплообменника x5; 1.0
П4:	Если:	Нет расхода на входе x5 в разветвление трубопроводов
	То :	Нет расхода на выходе x6
		Нет расхода на выходе x1; 1.0
П5:	Если:	Закрыта задвижка на участке x6
	То :	Нет расхода на участке трубопровода x6; 1.0
П6:	Если:	Закрыта задвижка на участке x1
	То :	Нет расхода на участке трубопровода x1; 1.0

База данных (исходное состояние)

f1: Задвижка на x1 открыта;1.0

f2: Задвижка на x6 открыта; 1.0

f3: Насос отключен 1.0 ⇨ исходное событие

База данных (первый цикл вывода)

f1: Задвижка на x1 открыта; 1.0

f2: Задвижка на x6 открыта;1.0

f3: Насос отключен; 1.0

П1⇨ f4: Нет расхода на x3; 1.0

f5: Нет расхода на x4; 1.0

База данных (второй цикл вывода)

f1: Задвижка на x1 открыта; 1.0

f2: Задвижка на x6 открыта; 1.0

f3: Насос отключен; 1.0

f4: Нет расхода на x3; 1.0

f5: Нет расхода на x4; 1.0

П3⇨ f6: Нет расхода на x5; 1.0

База данных (третий цикл вывода)

f1: Задвижка на x1 открыта; 1.0

f2: Задвижка на x6 открыта; 1.0

f3: Насос отключен; 1.0

f4: Нет расхода на x3; 1.0

f5: Нет расхода на x4; 1.0

f6: Нет расхода на x5; 1.0

П4⇨ f7: Нет расхода на x1; 1.0

f8: Нет расхода на x6; 1.0

На этом, при данном состоянии оборудования, процесс обновления базы данных дедуктивной системы прекращается. Таким образом, в рассмотренном примере исходный факт f3 порождает факты f4-f8.

2.2. Предложения и база данных языка

Основными элементами, из которых конструируются правила и факты языка ABIS, являются предложения. Каждое элементарное условие либо следствие правила, а также факты базы данных являются предложениями.

В основу при выборе структуры предложений в языке ABIS-системы была положена расширенная реляционная модель данных.

Базовыми понятиями этой модели являются "отношение", "атрибут", "домен" и "кортеж". Введем эти понятия на следующем простом примере. Пусть есть таблица с названием "Поставщик", состоящая из следующих столбцов:

"Предприятие"

"Товар"

"Обьем"

"Единица измерения"

в которой имеются две записи:

Предприятие	Товар	Обьем	Единица измерения
Электростанция SCP	Электроэнергия	30	MW
Металлургический завод	Прокат	1000	тыс. тонн

В терминах реляционных моделей данных таблица представляет собой "отношение", столбцы - "атрибуты", диапазон значений записей в столбце - "домен" атрибута, а вся строка таблицы целиком - "кортеж" отношения. Модель данных - это множество отношений:

R1(A11, ... , A1n)

R2(A21, ... , A2m)

...

Rk(Ak1, ... , Akl)

где Aij - атрибут j отношения Ri.

Каждую строку (кортеж) отношения R(A1, ..., An) можно представить в таком виде:

R(A1= s1, ... , An=sn)

где si - конкретное значение атрибута Ai. Например, первую строку отношения "Поставщик" можно записать так:

Поставщик (Предприятие = Электростанция_SCP,
Товар = электроэнергия,
Обьем = 30,
Единица_измерения = MW)

либо в краткой форме записи (опуская имена атрибутов):

Поставщик (Электростанция_SCP, электроэнергия, 30, MW)

Остановимся теперь на тех значениях, которые могут принимать атрибуты. Здесь возможны три основных варианта:

- терминальное значение;
- неопределенное (неизвестное или безразличное) значение;
- ссылка на дочерний кортеж одного из отношений модели данных.

В рассмотренном выше примере отношения "Поставщик" все приведенные значения атрибутов являются терминальными. Но если рассмотреть такой факт: "Имеется какая-то электростанция, производящая 100 MW электроэнергии", то для его представления потребовался бы кортеж отношения "Поставщик" следующего вида:

Поставщик (_ , электроэнергия, 30, MW).

здесь в качестве значения атрибута "Предприятие" используется символ (_), представляющий неопределенное значение.

Представим теперь другой случай. Пусть "Товар", производимый предприятиями, нельзя описать одним терминальным значением, и требуется ввести отношение "Товар", например, следующего вида:

Товар

Название	Дата производства	Способ доставки
прокат	декабрь	железная дорога

В этом случае факт: "Металлургический завод в декабре произвел 10 тыс. тонн проката и отгрузил его по железной дороге" можно записать в следующем виде:

Поставщик (Металлургический_завод,

Товар (прокат, декабрь, железная дорога),

10, тыс. тонн)

Здесь значением атрибута "Товар" отношения "Поставщик" является прямая запись соответствующего кортежа отношения Товар, которая в терминологии ABIS-ситемы называется "ссылкой", поскольку во внутреннем представлении (после компиляции) она действительно преобразуется в адресную ссылку.

В правилах языка ABIS конструкции вида:

R(A1= s1, ... , An=sn)

также называются предложениями. Однако предложений, в которых допускаются только три типа значений атрибутов (терминальное, неопределенное и ссылка) оказывается для описания базы знаний недостаточно. Рассмотрим следующий пример. Пусть, помимо уже введенного отношения "Поставщик" имеется отношение "Потребитель" с теми же именами атрибутов - Предприятие, Товар, Обьем и Единица_измерения.

Тогда правило "Если : имеется потребитель какого-либо товара, то: должен быть и поставщик этого товара" можно записать только, явно указав и у поставщика и у потребителя, что речь идет об одном и том же товаре. Это позволяет сделать механизм переменных

Если : потребитель (_, _T, _, _)

то : поставщик (_, _T, _, _).

Здесь наличие одной переменной _T в левой и правой частях правила говорит о том, что речь идет об одном товаре.

Таким образом, в правилах базы знаний допускается использовать также переменные в качестве значений атрибутов.

При работе дедуктивной системы основная процедура - это сопоставление фактов базы данных и условных частей правил базы знаний.

После того, как определена модель данных с помощью отношений, процесс сопоставления сводится к тому, что если в условной части правила имеется кортеж отношения R, то это правило может успешно сопоставиться только с фактами, являющимися кортежами отношения R, причем сопоставление должно вестись поатрибутно. Процесс сопоставления кортежей факта и правила называется согласованием. Ниже приводится таблица успеха (1) либо неуспеха (0) согласования для разных типов атрибутов факта и правила, реализованная в языке ABIS.

Атрибут факта _____ Атрибут Предложения Правила	неопределенное значение	Терминальное значение	ссылка
Неозначенная переменная	1	1	1
Неопределенное значение	1	1	1
Терминальное значение	0	Проверка на равенство	0
Ссылка	0	0	рекурсия

Для повышения быстродействия работы системы база данных разбивается на множества таким образом, что в процедуре согласования каждого предложения из правила с фактами базы данных участвуют только факты из одного множества. Процесс рационального разбиения базы данных на множества возлагается на программиста. В дальнейшем изложении эти множества называются обьектами типа factset.

2.3. Управление работой дедуктивной системы

Весь процесс управления сводится к выбору правила базы знаний, которое в данный момент должно согласовываться с фактами из выбранных множеств. Для этой цели правила в языке ABIS разбиваются на группы, называемые модулями (module). Внутри модуля правила обрабатываются последовательно, если не встречается явных указаний либо на выход из обрабатываемого модуля и возврат к модулю, вызвавшему обрабатываемый, либо на выход из обрабатываемого модуля и переход к обработке другого модуля, либо на переход к указанному правилу обрабатываемого модуля.

2.4. Метод согласования и нечеткие множества

Одной из важных компонент правил и фактов в языке ABIS является понятие достоверности. Достоверность - это числовая характеристика от 0 до 1 степени уверенности в данном правиле, либо факте в конкретных условиях, сложившихся на текущий момент в базе знаний и базе данных.

При исходной подготовке базы знаний и базы данных достоверность - субъективная, экспертная характеристика. В процессе функционирования дедуктивной системы для работы с механизмом достоверности используется развитие аппарата нечетких множеств, получившее название метод согласования. Изложим кратко его особенности.

Основными элементами, из которых конструируются правила и факты языка ABIS, являются понятия предложения и элементарного предложения. Каждое условие либо следствие правила, а также факты являются элементарными предложениями. Обьединение элементарных предложений является предложением. Таким образом, правило Если : r то : s с достоверностью x(s, r); связывает в причинно-следственную цепочку предложения s и r:

x(s,r)

r ----------> s

"s следует из r с достоверностью x(s r)".

Каждый факт можно интерпретировать как пару {s, p(s)} где s - предложение факта и p(s) - текущая достоверность этого предложения.

Метод согласования утверждает, что если есть база данных, состоящая из фактов

{s1, p(s1)},...{sn p(sn)} и имеется база знаний с функцией достоверности x(s,r), то для произвольного нового факта базы данных {s,p(s)} достоверность должна расчитываться по формуле:

$$p(s) = \max_{i=1,n} \min [x^{\wedge}(s,si), p(si)]$$

где x^(s,si) - транзитивное замыкание матрицы x(s,r).

Смысл транзитивного замыкания заключается в том, что из всех причинно-следственных цепочек базы знаний из предложения r в предложении s (на рис. 2 изображены две такие цепочки через предложения s1 и s2) выбирается такая, которая имеет критический путь максимальной достоверности.

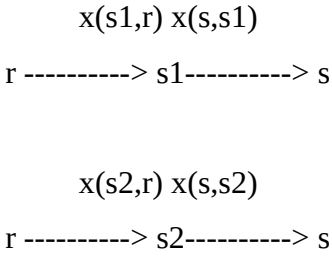

Рис. 2

Для примера, приведенного на рис. 2, значение x^(s,r) рассчитывается по формуле:

$$x^{\wedge}(s,r) = \max\{\min[x(s1,r), x(s,s1)], \min[x(s2,r), x(s,s2]\}.$$

3. ЭЛЕМЕНТЫ ЯЗЫКА

3.1. Общий синтаксис

3.1.1. Разделители

Пробелы, символы табуляции, перевода на новую строку и перевода на новую страницу, называемые обобщенными пробелами, используются в качестве разделителей. Вместо одного из них может использоваться любое их количество.

3.1.2. Комментарии

Комментарий - это последовательность символов, начинающаяся с /* и завершающаяся */, которая трактуется компилятором как односимвольный обобщенный пробел, но во всех других смыслах игнорируется.

Ограничители комментариев внутри комментариев не допускаются.

3.1.3. Идентификаторы

Идентификаторы - это имена, присваиваемые переменным, определяемым пользователям типам, отношениям, атрибутам отношений и модулям правил программы. Имена вводятся при описании переменных, отношений, определяемых пользователем типов. Имена модулей вводятся при записи заголовка модуля правил. Затем эти имена используются для ссылки на соответствующие объекты.

Идентификатор представляет собой последовательность из букв, цифр и знака подчеркивания, которая начинается с буквы или подчеркивания. Идентификатор может состоять из произвольного количества символов, но действительными будут только первые 16 символов.

Заглавные и строчные буквы в идентификаторах различаются. Пользовательские идентификаторы не могут совпадать с ключевыми словами языка ABIS-системы, а также с зарезервированными системными идентификаторами (см. раздел 3.1.4 "Ключевые слова и зарезервированные идентификаторы").

3.1.4. Ключевые слова и зарезервированные идентификаторы

Ключевые слова

Описатели	Declaration
	Types
	Relations
	Vars
	Key
	end_declaration
	Inset
	Outset
Кванторы	NOT
	ALL
	ANY
Типы данных	int
	long
	float
	string
	name
	ref
	factset
	fact
	module
	log
	enum
Операторы	IF
	THEN
	stop
	try
	exit

	continue
	break
Логические	SUCCESS [TRUE]
Константы	FAIL [FALSE]
Имена	Set_descriptor
отношений	rel_str

Зарезервированные идентификаторы - это имена встроенных процедур и функций, описанных в разделе 6.

3.2. Базовые типы данных

К базовым типам данных относятся целые числа (int, long), числа с плавающей точкой (float), строковые константы (string), имена (name), логические величины (log), ссылки на кортежи (ref).

В этом разделе дается синтаксис констант базовых типов и приводится объем памяти, занимаемой базовыми типами данных.

3.2.1. Целые константы

Целые (десятичные) константы (тип int) состоят из цифр 0-9 и принимают значения в диапазоне 16 или 32-разрядного представления целых чисел, в зависимости от используемой операционной системы.

Примеры: 12 111 957 1007

Замечание. Если значение превосходит наибольшее машинное целое со знаком, то старшие разряды числа усекаются.

3.2.2. Длинные целые константы

Длинные (десятичные) целые константы (тип long) явно определяются латинской буквой l или L, стоящей после константы и принимают значения в диапазоне 32 разрядного представления целых чисел.

Примеры: 121l = 121(десятичное); 956L = 956(десятичное)

3.2.3. Константы с плавающей точкой

Константа с плавающей точкой (тип float) всегда представляется числом с плавающей точкой двойной точности, т.е. как имеющая тип double в языке С и состоит из следующих частей:

- целой части - последовательности цифр;
- десятичной точки;
- дробной части - последовательности цифр;
- символа десятичной экспоненты e или E;
- самой десятичной экспоненты в виде целой константы, может быть со знаком.

Можно опустить:

- целую или дробную часть;
- десятичную точку или символ e(E) с экспонентой.

Примеры.

345 = 345 (десятичное)

3.14159 = 3.14159 (десятичное)

2.1E5 = 210000 (десятичное)

.123E3 = 123 (десятичное)

4037e-5 = .04037 (десятичное)

3.2.4. Строковые константы

Строковые константы (тип string) представляют собой строки символов кода ASCII , заключенные в кавычки: "...".

Примеры.

"This is a character string"

"Это - строковая константа"

В строковых константах можно использовать специальные (управляющие) символы кода ASCII в особой форме записи:

Новая строка (перевод строки)	HL(LF)	\n
Горизонтальная табуляция	HT	\t
Вертикальная табуляция	VT	\v
Возврат на шаг	BS	\b
Возврат каретки	CR	\r
Перевод формата	FF	\f
Обратная косая	\	\\
Апостроф	'	\'
Кавычки	"	\"

Кроме того, любой символ кода ASCII может быть представлен с точностью трех восьмеричных цифр: \0ddd.

Длинные строковые константы в файлах исходных текстов ABIS-программы (в модулях, наборах фактов, при инициализации переменных) можно переносить на следующую строку файла. Перенос обозначается символом '\', за которым следует символ новой строки или последовательность разделителей (пробелов или символов табуляции), заканчивающаяся символом новой строки. Ввод строковой константы возобновляется после символа новой строки (с первого символа следующей строки).

При выводе наборов фактов в файлы в виде исходных текстов длинные строковые константы автоматически переносятся на следующую строку файла, что обозначается символом '\', за которым следует символ новой строки.

3.2.5. Строки-идентификаторы

Строки-идентификаторы (тип name) представляют собой те же строковые константы, но записываемые без кавычек. Этот тип данных при использовании несет с собой повышенную вероятность ошибки, т.к. значения строк-идентификаторов не должны совпадать с именами переменных, ключевыми словами и зарезервированными идентификаторами.

3.2.6. Булевские константы

Булевские константы (тип log) принимают два значения: FAIL (синоним FALSE) и SUCCESS (синоним TRUE).

3.2.7. Ссылка на кортежи

Данные этого типа предназначены для организации фактов в виде деревьев из кортежей отношений.

Ссылки (данные типа ref) представляют собой кортежи отношений, включенные в другие кортежи в форме значений определенных атрибутов.

3.2.8. Указатели на модули

Данные этого типа (ref) представляют собой указатели на имена модулей языка ABIS. Они используются для неявного вызова модулей посредством оператора try (раздел 4.1.3.4.4).

3.2.9. Факты

Отдельный факт описываются по следующему формату:

имя_отношения (значение_атрибута_1,
...,
значение_атрибута_N
); [достоверность]

где:

имя_отношения	- это имя описанного отношения,
значение_атрибута	- это одно из терминальных значений соответствующего типа или неопределенное значение, обозначаемое символом подчеркивания (_)
Достоверность	- это константа плавающего типа в диапазоне от 0.0 до 1.0 (если она опущена, то берется максимальное значение - 1.0).

Таким образом, факт - это кортеж некоторого отношения, наделенный значением достоверности.

Пример: факт отношения window

Отношение window имеет следующие атрибуты.

Numbe - номер окна;

r

Type - тип окна;

Name - названия окна;

Param - состояние окна;

Для описания окна типа menu с заголовком Header, номером 2, в состоянии open, необходим следующий кортеж отношения window:

window (2,menu,Header,open)

3.2.9.1. Наборы фактов

Для выделения некоторой группы фактов из всего множества фактов в обособленный набор фактов нужно этой группе фактов предпослать заголовок:

factset имя_набора :

после чего явно перечислить все факты, включаемые в этот набор; здесь имя_набора должно быть заранее определено в качестве имени глобальной переменной типа factset в разделе описаний.

Пример множества фактов с именем _common, содержащего кортежи отношений process и window

factset _common:

process(main,init);

window(1, _ , _ , _); 1.0

window(2, _ , _ , _); 1.0

window(3, _ , _ , _); 1.0

window(4, _ , _ , _); 1.0

Данные этого типа (типа factset) представляют собой ссылки на адреса наборов фактов в оперативной памяти.

Эти данные используются для передачи параметров при вызове модулей правил и при выполнении процедур, которые имеют в качестве своих параметров наборы фактов.

Ссылки этого типа существуют только внутри выполняемой программы. Они не могут быть выведены на терминал, в отладчике и при выводе в файл. И наоборот, ссылки на наборы фактов не могут быть определены заранее в исходных текстах программ. Они могут быть сгенерированы только самой программой в процессе ее выполнения.

3.3. Перечислимые типы данных

Данные этого типа (тип enum) представляют собой списки возможных значений, явно определенные в программе.

Каждый список имеет свое имя и состав.

Использование перечислимых типов дает выигрыш в двух направлениях. Во-первых, данные этого типа при компиляции заменяются на целые числа, это экономит оперативную память и ускоряет работу. Во-вторых, использование перечислимых данных делает программу более строго типизированной, позволяя обнаружить многие ошибки на этапе компиляции.

Определение данных этого типа описано в разделе 4.2.1.

3.4. Преобразование типов

Язык ABIS является строго типизированным: все атрибуты в тексте ABIS-программы могут принимать значения только того типа, который был задан при определении соответствующего отношения. При несовпадении типов следует явно воспользоваться соответствующей функцией преобразования типов (6.2).

3.5. Неопределенные значения атрибутов

Поскольку при записи предложений и правил типичной является ситуация, когда значение одного или нескольких атрибутов в отношении несущественно или неизвестно (незаполнено), то в дополнение в диапазоны возможных значение всех системных типов и типов, определенных пользователем, вводится так называемое неопределенное значение атрибутов. Оно обозначается символом подчеркивания _. Фактически, кортеж отношения с

неопределенным значением одного из атрибутов означает множество кортежей, где этот атрибут принимает все допустимые для его типа терминальные значения и только наличие такого сорта значений позволяет ввести процесс означивания атрибутов: при совпадении известных значений атрибутов с шаблоном устанавливать значение неопределенных атрибутов.

4. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

4.1. Правила

Основной активной единицей языка являются правила, которые имеют следующий формат:

IF
левый_список_предложений
THEN
правый_список_предложений ; [достоверность]

где достоверность	- это константа плавающего типа в диапазоне от 0.0 до 1.0; если она опущена, то принимается максимальное значение 1.0.
списки-предложений	- это последовательность предложений языка, разделенных запятыми.

Левый_список_предложений называется условием правила, а правый_список_предложений - следствием правила.

4.1.1. Предложения в правилах

Предложения в правилах бывают следующих типов:

- оператор согласования фактов;
- оператор порождения фактов;
- оператор присваивания;
- логическое выражение;
- операторы управления;
- процедуры.

Большая часть типов предложений может использоваться как в условии, так и в следствии правила. Исключение составляют только логические выражения, которые в следствии правила не имеют смысла, а также операторы согласования и порождения фактов.

Оператор согласования всегда стоит в условии правила, оператор порождения всегда в следствии; в некоторых случаях они могут иметь совершенно одинаковую форму записи.

4.1.2. Операции и выражения

Логические операции

Логические операции осуществляются над данными типа log. Результат операций тоже типа log. Логические операции бывают следующих типов:

NOT - не

AND - и

OR - или

XOR - исключающее или

4.1.3. Операции сравнения

Операции осуществляются над скалярными типами. Результат операции типа log. Операции сравнения бывают следующих типов:

== - равно

!= - не равно

> - больше**Ошибка! Закладка не определена.**

< - меньше

>= - больше или равно

<= - меньше или равно

4.1.4. Арифметические операции

Операции осуществляются над данными типов int, long, float. Арифметические операции бывают следующих типов:

+ - сложение

- - вычитание или унарный минус

* - умножение

/ - деление

% - остаток от деления (только для int и long)

4.1.5. Операторы

Операторы языка ABIS синтаксически и по выполняемым действиям разделяются на операторы управления, операторы согласования и порождения, и оператор вызова модуля.

Операторы управления try, stop, exit, continue, break предназначены для управления порядком выполнения программы - модулей, правил и предложений в правиле. По стандартной схеме (по умолчанию) система выполняет правило, пока в рабочей области обнаруживаются еще необработанные комбинации фактов, и пытается выполнить модуль, пока он порождает новые факты (или хотя бы более достоверные).

Операторы stop, exit и break прекращают выполнение для всей программы, модуля или правила соответственно. Оператор continue, напротив, принуждает систему выполнять следующим определенное правило модуля, независимо от предыдущих попыток. Оператор continue с параметром - это аналог оператора типа go to с вычисляемой точкой перехода. Оператор try указывает, какой модуль надо выполнить и какие аргументы надо ему передать. Все операторы, кроме try, можно использовать без аргументов и следовательно, без скобок.

Операторы согласования и порождения синтаксически очень похожи (только при согласовании можно указывать кванторы: ALL, ANY, NOT, которые бессмысленны при создании нового факта), но их действие (поиск факта для согласования или создание нового факта) зависит от положения в правиле - в условии или следствии. Элементы, составляющие эти операторы разделяются пробелами.

4.1.6. Оператор согласования фактов

Оператор согласования служит для того, чтобы из заданного набора фактов извлечь один или несколько фактов, удовлетворяющих заданным условиям, с возможным присвоением извлеченному факту имени. Условия на извлекаемые факты накладывают обобщенный кортеж отношения и квантор.

Оператор согласования имеет следующий формат:

[Имя факта:] [Имя набора] [Квантор]

Обобщенный кортеж отношения,

где Имя факта - это частная переменная (идентификатор, начинающийся с подчеркивания), которая нужна только для последующей ссылки на согласованный с этим предложением факт.

Имя набора - это имя глобальной или локальной переменной типа factset, описывающее входной набор фактов, с которыми будет происходить операция согласования.

Если в операторе согласования набор не задан, то используется входной набор модуля. Если и описание входного набора модуля отсутствует, то согласование производится с фактами из анонимного набора (Unnamed_set).

Квантор - определяет способ согласования кортежа с фактами набора, и задается ключевыми словами ALL, NOT, ANY.

ALL - согласование со всеми фактами рабочего набора;

ANY - согласование до первого успеха, остальные факты, если даже и присутствуют, в расчет не принимаются;

NOT - согласование считается успешным, если в рассматриваемом наборе нет ни одного согласующегося факта.

Если квантор не задан, то по умолчанию используется квантор ALL.

Обобщенный кортеж отношения - задает шаблон, по которому будут согласовываться факты из рассматриваемого набора. Обобщенный кортеж отличается от обычного тем, что в нем допускается использование переменных. Переменная может быть глобальной, локальной или частной. Тип глобальной или локальной переменной должен совпадать с типом соответствующего атрибута. Частная переменная не описывается - она получает тип атрибута отношения.

Переменные в кортеже имеют различную семантику.

Глобальные и локальные переменные представляют свои текущие значения. Частные переменные служат для присвоения им в результате согласования значений, терминальных или неопределенных, с целью их последующего использования.

Более подробно о процедуре согласования и об использовании при этом переменных разных видов, см. раздел 5.2.

4.1.7. Оператор порождения фактов

Оператор порождения имеет следующий формат:

[Имя факта:] [Имя набора] Обобщенный кортеж отношения

Смысл оператора порождения заключается в том, что на базе обобщенного кортежа отношения, с учетом выявленных значений переменных, формируется факт, который заносится в указанный набор.

Все компоненты оператора порождения фактов синтаксически совпадают с соответствующими компонентами оператора согласования.

4.1.8. Оператор присваивания

Оператор присваивания заключается в присваивании переменной терминального или неопределенного значения.

Оператор присваивания имеет следующий формат:

[переменная]=[выражение]

или [переменная]=([логическое выражение])?

[выражение]

| [выражение],

где [логическое выражение] - выражение, содержащее логическую операцию или операцию сравнения.

Переменная может быть глобальной или локальной (но не частной).

Выражение представляет собой скобочную запись последовательности операндов и операций, причем типы операндов и операций должны соответствовать. Для согласования типов используют функции преобразования типов. Операнды это: константы, переменные, функции или выражения. Порядок вычислений выражений слева направо, операции (кроме

функций, которые отрабатываются в первую очередь) приоритетов не имеют и порядок выполнения должен задаваться при помощи скобок. Переменные в выражении могут быть любыми, но частные переменные должны быть предварительно означены, иначе они имеют пустое значение.

4.1.9. Операторы управления

Операторы управления используются для управления порядком выполнения правил в модуле.

Имеются следующие операторы управления:

- оператор продолжения continue;
- оператор возврата exit;
- оператор завершения stop;
- оператор вызова модуля try;
- оператор прерывания правила break.

4.1.10. Оператор продолжения

Оператор продолжения используется для того, чтобы продолжить выполнение модуля с n-го правила и имеет следующий формат:

continue [(n:int)]

Этот оператор прекращает выполнение текущего правила и передает управление на n-ое правило текущего модуля. Если параметр n отсутствует, имеет пустое значение или значение n = 0, то управление передается на первое правило. Если n больше числа правил в модуле, то выполнение аналогично оператору exit(). Правила нумеруются с 1. Для адресации правила рекомендуется использовать функцию rulenum().

4.1.11. Оператор возврата

Оператор возврата завершает выполнение текущего модуля и имеет следующий формат:

exit[(код возврата)],

где код возврата - это арифметическое выражение, которое возвращается как текущее значение согласования; значение кода возврата должно быть в диапазоне от 1.0 до 0.0 (иначе

оно выравнивается к ближайшей границе). В главном модуле start выполнение этого оператора аналогично оператору stop.

4.1.12. Оператор завершения

Оператор завершения

stop

приводит к немедленному завершению выполнения программы, независимо от того, где он встретился, и возврат управления к системе.

4.1.13. Оператор вызова модуля

Оператор вызова имеет следующий формат:

try имя_модуля ([a1 [, a2 [,...]]])

передает управление на обозначенный модуль и передает ему фактические аргументы, которыми могут быть выражения любой сложности. Здесь имя_модуля - это или идентификатор модуля, если он известен в точке вызова, или переменная типа module, получившая предварительно соответствующее значение. Порядок и тип фактических аргументов в операторе вызова должен соответствовать описанию модуля. Значения аргументов можно опускать (если они последние в списке, то можно опускать и разделяющие их запятые) - при этом соответствующие аргументы получают пустые значения.

При каждой активации модуля создается свой стек для аргументов, локальных и частных переменных, поэтому вполне допустимы рекурсивные вызовы модулей.

4.1.14. Оператор прерывания

Оператор break прекращает выполнение текущего правила и передает управление на следующее правило текущего модуля.

4.1.15. Частные (приватные) переменные правил

Кроме глобальных и локальных переменных в правилах может использоваться еще один вид переменных - частные (приватные) переменные правил. Эти переменные не описываются явно, а их типы определяются по контексту кортежа, в котором они впервые встречаются в правиле. Присвоение им значений происходит по рекурсивной процедуре означивания.

Синтаксически частные переменные имеют следующие ограничения:

имена частных переменных должны начинаться с символа подчеркивания _ (для глобальных и локальных переменных это не обязательно);

имена частных переменных должны отличаться от имен других переменных, доступных в данном правиле.

4.2. Способы структурирования программы

Полная исходная программа на языке ABIS включает:

базу знаний, состоящую из декларативной части (раздела описаний ABIS-программы) и продукционной части (модулей правил);

базу данных (множество фактов), описывающую исходное состояние дедуктивной системы. В процессе выполнения ABIS-программы база знаний не претерпевает изменений, а база данных меняется в процессе логического вывода.

ABIS-программа представляет собой два или более файлов исходного текста (раздел 7.2). Этот файл (или файлы) поступает в качестве входного на компилятор языка ABIS, в процессе компиляции происходит проверка синтаксиса и преобразование программы во внутреннее представление субъектов языка ABIS. Создание внутреннего представления заключается в построении древообразных структур отношений и правил, связанных взаимными ссылками.

Исходный текст программы состоит из следующих разделов:

- раздел описаний модели,
- раздел модулей правил,
- раздел наборов фактов.

Первые два раздела представляют собой базу знаний системы (соответственно ее декларативную и продукционную часть), третий - базу данных.

Если ABIS-программа состоит из нескольких файлов, то раздел описаний должен находиться в первом файле.

4.2.1. Описание модели

Описание модели начинается с ключевого слова declaration, заканчивается ключевым словом end_declaration и включает в себя следующие секции

секцию определяемых (программистом) типов данных,

секцию описания отношений,

секцию описания глобальных переменных, которые служат для обмена информацией между модулями правил.

Описание модели компилируется независимо от всех других частей программы (не имеет значения, занимает ли оно отдельный файл или же в этот файл наряду с описанием модели включены модули правил или наборы фактов), и создается отдельный объектный файл описания с именем "name.dcl", где "name" - это имя исходного файла, содержащего описание модели.

4.2.2. Описание определяемых типов данных

Секция описания определяемых типов данных служит для создания программистом своих собственных типов данных. Определяемые типы данных представляют собой явное перечисление множества значений (имен). Секция начинается с ключевого слова types, после которого идут операторы описания определяемых типов данных.

Оператор описания определяемых типов имеет следующий формат:

имя_определяемого_типа : enum (перечислимое_значение, ...);

где имя_определяемого_типа и перечислимое_значение - это идентификаторы.

Все идентификаторы, используемые при определении типов, должны быть различными и не должны совпадать с зарезервированными словами языка. После компиляции все перечислимые значения заменяются во внутреннем представлении на числовые величины типа int, которые семантически такими не являются.

Пример раздела типов данных, включающий два типа - wind_type и wind_param с соответствующими списками допустимых значений:

declaration

types

wind_type:enum(menu,text);
wind_param:enum(open,close);

4.2.3. Описание отношений

Секция описания отношений служит для описания структуры данных. Подраздел начинается с ключевого слова relation, после которого непосредственно идут описания отношений.

Описание отношения имеет следующий формат:

имя_отношения (имя_атрибута_1 : тип_атрибута, [key число]
...,
имя_атрибута_N : тип_атрибута,[key число]
);

где имя_отношения и имя_атрибута - идентификаторы.

Описатель key указывает на то, что описание имеет ключ. Оно может стоять только после одного из атрибутов (ключевого атрибута) отношения типа string или name.

После key должно стоять число, указывающее, сколько символов из значения атрибута будет выделяться для формирования ключа.

Имя отношения должно быть уникальным в рамках секции описаний отношений данных, а имя атрибута - в рамках отношения. В качестве типа атрибута могут использоваться базовые типы данных int, long, float, string, name, log, factset, ref, module или один из типов, определенных программистом в разделе types.

Ключ в языке ABIS служит для ускорения работы операторов согласования и порождения.

В отличие от реляционных баз данных, в языке ABIS допускается наличие произвольного числа фактов с одинаковым значением ключевого атрибута в одном наборе.

Пример подраздела отношений, описывающего структуру двух отношений Set_descriptor и window:

relation

Set_descriptor (name:string,
cod:int,
count:int,
time:long);

window (Number:int,
Type:wind_type,
Name :name,
Param :wind_param);

Отношение Set_descriptor является зарезервированным отношением языка ABIS, используемым при сетевых операциях с наборами фактов.

4.2.4. Описание глобальных переменных

В этой секции описываются глобальные переменные, доступные в любом модуле ABIS-программы, которые служат для обмена информацией между модулями правил. Подраздел начинается с заголовка vars, после которого расположены непосредственно описания глобальных переменных. Описание глобальной переменной имеет следующий формат:

имя_переменной : тип_переменной [= нач_значение] ;

Идентификаторы, используемые в качестве имен глобальных переменных, должны быть уникальными. Нельзя также использовать в качестве имен переменных ключевые слова и зарезервированные идентификаторы (раздел 3.1.4).

В качестве типа переменной могут использоваться базовые типы данных int, long, float, string, name, log, factset, module или же один из типов, определенных программистом.

Начальное значение можно задавать для типов переменных, определяемых пользователем, или одного из следующих базовых типов языка ABIS: int, long, float, string, name, log. Эти начальные присваиваются глобальным переменным при загрузке программы.

Начальное значение переменной может отсутствовать, в этом случае переменная получает неопределенное значение.

Для переменных типа factset задание начальных значений (т.е. включение в набор некоторого множества фактов) происходит в разделе наборов фактов и производится следующим образом:

factset имя-набора : перечисление соответствующих фактов

Пример секции описания глобальных переменных:

```
vars

_common :factset;
_locname1 : string;
_calculator :int = 0;
_cut :float = 0.3;
_window :int;
_format : name = undef;
_mode : string= "undef";
_load : wind_type;
_load1: wind_type = menu;

end_declaration
```

4.2.5. Раздел модулей правил

Продукционная часть базы знаний разбивается на модули. Модуль может содержать любое количество правил, концом модуля считается начало нового модуля или раздела наборов фактов.

Модули позволяют структурировать программу на логически независимые базы знаний; путем активизации модулей программист может управлять процессом логического вывода. Кроме того, наборы фактов и модули (наборы правил) можно загружать дополнительно в работающую систему, используя процедуры операций с файлами.

Модуль должен содержать хотя бы одно правило, например, правило вида:

```
IF (SUCCESS)
THEN exit;
```

4.2.6. Определение модуля

Определение модуля имеет следующую структуру:

заголовок модуля;

описание локальных переменных модуля;

описание входного и выходного наборов модуля;

тело модуля (последовательность правил).

Пример модуля правил с именем cycle, использующего глобальное множество IN и локальное множество OUT для работы с кортежами отношения process:

```
module cycle(IN:factset):
OUT:factset;

IF IN process(start,initialisation),
 OUT process(stop ,initialisation)
THEN exit;
```

4.2.7. Заголовок модуля

Заголовок модуля имеет следующий формат:

```
module имя_модуля ([имя_аргумента1: тип_аргумента1,
 ...,
 [имя_аргументаN: тип_аргументаN]]
 ) ;
```

Описание аргументов (формальных параметров) модуля может отсутствовать только в том случае, если при вызове модуля передача параметров не предусмотрена.

Здесь имена аргументов представляют собой локальные переменные, которые служат для передачи в модуль параметров, когда происходит вызов этого модуля при помощи оператора try.

В предыдущем примере строка

```
module cycle(IN:factset):
```

является заголовком модуля cycle.

4.2.8. Список локальных переменных модуля

Описание локальной переменной имеет следующий вид:

имя_переменной : тип [= начальное_значение];

что совпадает с описанием глобальной переменной.

Имена аргументов и локальных переменных в одном модуле должны быть разными, но могут совпадать с именами глобальных переменных - в этом случае соответствующие глобальные переменные в этом модуле затеняются и становятся недоступны.

Аргументы и локальные переменные существуют только в процессе работы модуля, они недоступны по имени из других модулей; при каждом вызове модуля в системном стеке заводится новый набор аргументов и локальных переменных, при выходе из модуля значения локальных переменных теряются, и занимаемая ими память освобождается. Таким образом, модули правил допускают рекурсию.

Аргументы (формальные параметры) получают значения при вызове модуля оператором try, локальные переменные при каждой инициализации модуля получают указанные начальные значения; по умолчанию аргументы и переменные получают пустое значение. Передача аргументов для всех типов данных, за исключением factset, происходит по значению. В случае типа factset значения передаются по ссылке и доступны как вызывающему, так и вызываемому модулям.

Отсюда следует, что в вызывающем модуле надо учитывать возможные изменения значений для типа factset (например, очистку), которые может произвести вызываемый модуль. Если по смыслу программы требуемый набор нужно передать по значению (т.е. исключить изменение исходного состояния), то надо или в вызывающем модуле перед вызовом, или в вызываемом модуле перед обработкой копировать содержимое набора в отдельную переменную процедурой unite или copy.

В предыдущем примере переменная OUT является локальной переменной типа factset для модуля cycle.

4.2.9. Входной и выходной наборы модуля

Описание этих наборов задает, какие наборы фактов будут являться входными и выходными в операторах согласования и, соответственно, порождения фактов; это описание действует на все правила модуля, если в операторах согласования или порождения явно не

указаны другие наборы. Задание каждого из этих наборов является необязательным. Если оно (частично или целиком) опущено, то в качестве входного или выходного набора используется непоименованный набор фактов, адресуемый глобальной переменной с именем Unnamed_set.

Входной и выходной наборы данных модуля описываются, соответственно, по следующему формату:

[inset имя_входного_набора;]

[outset имя_выходного_набора;]

здесь в качестве имени набора могут быть использованы имена глобальных переменных, а также имена аргументов или локальных переменных данного модуля. Содержимое наборов должно контролироваться программистом - например, использование в качестве входного набора локальной переменной предполагает, что до первого согласования в этот набор будут занесены какие-то факты, поскольку любое согласование с пустым набором всегда будет неудачным.

4.2.10. Описание модуля

Наряду с определением модуля возникает необходимость в описании модуля. Описание модуля задает шаблон модуля (имя, число, порядок и типы аргументов), который используется для последующих ссылок на этот модуль (в частности вызовов) и требуется только в том случае, если ссылка в тексте программы появляется раньше, чем определение (тело) этого модуля или если модуль определен в другом файле. Число, порядок и типы передаваемых параметров должны в общем случае совпадать с определением модуля. Отдельное описание необязательно, если определение модуля содержится в том же файле раньше всех ссылок на этот модуль. Описание модуля представляет собой заголовок модуля с последующей точкой с запятой:

module имя_модуля ([имя_аргумента 1: тип_аргумента],

...,

[имя_аргумента N: тип_аргумента]

) ;

аргументы могут отсутствовать.

Имя_аргумента в описании модуля несущественно и может быть произвольным, но внутри списка аргументов имена не должны повторяться.

4.2.11. Раздел наборов фактов

Раздел наборов фактов помещается в файле ABIS-программы после раздела модулей правил.

Раздел фактов программы может содержать произвольное количество наборов фактов (причем они могут быть разнесены по нескольким файлам) или может вообще отсутствовать.

Набор фактов должен иметь заголовок:

factset имя_набора :

после которого явно перечисляются все факты, включаемые в этот набор; здесь имя_набора должно быть заранее определено в качестве имени глобальной переменной типа factset в разделе описаний. Если указанное имя_набора не было заранее определено, то следующие за этим заголовком факты будут занесены в анонимный набор.

Пример множества фактов с именем _common, содержащего кортежи отношений process(Condition:name, State:name) и window :

```
factset _common:

process(main,init);
window( 1, _ , _ , _); 1.0
window( 2, _ , _ , _); 1.0
window( 3, _ , _ , _); 1.0
window( 4, _ , _ , _); 1.0
```

5. СРЕДСТВА ОБМЕНА ДАННЫМИ

5.1. Классы переменных и области видимости

Переменные в ABIS-программе бывают трех типов: глобальные, локальные и частные. Синтаксически все переменные записываются как идентификаторы.

Одиночный символ подчеркивания _ - это пустая переменная (в операторе согласования) или пустая константа (в остальных местах).

5.1.1. Глобальные переменные

Глобальные переменные ABIS-программы являются доступными в любом месте ABIS-программы. Глобальные переменные определяются в разделе описаний. Описание глобальной переменной сводится к указанию их типа и возможному присваиванию им начальных значения. Если начальное значение переменной не было присвоено, она получает пустое значение.

5.1.2. Локальные переменные

Локальные переменные являются динамическими (существуют только при работе модуля, в котором они описаны) и доступны только в этом модуле. Описание локальных переменных сводится к указанию их типа и возможному присваиванию им начальных значений. Если начальное значение локальной переменной не было присвоено, она получает пустое значение.

5.1.3. Частные переменные

Частные переменные являются динамическими (существуют только при работе соответствующего правила). Частные переменные служат для фиксации информации из полученных в процессе согласования фактов. Тип локальной переменной определяется при первом ее появлении в кортеже оператора согласования правила в соответствии с типом атрибута. До согласования они не имеют никакого значения (даже пустого).

5.2. Описание логической структуры

5.2.1. Выполнение программы на уровне модулей

Продукционная часть программы состоит из последовательности модулей правил. Выполнение программы начинается с первого по порядку модуля с именем start.

Модули выполняются либо последовательно, либо в результате непосредственных запросов на их выполнение, которые могут осуществляться из текущего правила программы с помощью оператора try.

Если в модуле нет явного выхода (оператора exit), то он будет выполняться до тех пор, пока во всей совокупности рабочих наборов модуля (рабочей области модуля) система логического вывода будет находить еще не обработанные факты.

Под рабочей областью модуля понимают все множества фактов, встречающиеся в теле модуля.

Если в процессе выполнения модуля выполнено последнее правило модуля, и к этому моменту исчерпаны все рабочие наборы, и не было порождено ни одного нового факта (либо потому, что не оказалось фактов, согласующихся с правилами модуля, либо потому, что все такие факты уже были обработаны и на их основе порождены все возможные новые факты и занесены в соответствующие наборы фактов), то модуль прекращает выполнение и возвращает управление к правилу, которое инициировало выполнение этого модуля, или заканчивает работу программы.

5.2.2. Выполнение программы на уровне правил

Правила, из которых состоит модуль, выполняются последовательно (если этому не препятствуют операторы управления: stop, exit, stop, continue и try). Каждое правило выполняется над фактами рабочей области данного правила.

Этот процесс является многопроходным, т.е. после прохода по всем фактам рабочей области правила, если были порождены новые факты, пополнившие рабочую область, то процесс запускается сначала.

5.2.3. Обработка условия

Обработка правила начинается с условия правила. Предложения, составляющие условие правила, обрабатываются по очереди.

При обработке предложения, если в нем самом явно задан набор фактов, то он и будет рабочим набором. При отсутствии явно заданного набора фактов рабочим набором будет входной набор модуля. Если же и входной набор не указан, то рабочим набором будет непоименованный набор фактов Unnamed_set.

Процесс обработки предложений в условии правила зависит от типа этих предложений. Предложения делятся на два принципиально отличающихся типа: процедурные и логического вывода (согласования).

Процедурные предложения (к которым относятся оператор присваивания, логические операторы и процедуры) выполняются точно так же, как и операторы в процедурных языках программирования. Эти операторы возвращают логическое значение "SUCCESS" в случае успешного выполнения или "FAIL", если они закончились неудачей (например, при использовании неозначенных локальных переменных в выражении, при попытке обращения к несуществующим файлам и так далее).

Оператор согласования выделяет из рабочего набора факты, удовлетворяющие поставленным в операторе условиям. Успехом оператора согласования является обнаружение такого факта. Если же ни один из фактов рабочего набора не удовлетворяет поставленным условиям, то логическим результатом оператора согласования будет неуспех.

5.2.4. Текущая достоверность

По результатам каждого успешного оператора согласования корректируется текущее значение достоверности условия правила. Текущее значение достоверности берется равным минимуму между прежним текущим значением достоверности и достоверностью согласованного факта. Исходное значение достоверности принимается равным 1.

5.2.5. Обработка следствия правила

Если обработка всех предложений условия на какой-то комбинации фактов рабочей области правила достигает успеха, то происходит переход к обработке следствия правила.

Обработка предложений следствия правила осуществляется так же последовательно, как и в условии правила. При этом выполнение процедурных предложений в следствии происходит точно так же, как и в условии правила.

Оператор порождения вызывает порождение нового факта по образцу обобщенного кортежа отношения (с учетом значений означенных локальных переменных) и занесение этого факта в рабочий набор. При этом достоверность порожденного факта принимается равной минимуму между полученной достоверностью условия (которая равна минимуму достоверности, согласованных фактов предложений условия) и достоверностью, приписываемой всему правилу.

После каждого срабатывания правила (выполнения следствия правила), если обработка правила не закончена, производится возврат к выполнению условия и процедура согласования повторяется для другой комбинации фактов из рабочей области. Таким образом, одно правило может породить несколько фактов (или не породить ни одного).

5.2.6. Выполнение оператора согласования в условии правила

Оператор согласования, структура которого представлена в 4.1.3, выполняется над подмножеством из текущего набора фактов, относящихся к тому отношению, которое указано в обобщенном кортеже оператора.

Подмножество рассматривается как список фактов {S1,S2,...,Sn}, где у каждого факта Si есть сосед справа Si-1 и сосед слева Si+1. Исключение составляют лишь крайние факты списка.

Работа оператора согласования состоит в последовательном просмотре всех элементов списка {S1,...,Sn}.

Однако, это осуществляется по-разному, в зависимости от используемого квантора: ALL, ANY или NOT.

5.2.7. Выполнение оператора согласования без квантора или с квантором ALL

Если список {S1,...,Sn} пуст, то происходит возврат к выполнению предыдущего оператора согласования. Если же такового нет, то выполнение правила завершается.

Первым просматривается 1-й факт, затем 2-й и т.д. Последним просматривается n-й факт.

Для произвольного i-го факта выполнение оператора состоит в следующем.

1 шаг: Выбирается факт Si; на факт Si+1, если i не равно n, устанавливается указатель в интерпретаторе языка ABIS.

2 шаг: Производится проверка на согласование всех атрибутов обобщенного кортежа оператора с соответствующими значениями атрибутов факта Si:

глобальные, локальные и частные означенные переменные в обобщенном кортеже заменяются на свои значения;

согласование атрибутов производится на основе таблицы успеха/неуспеха согласования, представленной в разделе 2.2.

3 шаг: В случае успеха согласования по всем атрибутам обобщенного кортежа выполняются следующие действия:

- частные неозначенные переменные из обобщенного кортежа означиваются и далее будут иметь значения при выполнении всех последующих предложений правила;
- текущая достоверность правила пересчитывается как минимум достоверности, полученной в предыдущих операторах согласования и достоверности Si;
- происходит рекурсивный переход к вычислению следующего предложения правила;
- происходит возврат к выполнению текущего оператора согласования.

4 шаг: Если Si не является последним в списке, шаги 1 - 4 повторяются для следующего за Si фактом Si+1.

5 шаг: происходит возврат к вычислению предыдущего оператора согласования. Если же такового нет, выполнение правила завершается.

При выполнении перехода от Si к Si+1 , проявляются особенности реализации языка ABIS, которые следует принимать во внимание при использовании операторов согласования.

В языке ABIS списки {S1,...,Sn} реализованы в виде так называемых кольцевых списков. Их можно представить в виде замкнутых колец (см. рис. 3), каждое звено которых содержит две ячейки: первая содержит указание на содержимое факта, вторая содержит ссылку на последующую ячейку.

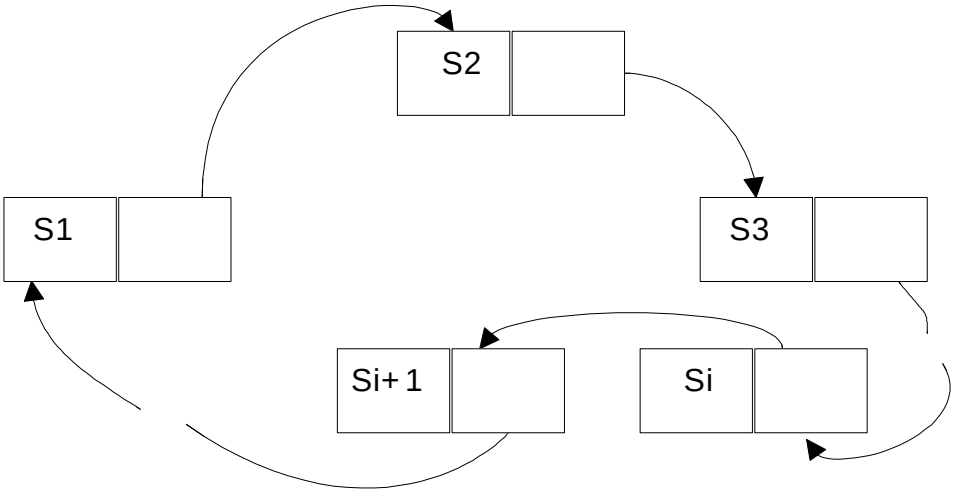

Рис. 3.

Поэтому, кортеж Si+1 может быть найден лишь в том случае, если Si содержится в базе данных языка ABIS.

На 1-м шаге указанной выше процедуры происходит установка указателя на звено, соответствующее Si+1 кортежу.

В ходе выполнения последующих предложений текущего правила могут использоваться процедуры удаления фактов, операторы порождения фактов и вызываться на выполнение другие модули. Это может привести к удалению или добавлению фактов в список {S1,...,Sn}.

Рассмотрим оба случая отдельно. Добавление новых фактов может происходить двумя вариантами. В случае, если отношение не имеет ключа, то добавленные факты помещаются в конец списка {S1,...,Sn,...}. В результате оператор согласования вначале будет выполнен для старых, а затем для добавленных фактов. При этом вновь могут быть добавлены факты и т.д. То есть таким путем можно организовать бесконечное порождение фактов в ходе выполнения одного правила. Это приведет к переполнению памяти и аварийному завершению ABIS-программы.

Второй вариант добавления фактов возникает в случае, если отношение имеет ключ. В этом случае факты могут быть добавлены либо до Si+1, либо после него, в зависимости от значения ключевых атрибутов. Это вносит элемент непредсказуемости в ход выполнения оператора согласования.

Удаление фактов может привести к аварийному завершению ABIS-программы, если будет удален факт Si+1, на который поставлен указатель в интерпретаторе языка.

Описанные ситуации, представленные выше, не диагностируются компилятором языка ABIS.

5.2.8. Выполнение оператора согласования с квантором ANY

Если список {S1,...,Sn} пуст, то происходит возврат к выполнению предыдущего оператора согласования. Если же такового нет, то выполнение правила завершается.

В противном случае факты из списка {S1,...,Sn} просматриваются подряд один за другим.

Для произвольного Si выполнение оператора состоит в следующем:

1 шаг: Производится проверка на согласование, аналогичная шагу 2 в разделе 5.2.6.

2 шаг: В случае успеха согласования выполняется следующее:

- частным неозначенным переменным присваиваются значения, которые сохраняются при выполнении всех последующих предложений правила;
- текущая достоверность правила рассчитывается как минимум достоверности, полученной в предшествующих операторах согласования, и достоверности Si;
- выполнение оператора считается успешным и происходит переход к выполнению следующего предложения правила.

3 шаг: Если Si не является последним в списке {S1,...,Sn}, шаги 1 - 3 повторяются для Si+1. Если Si последний в списке {S1,...,Sn}, то выполнение оператора считается неуспешным и осуществляется переход к выполнению предшествующего оператора согласования с квантором ALL (или без квантора). В случае отсутствия такового, выполнение правила завершается.

По смыслу, оператор согласования с квантором ANY является более строгим условием, чем оператор согласования с квантором ALL (или без квантора).

Он находит только один согласующийся факт из списка {S1,...,Sn} и осуществляет только один вариант означивания частных переменных.

5.2.9. Выполнение оператора согласования с квантором NOT

Если список {S1,...,Sn} пуст, то согласование считается успешным и происходит переход к выполнению следующего условия правила.

В противном случае факты списка {S1,...,Sn} просматриваются один за другим.

Для произвольного Si выполнение оператора состоит в следующем.

1 шаг: Проводится проверка на согласование, аналогичная шагу 2 в 5.2.6.

2 шаг: В случае успеха согласования выполнение оператора считается неуспешным и осуществляется переход к выполнению предшествующего оператора согласования с квантором ALL (или без квантора). Если такового не имеется, выполнение правила завершается.

3 шаг: Если Si не последний в списке {S1,...,Sn}, шаги 1 - 2 повторяются для Si+1. Если Si = Sn, выполнение оператора считается успешным и осуществляется переход к следующему предложению правила.

5.2.10. Особенности использования переменных при обработке правила

Глобальные переменные определяются и обрабатываются как обычные глобальные переменные процедурных языков программирования. Они доступны из любого правила любого модуля и могут служить для передачи данных от правила к правилу. Таким образом, они представляют собой как бы особую часть рабочей области, и используются для передачи данных между модулями правил в тех случаях, когда обычный способ передачи через наборы фактов невозможен или неэффективен. Глобальные переменные могут получать значения только через процедурные операторы (как правило, операторы присваивания), причем новое значение отменяет старое. Глобальные переменные могут быть означены в операторе согласования.

Локальные переменные определены только в пределах модуля правил, а в остальном ничем не отличаются от глобальных переменных.

Частные переменные служат для означивания (получения и удержания значений атрибутов фактов, согласованных в предложении согласования). Означенные частные переменные теряют свое значение при рекурсивном возврате в процессе поиска среди фактов рабочей области (раздел 5.2.6).

6. ВСТРОЕННЫЕ ЭЛЕМЕНТЫ

Встроенными элементами языка ABIS являются процедуры и функции.

Процедуры могут использоваться как отдельный оператор в отдельном предложении. Каждая процедура возвращает логическое значение: SUCCESS в случае успешного завершения, FAIL - в случае неудачи. Таким образом, они могут использоваться в условной части правила. Некоторые процедуры (например, n_done) только там и имеют смысл. Процедуры разделяются на два класса: с фиксированным числом аргументов и с переменным числом аргументов. Для первых число и тип аргументов жестко фиксированы и должны соответствовать данному описанию.

Аргументами могут быть константы, переменные или выражения любой сложности. Однако, надо учитывать, что по общим правилам вычисления аргументов для процедур и кортежей, если аргументы опущены, то им по умолчанию принимают пустые значения.

Формат описания процедур с фиксированным числом аргументов:

имя_процедуры (a1:тип1, a2:тип2, ... ai:типi)

здесь ... обозначает переменное число аргументов; для каждой конкретной процедуры число i фиксировано.

Формат описания процедур с переменным числом аргументов:

имя_процедуры (a1:тип1, a2:тип2, ...)

здесь ... обозначает переменное число аргументов для конкретного вызова этой процедуры, тип этих аргументов должен быть таким же, как и последний из описанных.

Функции могут иметь до двух аргументов, использоваться в выражениях, и их аргументы также могут быть выражениями произвольной сложности. Тип аргументов проверяется при компиляции и должен соответствовать описанию. Формат описания функций:

имя_функции(аргумент:тип_аргумента1,
аргумент2:тип_аргумента2):
тип_возвращаемого_значения

Если функция допускает аргумент несколько типов, то тип этого аргумента описывается переменной, которая подставляется в качестве аргумента.

6.1. Арифметические функции

Функция

min (x1:t, x2:t):t

определяет минимум аргументов x1, x2 типа t, где t - int, long, float, log, string, name или перечислимый тип.

Функция

max (x1:t, x2:t):t

определяет максимум аргументов x1, x2 типа t, где t - int, long, float, log, string, name или перечислимый тип.

Функция

pow (x1:float, x2:t):float

возводит аргумент x1 в степень аргумента x2 типа t, где t - int, long, float.

Функция

sin (x:float):float

вычисляет синус аргумента x, выраженного в радианах.

Функция

cos (x:float):float

вычисляет косинус аргумента x, выраженного в радианах.

Функция

atan (x:float):float

вычисляет арктангенс аргумента x, выраженного в радианах.

Функция

sqrt (x:float):float

вычисляет квадратный корень аргумента x.

Функция

ln (x:float):float

вычисляет натуральный логарифм аргумента x.

Функция

log10 (x:float):float

вычисляет десятичный логарифм аргумента x.

Функция

random (x:int):int

возвращает псевдо-случайное целое число. Аргумент x, если он не равен 0, используется для установки генератора случайных чисел: x = 1 реинициализирует генератор, любое другое число устанавливает генератор в случайную начальную точку.

Функция

abs (x:t):t

возвращает абсолютное значение аргумента x типа t, где t - int, long, float.

6.2. Функции преобразования типов

Функция

int (x:t):int

преобразует в целое число аргумент x типа t, где t - int, long, float, log или перечислимый тип.

Функция

long (x:t):long

преобразует в длинное целое число аргумент x типа t, где t - int, long, float, log или перечислимый тип.

Функция

float (x:t):float

преобразует в число с плавающей точкой аргумент x типа t, где t - int, long, float, log или перечислимый тип.

Функция

name (x:t):name

преобразует в строку-идентификатор аргумент x типа t, где t - name, string или перечислимый тип.

Функция

string (x:t):string

преобразует в строку аргумент x типа t, где t - name, string или перечислимый тип.

Функция

stoint (x:string):int

преобразует строку x в целое число.

Функция

stolong (x:string):long

преобразует строку x в длинное целое число.

Функция

stofloat (x:string):float

преобразует строку x в число с плавающей точкой.

Функция

enum(x:t1,y:t2):t2

преобразует параметр x, тип которого t1 - name, string или перечислимый тип, в значение функции перечислимого типа t2, который определяется типом параметра y, передающего функции свой тип, а не значение.

Функция

check_str (s:string, t:string):int

проверяет соответствие содержания строки s численному типу, задаваемому строкой t:

t="int" - целое;

t="long" - длинное целое;

t="float" - вещественное с плавающей точкой.

Функция check_str возвращает значение 0, если содержание строки s может быть успешно преобразовано в число типа, заданное значением параметра t, или значение -1 в противном случае.

6.3. Процедуры управления достоверностью

Предназначены для изменения стандартного способа вычисления достоверности - переключения на режим инкрементного вычисления (чем больше одинаковых фактов порождено, тем выше достоверность этого факта) или для явного вычисления и установки достоверности любым способом.

Процедура

set_trust (f:fact, t:float)

присваивает факту f значения достоверности t, причем t должно быть от 0.0 до 1.0, иначе t автоматически приводится к ближайшей границе этого диапазона.

Процедуры

incr_trust_on ()

incr_trust_off ()

осуществляют включение и, соответственно, выключение (переход в стандартный режим) инкрементного режима вычисления достоверности. В инкрементном режиме достоверность одинаковых порождаемых фактов суммируется, но не превышает максимального значения 1.0.

6.4. Процедуры и функции с фактами и наборами фактов

6.4.1. Удаление факта

Процедура

delete (a:fact)

удаляет факт, согласованный ранее. Аргументом здесь может быть или переменная типа факт, или целое число, равное порядковому номеру в правиле для оператора согласования, с которым согласован данный факт.

6.4.2. Запись факта

Процедура

append (f:fact, s:factset)

записывает факт f в набор s. Факт здесь идентифицируется так же, как в процедуре delete.

6.4.3. Замена факта

Процедура

replace (f:fact, a:кортеж)

заменяет факт f фактом, порожденным по шаблону второго аргумента. Замена допустима только для фактов одного отношения. Практически выполняется удаление факта f, формирование факта по кортежу и занесение его в тот же набор данных. Не гарантируется, что новый факт будет расположен на том же месте в наборе, что и старый.

6.4.4. Удаление всех фактов из набора

Процедура

clear (s:factset)

удаляет все факты из набора s. Переменная s получает пустое значение.

Процедура

clears (setname:string)

удаляет все факты из набора, имя которого задано строкой setname.

6.4.5. Объединение наборов

Процедура

unite (s1:factset, s2:factset)

объединяет факты из набора s1 и набора s2 в набор s2. Набор s1 не изменяется.

6.4.6. Процедуры копирования

Процедура

copy (s1:factset,s2:factset)

копирует набор s1 в набор s2. Набор s1 не изменяется, набор s2 предварительно очищается.

Процедура

copyn1 (s:string, set:factset)

копирует набор фактов, имя которого задано строкой s, в набор фактов set.

Процедура

copyn2 (set:factset, s:string)

копирует набор фактов set в набор фактов, имя которого задано строкой s.

Процедура

copys (setname1:string,setname2:string)

копирует набор фактов, имя которого задано строкой setname1, в набор фактов, имя которого задано строкой setname2.

6.5. Процедуры и функции для операций с файлами

6.5.1. Имена файлов в операционной системе UNIX

При работе в операционной системе UNIX, в которой различаются прописные и строчные буквы в именах файлов, в языке ABIS можно отменить или сохранить чувствительность имен файлов к использованию прописных или строчных букв. Для сохранения указанной чувствительности при вводе команды abx2 (раздел 7.1.) необходимо установить опцию /u. По умолчанию прописные и строчные буквы в именах файлов не различаются.

Функция

unixfname(mode:int):int

устанавливает или отменяет в процессе выполнения ABIS-программы чувствительность имен файлов к использованию прописных и строчных букв при работе в операционной системе UNIX: при отличном от нуля значении параметра mode устанавливает, при нулевом значении отменяет. В качестве своего значения функция unixfname возвращает предыдущее значение параметра mode.

6.5.2. Запись в файлы в формате MS-DOS

При работе в операционных системах UNIX, VAX-VMS перенос на следующую строку текстового файла обозначается одним, а не двумя символами, как в системе MS-DOS. Для записи в текстовые файлы в формате MS-DOS при работе в других средах необходимо установить опцию /f при вводе команды abx2 (раздел 7.1).

Функция

dosfile(mode:int):int

устанавливает или отменяет в процессе выполнения ABIS-программы режим записи в текстовые файлы в формате MS-DOS при работе в операционных системах, отличных от MS-DOS: при отличном от нуля значении параметра mode устанавливает, при нулевом значении отменяет. В качестве своего значения функция dosfile возвращает предыдущее значение параметра mode.

6.5.3. Процедуры экспорта наборов фактов в файлы

Процедура

export (filename:string, set:factset)

записывает набор фактов set в виде исходных текстов в файл с именем, заданным в строке filename. Набор f не изменяется.

Процедура

exportn (filename:string, set:factset)

записывает набор фактов set в файл с именем filename в виде исходных текстов. При этом, в отличие от процедуры export, в начало файла вставляется ключевое слово factset и имя записываемого набора фактов, заканчивающееся двоеточием (при условии, что набор описывается глобальной переменной). Это позволяет в дальнейшем компилировать полученный файл и загружать его содержимое в объектном виде. Факты, записанные в файл, могут быть также загружены в виде исходных текстов процедурой import (при этом название набора фактов, указанное в файле игнорируется).

Процедура

exportm(filename:string, set:factset, setname:string)

записывает набор фактов set в файл с именем filename в виде исходных текстов, вставляя в начало файла ключевое слово factset и заканчивающееся двоеточием имя, содержащееся в строке setname и, в общем случае, отличное от имени переменной set.

Процедура

exports (filename:string, setname1:string, setname2:string)

записывает факты из набора, имя которого задано строкой setname1, в файл с именем filename в виде исходных текстов, вставляя в начало файла ключевое слово factset и заканчивающееся двоеточием имя, содержащееся в строке setname2 (в общем случае имена setname1 и setname2 могут быть различными).

6.5.4. Процедуры импорта наборов фактов из файлов

Процедура

import (filename:string, set:factset)

загружает в набор set факты, записанные в файле с именем filename в виде исходных текстов (процедурами экспорта или тестовым редактором). Файл filename не изменяется.

Синтаксис описания фактов такой же, как и в исходной программе, но заголовок набора в данном случае не влияет на то, куда будут помещены факты.

Процедура

imports (filename:string, setname:string)

загружает в виде исходных текстов факты из файла с именем filename и записывает их в набор фактов, имя которого задано строкой setname (при этом название набора фактов, указанное в файле игнорируется).

6.5.5. Процедуры записи наборов фактов в файлы

Процедура

save (filename:string, set:factset)

сохраняет содержимое набора фактов set в объектном виде в файле с именем, заданным в строке filename. Набор set не изменяется.

Процедура

save_abx (filename:string, set:factset)

записывает набор фактов set в файл с именем filename в объектном виде, отличном от результата выполнения процедуры save. Факты, записанные в файл, в дальнейшем могут быть введены в программу процедурами load_abx, load_abn (а не load) или при начальной загрузке программы.

6.5.6. Процедуры загрузки наборов фактов из файлов

Процедура

load (filename:string, set:factset)

загружает факты, записанные в файле с именем filename процедурой save, в набор фактов set. Файл filename не изменяется.

Процедура

load_abx (filename:string)

загружает объектный файл правил (файл типа .rul), имя которого задано в строке filename. При этом модули правил загружаются заново, независимо от того, были ли они уже загружены. Наборы фактов, содержащиеся в этом файле добавляются к наборам фактов, заданным при компиляции (в заголовке factset имя_набора:).

Процедура

load_abn (filename:string)

загружает из файла с именем filename набор фактов в объектном виде, полученном путем компиляции исходного текста или с использованием процедуры save_abx. При этом, в отличие от процедуры load_abx, включение имени filename в make-файл загрузки программы не обязательно.

6.5.7. Процедуры удаления

Процедура

free_module (m:module)

удаляет тело модуля, заданного именем модуля m. Заголовок (описание) модуля сохраняется.

Дальнейшие обращения к модулю будут отрабатываться как обращения к пустому модулю.

Процедура

free_abx (filename:string)

удаляет все модули (заголовки остаются), входящие в объектный файл (типа .rul), имя которого задано в строке filename. Наборы фактов, входящие в этот файл, если они загружены в рабочую область ранее (при начальной загрузке или явно процедурой load_abx()), не удаляются.

Процедура

remove_file (filename:string)

удаляет файл, заданный именем в строке filename. Если путь не задан явно в имени файла, то подразумевается текущая директория.

6.6. Функции ввода-вывода строк

Функция

fopen (fname:string, mode:int):int

открывает файл с именем filename в режиме mode, возвращая при успешном завершении номер этого файла, используемый затем в качестве параметра функций fgets,

fputs, fclose. Параметр mode принимает те же значения, что и второй параметр функции open языка C, в частности:

mode=0 - файл открывается для чтения;

mode=1 - файл открывается для записи (соответствует функции creat языка C);

mode=2 - файл открывается для чтения и записи.

При ошибке функция fopen возвращает значение -1.

Функция

fgets (s:string, filen:int):int

считывает из файла с номером filen (предварительно открытого функцией fopen) строку - последовательность символов до очередного разделителя (пробела, символа табуляции или новой строки), возвращая содержание этой строки в виде значения переменной s. В качестве своего значения функция возвращает число введенных символов или -1 при ошибке.

Функция

fputs (s:string, filen:int):int

записывает строку s в файл с номером filen (предварительно открытый функцией fopen), возвращая в качестве своего значения число записанных символов или -1 при ошибке.

Функция

fclose (filen:int):int

закрывает файл с номером filen, открытый функцией fopen, возвращая 0 при успешном завершении и -1 при ошибке.

6.7. Процедуры стандартного ввода/вывода

Процедуры данного раздела обеспечивают ввод данных с клавиатуры и вывод информации на экран или в строковую переменную в символьной форме. Процедуры putv и sputv имеют переменное число аргументов. При описании процедур под простыми типами понимаются скалярные типы данных int, long, float, name, string, enum-типы.

Процедура

puts (a:t)

преобразует единственный аргумент a простого типа t в символьную форму и выводит результат на экран (в поток stdout в терминах языка C).

Процедура

gets (s:string)

вводит строку (кончающуюся по Enter) с клавиатуры (из потока stdin в терминах языка C) и записывает результат в строковую переменную s. Предыдущее значение строковой переменной (если было) теряется.

Процедура

putv (a:простой_тип, ...)

преобразует последовательность аргументов простого типа в символьную форму и выводит результирующую строку на экран (в поток stdout).

Процедура

sputv (v:string, a:простой_тип, ...)

преобразует последовательность аргументов a1 простого типа в символьную форму и записывает результат в переменную v. Предыдущее значение v теряется.

Процедура

printf (s:string, a1:простой_тип, a2:простой_тип,...)

преобразует последовательность аргументов a1,a2,… простого типа в символьную форму и выводит результат на экран в поток stdout под управлением форматной строки s. Полный аналог одноименной функции языка C. Можно использовать следующие форматные символы:

c s d u o x X f e E g G

кроме того можно использовать управление шириной и точностью поля и символы: #, -, *, +.

Процедура

sprintf (v:string, s:string, a1:простой_тип, a2:простой_тип,...)

преобразует последовательность аргументов a1,a2,… простого типа в символьную форму и записывает результат в переменную v под управлением форматной строки s. Полный аналог одноименной функции языка C. Допустимые форматные символы такие же, как для printf. Предыдущее значение v теряется.

6.8. Работа со строками

Функция

strlen (s:string):int

возвращает длину строки аргумента (от 0) или пустое значение (_).

Процедура

str_concat (v:string, s1:string, s2:string)

соединяет строки s1, s2 и записывает результат в переменную v. Предыдущее значение v теряется.

Процедура

str_select (v:string, s:string, n1:int, n2:int)

выделяет из строки s подстроку, начинающуюся с индекса n1 длиной n2 и записывает в переменную v. Если n2 больше длины строки s или имеет пустое значение, то в v переноситься остаток строки. Предыдущее значение v теряется. Нумерация символов в строке s (значения индекса n1) начинается с 0.

Функция

strfind (s:string,ss:string):int

определяет индекс первого вхождения подстроки ss в строку s (0, ..., strlen(s)-1). Функция возвращает пустое значение, если такой подстроки нет.

Функция

strtolower (s:string):string

преобразует все прописные буквы, встречающиеся в строке s, в строчные буквы, возвращая в качестве своего значения преобразованную строку.

Функция

strdelsp (s:string):string

удаляет пробелы в начале и конце строки, являющейся значением переменной s, и возвращает результат в виде своего значения.

Функция

strrfind (s:string, ss:string):int

осуществляет поиск подстроки ss в строке s, возвращая в качестве своего значения индекс последнего вхождения подстроки ss в строке s в диапазоне 0,...,strlen(s)-1 или пустое значение, если такой подстроки не найдено. В отличие от функции strfind, функция strrfind определяет последнее, а не первое вхождение подстроки.

6.9. Выполнение команд операционной системы

Процедура

system (s:string)

выполнение команды операционной системы, заданной в строке s.

6.10. Функции и процедуры работы с сетью

6.10.1. Функции и процедуры обмена наборами фактов

Обмен наборами фактов между процессами осуществляется по сетевым каналам типа stream, для которых требуется установление прямой связи, с использованием следующих функций и процедур языка ABIS.

Функция

s_openr(locname:string, callname:string):int

открывает канал для приема от других процессов запросов на установление связи. Параметры locname и callname определяют сетевые имена, связанные с именем устройства и номером порта через файлы конфигурации сети. В качестве своего значения функция возвращает номер открытого канала при нормальном завершении или значение -1 при выходе с ошибкой.

Функция

n_accept(chan:int, flag:int):int

принимает по каналу с номером chan запрос на установление связи и открывает канал для приема наборов фактов от процесса, приславшего запрос (канал с номером chan должен быть предварительно открыт функцией s_openr). Входной параметр flag устанавливает режим приема запросов:

0 - ожидание запроса, если он еще не получен;

1 - прием полученного запроса или выход без ожидания при отсутствии запроса.

В качестве своего значения функция n_accept возвращает:

номер открытого канала при нормальном завершении;

0 при flag=1 и отсутствии запроса;

-1 при выходе с ошибкой.

Процедура

n_listen(chan:int, locname:string, callname:string)

открывает канал для приема наборов фактов от другого процесса (эквивалентна последовательному вызову функций s_openr и n_accept при flag=0). Входные параметры locname и callname определяют сетевые имена, связанные с именем устройства и номером

порта через файлы конфигурации. При нормальном завершении процедуры номер открытого канала присваивается выходному параметру chan. Процедуру выполняется с ожиданием поступления запроса на установление связи от другого процесса.

Функция

n_connect(locname:string, callname:string):int

посылает другому процессу запрос на установление связи для передачи наборов фактов. Параметры locname и callname определяют сетевые имена, связанные с именем устройства и номером порта через файлы конфигурации. В качестве своего значения функция возвращает номер открытого канала, если запрос принят, или значение -1, если запрос не принят или обнаружена ошибка (функция выполняется без ожидания приема запроса другим процессом).

Процедура

n_call (chan:int, locname:string, callname:string)

открывает канал для передачи наборов фактов другому процессу, посылая запросы на установление связи до тех пор, пока связь не будет установлена (процедура выполняется с ожиданием приема запроса другим процессом). Входные параметры locname и callname определяют сетевые имена, связанные с именем устройства и номером порта через файлы конфигурации. При нормальном завершении процедуры номер открытого канала присваивается выходному параметру chan.

Процедура

n_send (chan:int, set:factset)

посылает набор фактов set другому процессу по каналу с номером chan, предварительно открытому функцией n_connect или процедурой n_call. Процедура выполняется с ожиданием окончания передачи всего набора фактов или обнаружения ошибки. Перед передачей в наборе фактов формируется или корректируется факт отношения Set_descriptor.

Процедура

n_receive (chan:int, set:factset)

принимает набор фактов от другого процесса по каналу с номером chan, предварительно открытому функцией n_accept или процедурой n_listen. Принятые факты записываются в набор фактов set. Процедура выполняется с ожиданием поступления набора фактов.

Процедура

nw_receive (chan:int, set:factset)

устанавливает режим ожидания приема набора фактов от другого процесса по каналу с номером chan, предварительно открытому функцией n_accept или процедурой n_listen. Прием осуществляется асинхронно, т.е. после вызова процедуры nw_receive выполнение ABIS-программы продолжается независимо от поступления данных. Принятые факты записываются в набор фактов set.

Процедура

n_done (chan:int)

проверяет поступление набора фактов от другого процесса по каналу с номером chan, для которого предварительно с использованием процедуры nw_receive был установлен режим ожидания приема. Процедура возвращает SUCCESS, если данные получены, или FAIL, если данные еще не получены или обнаружена ошибка.

Функция

nf_done (chan:int):int

проверяет поступление набора фактов от другого процесса по каналу с номером chan, для которого предварительно с использованием процедуры nw_receive был установлен режим ожидания приема.

В качестве своего значения функция возвращает:

1 , если данные получены;

0 , если данные еще не получены;

-1, если обнаружена ошибка.

В отличие от процедуры n_done, функция nf_done позволяет отличать случай, когда данные еще не получены, от случая, когда обнаружена ошибка.

Функция

is_read ():int

ожидает поступления данных или запросов на установление связи по каким-либо из каналов, открытым для приема функциями s_openr, n_accept или процедурой n_listen. В качестве своего значения функция возвращает количество каналов, по которым поступили данные или запросы. При обнаружении ошибки функция возвращает значение -1.

Процедура

n_shutdown (chan:int)

закрывает канал с номером chan, открытый функцией n_accept для приема наборов фактов.

Процедура

n_hangup (chan:int)

закрывает канал с номером chan, открытый функцией n_connect или процедурой n_call для передачи наборов фактов или открытый процедурой n_listen для приема наборов фактов.

Процедура

s_hangup (chan:int)

закрывает канал с номером chan, открытый функцией s_openr для приема запросов на установление связи.

6.10.2. Функции и процедуры обмена строками

Обмен строками между процессами осуществляется по сетевым каналам типа stream с использованием следующих функций и процедур языка ABIS.

Функция

s_accept (chan:int, flag:int):int

принимает по каналу с номером chan запрос на установление связи и открывает канал для приема строк или массивов чисел от другого процесса. Канал с номером chan должен быть предварительно открыт функцией s_openr (раздел 6.10.1). Входной параметр flag устанавливает режим приема запросов:

0 - ожидание запроса, если он еще не получен;

1 - прием полученного запроса или выход без ожидания при отсутствии запроса.

В качестве своего значения функция s_accept возвращает:

номер открытого канала при нормальном завершении;

0 при flag=1 и отсутствии запроса;

-1 - при обнаружении ошибки.

Функция

s_listen (locname:string, callname:string):int

открывает канал для приема строк или массивов чисел от другого процесса (эквивалентна последовательному вызову функций s_openr и s_accept при flag=0). Входные параметры locname и callname определяют сетевые имена, связанные с именем устройства и номером порта через файлы конфигурации. В качестве своего значения функция возвращает номер открытого канала при нормальном завершении или значение -1 при обнаружении ошибки.

Функция выполняется с ожиданием поступления запроса на установление связи от другого процесса.

Функция

s_connect (locname:string, callname:string):int

посылает другому процессу запрос на установление связи для передачи строк или массивов чисел. Параметры locname и callname определяют сетевые имена, связанные с именем устройства и номером порта через файлы конфигурации. В качестве своего значения функция возвращает номер открытого канала, если запрос принят, или значение -1, если запрос не принят или обнаружена ошибка (функция выполняется без ожидания приема запроса другим процессом).

Функция

s_call (locname:string, callname:string):int

открывает канал для передачи строк другому процессу, посылая запросы на установление связи до тех пор, пока связь не будет установлена (функция выполняется с ожиданием приема запроса другим процессом). Входные параметры locname и callname определяют сетевые имена, связанные с именем устройства и номером порта через файлы конфигурации. В качестве своего значения функция возвращает номер открытого канала при нормальном завершении или значение -1 при обнаружении ошибки.

Процедура

s_send (chan:int, s:string)

посылает строку s другому процессу по каналу с номером chan, предварительно открытому функцией s_connect или s_call. Процедура выполняется с ожиданием окончания передачи всей стоки или обнаружения ошибки.

Процедура

s_receive (chan:int, flag:int, s:string)

принимает по каналу с номером chan (предварительно открытому функцией s_accept или s_listen) строку, присваивая ее значение параметру s. Значение параметра flag определяет режим приема:

0 - с ожиданием поступления данных;

1 - без ожидания поступления данных.

При flag=1 процедура возвращает SUCCESS, если данные получены, или FAIL, если данные еще не получены или обнаружена ошибка.

Процедура

s_sendb (chan:int, addr:int, nelem:int)

посылает первые nelem элементов массива чисел с адресом addr другому процессу по каналу с номером chan, открытому функцией s_connect или s_call. Массив должен быть предварительно сформирован с использованием функций malloci, mwritei или функций mallocf, mwritef. Если значение параметра nelem не определено или превышает число элементов в массиве, то процедура посылает весь массив. Процедура выполняется с ожиданием окончания передачи всего массива или обнаружения ошибки.

Процедура

s_receiveb (chan:int, flag:int, addr:int)

принимает по каналу с номером chan (предварительно открытому функцией s_accept или s_listen) массив чисел и записывает его содержимое в массив с адресом addr, созданный функцией malloci или mallocf. Значение параметра flag определяет режим приема:

0 - с ожиданием поступления данных;

1 - без ожидания поступления данных.

При flag=1 процедура возвращает SUCCESS, если данные получены, или FAIL, если данные еще не получены или обнаружена ошибка.

Число принятых чисел ограничивается сверху числом элементов в массиве с адресом addr.

Функция

sf_receive (chan:int, s:string):int

проверяет поступление строки от другого процесса по каналу с номером chan, предварительно открытому функцией s_accept или s_listen. Если строка получена, то ее содержимое присваивается выходному параметру s. Функция выполняется без ожидания поступления данных и возвращает значения:

число полученных байт;

0 , если данные еще не получены;

-1, если обнаружена ошибка.

В отличие от процедуры s_receive, вызываемой при flag=1, функция sf_receive позволяет отличать случай, когда данные еще не получены, от случая, когда обнаружена ошибка.

Функция

sf_receiveb (chan:int, addr:int):int

проверяет поступление массива чисел от другого процесса по каналу с номером chan, предварительно открытому функцией s_accept или s_listen. Если массив получен, то его содержимое записывается в массив с адресом addr, созданный функцией malloci или mallocf. Функция выполняется без ожидания поступления данных и возвращает значения:

число полученных байт;

0 , если данные еще не получены;

-1, если обнаружена ошибка.

В отличие от процедуры s_receiveb, вызываемой при flag=1, функция sf_receiveb позволяет отличать случай, когда данные еще не получены, от случая, когда обнаружена ошибка. Число принятых чисел ограничивается сверху числом элементов в массиве с адресом addr.

В число каналов, по которым функция is_read() (раздел 6.10.1) ожидает поступления данных, входят каналы, открытые функциями s_accept, s_listen.

Процедура

s_shutdown (chan:int)

закрывает канал с номером chan, открытый функцией s_accept для приема строк или массивов чисел.

Каналы, открытые функциями s_connect, s_call или s_listen, закрываются процедурой s_hangup (раздел 6.10.1).

6.10.3. Функции и процедуры обмена посылками типа datagram

При обмене посылками типа datagram установление прямой связи между процессами не требуется, что позволяет принимать по одному каналу посылки от нескольких процессов. Однако, каналы этого типа имеют меньшую надежность, по сравнению с каналами типа stream. Для обмена данными по каналам типа datagram используются следующие функции и процедуры языка ABIS.

Функция

d_listen (namedg:string):int

открывает канал для приема посылок типа datagram от других процессов. Входной параметр namedg имеет значение сетевого имени, связанного с именем устройства и номером порта через файлы конфигурации сети. В качестве своего значения функция возвращает

номер открытого канала при нормальном завершении или значение -1 при обнаружении ошибки.

Функция

d_call ():int

открывает канал для передачи посылок типа datagram другим процессам, возвращая номер этого канала при успешном завершении или значение -1 при обнаружении ошибки.

Функция

d_addname (namedg:string):int

регистрирует удаленный порт для передачи по нему посылок типа datagram. Входной параметр namedg имеет значение сетевого имени, связаного с именем устройства и номером порта через файлы конфигурации сети. При нормальном завершении ссылочный номер порта возвращается в качестве значения функции. Этот ссылочный номер используется затем в качестве параметра процедуры d_send. При обнаружении ошибки функция возвращает значение -1.

Процедура

d_send (chan:int, proc:int, s:string)

передает по каналу с номером chan (предварительно открытому функцией d_call) посылку типа datagram в виде содержания строки s. Посылка передается в удаленный порт со ссылочным номером proc, предварительно зарегистрированный функцией d_addname.

Процедура выполняется с ожиданием окончания передачи всей посылки и обнаружения ошибки.

Процедура

d_receive (chan:int, flag:int, s:string)

принимает посылку типа datagram по каналу с номером chan (предварительно открытому функцией d_listen). Принятая посылка возвращается в виде значения строки s. Параметр flag устанавливает режим приема посылки:

0 - ожидание поступления посылки, если она еще не получена;

1 - прием полученной посылки или выход без ожидания приема, если посылка еще не получена.

При flag=1 процедура возвращает SUCCESS, если посылка получена, или FAIL, если посылка еще не получена или обнаружена ошибка.

Процедура

d_hangup (chan:int)

закрывает канал с номером chan, открытый функцией d_listen для приема посылок или функцией d_call для передачи посылок.

Процедура

d_delname (proc:int)

удаляет ссылочный номер порта proc, зарегистрированный функцией d_addname.

6.10.4. Функции для файлов конфигурации сети

Функция

s_getaddr (locname:string, callname:string):string

осуществляет поиск записей с заданными сетевыми именами местного и удаленного портов locname и callname в файлах конфигурации "nettcp.cfg", "nettcp.ini" (см. описание сетевой библиотеки). В качестве своего значения функция возвращает строку, в которой записаны имя устройства и текущее значение номера порта, разделенные пробелом. Если искомые записи не найдены ни в одном из файлов конфигурации, то функция возвращает пустую строку.

Функция

s_putaddr (line:string):int

вносит или корректирует записи в файле конфигурации "nettcp.ini". Параметр line, содержит одну или несколько записей, разделенных символом новой строки '\n'. При этом, каждая запись содержит: имя местного порта locname, имя удаленного порта callname, имя устройства и текущее значение номера порта, разделенные пробелами. Если в файле "nettcp.ini" уже была запись с теми же именами locname и callname, то она заменяется записью из строки line. Функция возвращает значение 0 при нормальном завершении или значение -1 при обнаружении ошибки.

6.11. Представление имен отношений, значений их атрибутов и имен наборов фактов в виде строк

Для некоторых приложений языка ABIS кортеж любого отношения удобно описывать с помощью отношения вида:

rel_str (Rel_name:string, Attr_number:int, Attr1:string,

..., AttrN:string),

где Rel_name - имя исходного отношения;

Attr_number - число атрибутов в исходном отношении;

Attr1, ..., AttrN - переменные для значений атрибутов исходного отношения. При этом используются n первых атрибутов Attr1, ..., Attrn, где n - значение атрибута Atr_number (n меньше или равно N).

Процедура

to_rstr (set1:factset, set2:factset)

представляет факты из набора set1 в виде кортежей отношения rel_str и записывает их в набор set2 (процедура unite с дополнительным преобразованием фактов).

Процедура

from_rstr (set1:factset, set2:factset)

преобразует факты из набора set1, представленные в виде кортежей отношения rel_str, к их исходному виду и записывает эти факты в набор set2 (процедура unite с дополнительным преобразованием фактов).

Процедура

attr_rstr (s:string, f:fact, i:int)

присваивает переменной s значение атрибута с номером i в кортеже отношения rel_str, описываемом фактом f. Параметр i определяет номер запрашиваемого атрибута Attri в ряду

Attr1,...,AttrN.

Для успешного выполнения процедур to_rstr, from_rstr, attr_rstr раздел описания отношений программы должен включать отношение rel_str. Имя rel_str включено в состав ключевых слов языка ABIS. Имена (но не типы) атрибутов этого отношения могут быть любыми. Число атрибутов равно N+2, где N должно быть не меньше максимально возможного числа атрибутов в любом другом отношении.

Процедура

import_rstr (filename:string, setname:string)

загружает в виде исходных текстов факты из файла с именем filename, представляет их в виде кортежей отношения rel_str и записывает их в набор фактов, имя которого задано строкой setname (при этом название набора фактов, указанное в файле игнорируется).

Процедура

export_rstr (filename:string, setname1:string, setname2:string)

преобразует факты, представленные в виде кортежей отношения rel_str, из набора, имя которого задано строкой setname1, к их исходному виду и записывает их в файл с

именем filename в виде исходных текстов, вставляя в начало файла ключевое слово factset и заканчивающееся двоеточием имя, содержащееся в строке setname2 (в общем случае имена setname1 и setname2 могут быть различными).

6.12. Операции со ссылками на переменные

Функция

nametovar (s:string, var):int

присваивает переменной var произвольного типа значение другой переменной, имя которой задано строкой s, а тип должен совпадать с типом переменной var. Функция возвращает значение 0 при нормальном завершении или значение -1 при выходе с ошибкой.

Функция

vartoname (var, s:string):int

преобразует имя глобальной переменной var произвольного типа в строковую константу, возвращаемую в виде значения параметра s. Функция возвращает значение 0 при нормальном завершении или значение -1 при ошибке.

Функция

nametoref (s:string, ref:int):int

преобразует имя переменной, заданное значением строки s, в ссылку на эту переменную (адрес этой переменной в памяти). Функция возвращает значение 0 при нормальном завершении или значение -1 при ошибке.

Функция

vartoref (var):int

возвращает в качестве своего значения ссылку на переменную var любого типа (адрес этой переменной в памяти).

Функция

reftovar (ref:int, var):int

присваивает переменной var произвольного типа значение другой переменной, ссылку на которую задает значение параметра ref (типы переменных должны совпадать). Функция возвращает в качестве своего значения ссылку на переменную var.

Процедура

copyr (ref1:int, ref2:int)

копирует набор фактов, описываемый переменной со ссылкой ref1, в набор фактов, описываемый переменной со ссылкой ref2.

Процедура

copyr1 (ref:int, set:factset)

копирует набор фактов, описываемый переменной со ссылкой ref, в набор фактов, описываемый переменной set.

Процедура

copyr2 (set:factset, ref:int)

копирует набор фактов, описываемый переменной set, в набор фактов, описываемый переменной со ссылкой ref.

Процедура

clearr (ref:int)

удаляет все факты из набора, описываемого переменной со ссылкой ref.

6.13. Функции опроса

Функция

isempty (x:t):log

возвращает SUCCESS, если x имеет пустое значение, иначе FALSE. t - любой тип, определенный в языке.

Функция

factcount (x:factset):int

возвращает число фактов в наборе x (для пустого набора 0).

Функция

ftrust (x:fact):float

возвращает достоверность факта (от 0.0 до 1.0).

Функция

freememory (x:int):long

возвращает размер текущей свободной памяти в байтах при работе в операционной системе MS-DOS. При этом необходимо установить x = 0.

Функция

usedmemory (x:int):long

возвращает размер текущей занятой памяти в байтах, включая программу. При этом необходимо установить x = 0.

Функция

changes ():int

возвращает число изменений, произведенных правилами текущего модуля в текущем проходе до момента выполнения этой функции. Изменением считается добавление нового факта в рабочую область.

Функция

rulenum ():int

возвращает номер текущего правила (нумерация правил в модуле начинается с 1). Может использоваться для облегчения явных переходов по правилам модуля оператором continue.

6.14. Операции со временем

Функция

systime ():long

возвращает системное время компьютера в секундах с 0 часов 1 января 1970 года по Гринвичу.

Процедура

fsystime (time:long, mtime:int)

выдает системное время компьютера, присваивая переменной time время в секундах с 0 часов 1 января 1970 года по Гринвичу, и переменной mtime время в милисекундах.

Функция

ctime (time:long):string

преобразует значение времени time в секундах, отсчитываемого с 0 часов 1 января 1970 года по Гринвичу, в строку вида

часы:минуты:секунды .

Функция

convtime (time:long):string

преобразует значение времени time в секундах, отсчитываемого с 0 часов 1 января 1970 года по Гринвичу, в строку вида

часы:минуты:секунды день-месяц-год .

Функция

sleep (time:int):int

вызывает задержку выполнения ABIS-программы на time секунд. Задержка отсчитывается по системному таймеру. Функция sleep возвращает разницу между заданной задержкой и временем выполнения программы за этот период, которые могут не совпадать из-за выполнения других программ в операционной среде.

6.15. Операции с массивами чисел

Функция

malloci (number:int):int

создает массив из number чисел типа integer, возвращая в качестве своего значения адрес этого массива (адрес первого элемента массива) в оперативной памяти. При ошибке функция возвращает значение 0.

Функция

mallocf (number:int):int

создает массив из number чисел типа float, возвращая в качестве своего значения адрес этого массива (адрес первого элемента массива) в оперативной памяти. При ошибке функция возвращает значение 0.

Процедура

mwritei (addr:int, n:int, value:int)

записывает значение параметра value типа int в качестве элемента с номером n в массив с адресом addr, который должен быть предварительно создан функцией malloci. Нумерация элементов массива начинается с нуля (значение параметра n должно быть меньше, чем значение параметра number при вызове функции malloci).

Процедура

mwritef (addr:int, n:int, value:float)

записывает значение параметра value типа float в качестве элемента с номером n в массив с адресом addr, который должен быть предварительно создан функцией mallocf. Нумерация элементов массива начинается с нуля (значение параметра n должно быть меньше, чем значение параметра number при вызове функции mallocf).

Функция

mreadi (addr:int, n:int):int

возвращает значение типа int для элемента с номером n из массива с адресом addr, который должен быть предварительно создан функцией malloci. Нумерация элементов массива начинается с нуля (значение параметра n должно быть меньше, чем значение параметра number при вызове функции malloci).

Функция

mreadf (addr:int, n:int):float

возвращает значение типа float для элемента с номером n из массива с адресом addr, который должен быть предварительно создан функцией mallocf. Нумерация элементов массива начинается с нуля (значение параметра n должно быть меньше, чем значение параметра number при вызове функции mallocf).

Процедура

mfree (addr:int)

освобождает оперативную память, занятую массивом с адресом addr при его создании с использованием функции malloci или mallocf.

Функция

arraynum (addr:int):int

возвращает число элементов в массиве чисел с адресом addr, созданном функцией malloci или mallocf.

Процедура

arr_select (addr:int, addr0:int, n:int, k:int)

копирует k элементов из массива с адресом addr0, начиная с n-го элемента, в первые k элементов массива с адресом addr (нумерация элементов в массивах начинается с нуля). Массивы с адресами addr, addr0 должны быть предварительно созданы функцией malloci или mallocf. Если типы чисел в массивах различны, то при копировании выполняется преобразование типов. Если значение параметра k не определено или сумма значений параметров n и k превышает число элементов в массиве с адресом addr0, то копируются все элементы, начиная с n-го элемента. При выполнении процедуры число копируемых элементов ограничивается сверху числом элементов в массиве с адресом addr.

Функция

string (addr):string

для аргумента addr типа int копирует содержимое массива в двоичном формате с адресом addr в строку, возвращаемую в качестве значения функции.

6.16. Процедуры управления трассировкой

Процедуры

trace_on,

trace_off

включают и выключают режим трассировки выполнения модулей; соответствуют опции /p исполняющей системы (раздел 7.1). Позволяют более точно управлять процессом отладки программы.

7. ВЫЗОВ И ЗАГРУЗКА

7.1. Компиляция и выполнение программ

Список исходных или объектных файлов ABIS-программы задается в виде текстового файла, называемого далее make-файл (раздел 7.2). Загрузка ABIS-программы для компиляции или выполнения осуществляется под управлением информации, заданной в make-файле. Исполняющая система языка ABIS запускается командой:

abx2 [options] filename

где: filename – имя make-файла;

options - любая комбинация из:

/c	- компиляция исходных файлов, заданных в make-файле, при этом остальные опции игнорируются;
/t	- включение трассировки операций обмена данными, при этом на дисплей выводятся сообщения о начале и конце выполнения обменов по сети и обменах с файлами;
/l	- включение полной трассировки обмена по сети – запись всех передаваемых и принимаемых наборов в исходном виде в файлы;
/p	- включение трассировки программы: на дисплей выводятся сообщения о выполняемом модуле и текущем номере правила;
/s	- запуск отладчика программ языка ABIS (раздел 8);
/f	- установка записи в файлы в режиме MS-DOS (раздел 6.5.2);
/u	- установка различия прописных и строчных букв в именах файлов в среде UNIX (раздел 6.5.1);
/b	- сокращенная загрузка ABIS-программы (без таблиц имен переменных и отношений);
/i=filename	- переадресация стандартного ввода данных, осуществляемого процедурой gets, на файл с именем filename;

/o=filename - переадресация стандартного вывода данных, осуществляемого процедурами puts, putv, printf, на файл с именем filename.

/h - выдача на экран информации об опциях.

Опция /p, используемая совместно с опциями /c или /r, устанавливает режим компиляции с трассировкой - выдачей информации о номере текущей строки в файле.

7.2. Структура make-файла

Каждая строка make-файла содержит имя одного исходного или объектного файла. Содержащие ABIS-программу файлы, имена которых перечисляются в make-файлах, могут находиться в различных директориях. При этом для файлов, не находящихся в текущей директории, необходимо указывать полное имя, включающее путь к этому файлу.

Строка, начинающаяся с символа (#), обрабатывается как строка комментариев:

Описание модели должно содержаться в одном файле, который должен стоять первым в make-файле. Следующим должен стоять файл, содержащий модуль start.

Остальные файлы, содержащие модули правил и наборы фактов, могут стоять в произвольном порядке, но этот порядок должен быть один при компиляции и загрузке.

Если файл правил не резидентный, то перед ним должен стоять символ минус, при этом файл будет компилироваться, но загружаться будет только при явном обращении к одному из модулей этого файла.

Минимальное число файлов в программе - 2 (один с описанием, другой со стартовым модулем.

Существует два вида make-файла: список исходных файлов и список объектных файлов.

Список исходных файлов используется компилятором, или загрузчиком. Имена исходных файлов могут быть произвольными, но нельзя использовать стандартные расширения .dcl, rul. Рекомендуются следующие традиционные расширения имен: .d для файла описания, .r для файла правил и .bd для файла, содержащего наборы фактов.

Имена объектных файлов порождаются из имен соответствующих исходных файлов, но с добавлением расширений .dcl (для файла описаний) или .rul (для файлов модулей правил и/или наборов фактов).

Список объектных файлов содержит файлы расширениями dcl и rul. Файл с расширением dcl и первый файл с расширением rul (содержащий модуль start) загружаются всегда, остальные файлы из списка - только если они резидентные (перед именем файла не стоит знак минус). Выполняется загрузка только резидентных файлов; однако описание загружается для всех модулей (в том числе и из нерезидентных файлов). Порядок и состав файлов должен быть такой же, как и в списке исходных файлов - все нерезидентные файлы должны также перечисляться, хотя и не грузятся.

Примеры структуры make-файлов:

файл для компиляции comp.mak:

```
# исходный файл описания
 xmodel.d
# исходный файл, содержащий модуль start
 xstart.r
# другие исходные файлы
 work1.r
 work2.r
 overlay1.r
 overlay2.r
 overlay3.r
 work3.r
 overlay4.r
 work4.r
 overlay5.r
 databas1.db
 databas2.db
```

файл для загрузки: load.mak

```
# объектный файл описания
 xmodel.dcl
# объектный файл, содержащий модуль start
 xstart.rul
# резидентные объектные файлы
 work1.rul
```

work2.rul

транзиентные объектные файлы помечаются минусом,

это распределение может быть разным для

разных загрузок

- overlay1.rul

- overlay2.rul

overlay3.rul

work3.rul

overlay4.rul

work4.rul

overlay5.rul

- databas1.rul

databas2.rul

8. СРЕДСТВА ОТЛАДКИ ПРОГРАММЫ

8.1. Отладчик программ языка ABIS

При запуске ABIS-программы на выполнение командой abx2 с опцией /s (раздел 7.1) пользователь входит в среду отладчика языка ABIS со своим набором команд, для ввода которых на экране появляется командная строка с подсказкой вида

abx> .

После выполнения команды, не связанной с началом или продолжением выполнения программы (раздел 8.2), на экране вновь появляется подсказка среды отладчика для ввода следующей команды.

Выполнение программы начинается или продолжается после ввода одной из следующих команд (раздел 8.2): run, deb[ug], modul[e], rul[e], Rul[e], stat[ment], fact. При достижении очередной контрольной точки выполнение программы приостанавливается до получения управляющего сигнала с клавиатуры. При нажатии клавиши "Enter" выполнение программы продолжается в прежнем режиме. При последовательном нажатии клавиш "q" и "Enter" на экране появляется подсказка для ввода команды отладчика, с помощью которой можно изменить режим выполнения программы, посмотреть текущие значения переменных, а затем продолжить выполнение программы по одной из перечисленных выше команд или закончить выполнение программы по команде stop.

Для выхода из среды отладчика необходимо ввести команду

abx> quit .

8.2. Команды отладчика

Команды отладчика языка ABIS имеют полную и сокращенную формы записи. Далее при описании команд необязательные части ключевых слов, которые могут быть опущены в сокращенной форме записи, заключены в квадратные скобки [].

Выполнение ABIS-программы начинается или продолжается по одной из следующих команд:

run - выполнение программы без отладчика - без учета контрольных точек;

deb[ug] - выполнение программы с отладчиком - с остановками в контрольных точках, предварительно определенных командой bp[oint] или bv[ariable];

modul[e] - пошаговое выполнение программы с остановками при начале выполнения очередного модуля и выходе из него;

rul[e] - пошаговое выполнение программы с остановками при начале выполнения очередного правила;

Rul[e] - пошаговое выполнение программы с остановками при начале выполнения очередного правила и его окончании с выводом на экран информации о согласованных и порожденных фактах;

stat[ment] - пошаговое выполнение программы с остановками после выполнения очередного предложения и выводом на экран информации о согласуемых и порождаемых фактах;

fact - пошаговое выполнение программы с остановками после выполнения очередного предложения или выбора из базы данных очередного факта при выполнении оператора согласования с выводом на экран информации о выбираемых и порождемых фактах.

При достижении очередной контрольной точки на экран выдается текст текущего правила. Если в правиле определено текущее предложение, то оно выделяется соседними строками, состоящими из символов '-'.

Команда

bp[oint] module

устанавливает контрольную точку по имени модуля, которое задается значением параметра module. Если вместо имени модуля ввести ключевое слово NONE, то установленная прежде контрольная точка отменяется.

Команда

bv[ariable] factset

устанавливает контрольную точку по имени описывающей набор фактов переменной, которая задается значением параметра factset. Если вместо имени переменной ввести ключевое слово NONE, то установленная прежде контрольная точка отменяется.

Команда

ty[pe] variable

выводит на экран текущее значение переменной, имя которой задано значением параметра variable.

Команда

vi[ew] factset

выводит на экран текущее содержимое набора фактов по имени описывающей его переменной, которая задается значением параметра factset.

Команда

exp[ort] factset [filename]

записывает в файл содержимое набора фактов в виде исходных текстов. Переменная, описывающая набор фактов, задается значением параметра factset, имя файла - значением параметра filename. Если параметр filename отсутствует, то имя файла, создаваемого в текущей директории, образуется из имени переменной и расширения ".log".

По команде

lis[t] parameter

на экран выдается один из следующих списков, определяемый именем параметра parameter:

mod[ule] - список имен модулей;

gvar[iable] - список глобальных переменных с указанием их имен, типов и текущих значений;

lvar[iable] - список локальных переменных с указанием их имен, типов и текущих значений;

pvar[iable] - список частных переменных с указанием их имен, типов и текущих значений.

Режимы трассировки программы изменяются с помощью следующих команд, которые в зависимости от значения параметра on или off соответственно устанавливают или отменяют один из режимов :

progtr[ace] on|off - изменение режима сокращенной трассировки выполнения модулей, предусматривающего выдачу на экран сообщений о начале выполнения очередного модуля и выходе из него (по умолчанию этот режим установлен);

tr[ace] on|off - изменение режима трассировки операций обмена данными, предусматривающего выдачу на экран сообщений о сетевом взаимодействии, а также считыванию из файлов и записи в файлы (по умолчанию этот режим установлен);

modtr[ace] on|off - изменение режима полной трассировки выполнения модулей, предусматривающего выдачу на экран сообщений о выполнении каждого правила (по умолчанию этот режим не установлен);

nettr[ace] on|off - изменение режима трассировки сетевого обмена данными на нижнем уровне, предусматривающего выдачу на экран сообщений о приеме и передаче физических посылок (по умолчанию этот режим не установлен);

netlog[out] on|off - изменение режима трассировки сетевого обмена данными с записью принимаемых и посылаемых наборов фактов в файлы в виде исходных текстов (по умолчанию этот режим не установлен.

Отличительным признаком сообщений при трассировке модулей служит символ '#' в начале строки. После имени модуля и разделяющего символа '#' указывается номер текущего правила в модуле и после точки - номер текущего предложения в правиле.

Отличительным признаком сообщений при трассировке операций обмена данными служит символ ':' в начале строки.

При трассировке сетевого обмена данными с записью в файлы принимаемых и посылаемых наборов фактов имена файлов образуются из ключевого слова (rcv - для принимаемых и snd - для посылаемых данных), номера сетевого канала, разделяющего символа '_', порядкового номера посылки (по сквозной нумерации всех принимаемых и посылаемых наборов) и расширения ".log". Файлы создаются в текущей директории.

По команде

sh[ow] parameter

на экран выдается значение одного из параметров среды отладчика, заданного именем parameter из следующего списка:

bp[oint] - имя модуля, устанавливающего контрольную точку;

bv[ariable] - имя переменной, устанавливающей контрольную точку по набору фактов;

progtr[ace] - режим сокращенной трассировки модулей (on|off);

tr[ace] - режим трассировки обмена данными (on|off);

modtr[ace] - режим полной трассировки выполнения модулей (on|off);

nettr[ace] - режим трассировки сетевого обмена данными на нижнем уровне (on|off);

netlog[out] - режим трассировки сетевого обмена данными с с записью наборов фактов в файлы (on|off).

По команде time

на экран выдается информация о времени выполнения программы: общее время для всей программы и время выполнения отдельных модулей в абсолютном и процентном выражении.

По команде mem[ory]

на экран выдается информация об объеме оперативной памяти, используемой в данный момент программой.

По команде stop

прекращается выполнение программы.

По команде reload

происходит перезагрузка программы из объектных файлов (предварительно программа должна быть остановлена командой stop).

По команде quit

происходит выход из среды отладчика с одновременным окончанием выполнения программы.

9. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Язык ABIS входит в состав рабочего программного обеспечения (РПО) и конфигуратора системы верхнего блочного уровня (СВБУ) и используется на базе операционной системы LICS на рабочей станции и сервере, входящих в состав технических средств СВБУ.

Электронное научное издание

Бывайков Михаил Евгеньевич

«Язык ABIS. Описание языка»

В печать от 17.12.2012

Электронное издание комбинированного распространения

Электронно-оптический диск (CD-R), 0,6 Мб

Федеральное государственное бюджетное учреждение науки

Институт проблем управления им. В.А. Трапезникова

Российской академии наук

Федеральное государственное бюджетное учреждение науки

Институт проблем управления им. В.А. Трапезникова

Российской академии наук

117997,

ул. Профсоюзная, д. 65,

Россия, Москва