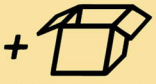
В. Н. Марков

Современное логическое программирование на языке Visual Prolog 7.5



Материалы на www.bhv.ru

bhv®

УДК 004.438 Visual Prolog
ББК 32.973.26-018.1
М26

Марков В. Н.

М26 Современное логическое программирование на языке Visual Prolog 7.5: учебник. — СПб.: БХВ-Петербург, 2016. — 544 с.: ил. — (Учебная литература для вузов)

ISBN 978-5-9775-3487-1

В учебнике излагается полный набор классических и новейших инструментов логического программирования, а также парадигмы функционального, обобщенного, императивного и объектно-ориентированного программирования, органически вошедшие в Visual Prolog 7.5. Рассматриваются основные способы представления и обработки графов, деревьев и массивов, инструменты профессионального программирования. Приводятся примеры разработки символьных преобразователей, калькуляторов, интерпретаторов языков программирования, игровых моделей и т. п. Книга содержит практикум по программированию и описание основных классов Visual Prolog. Учебник предназначен для изучения дисциплин «Логическое программирование» и «Функциональное и логическое программирование».

Для студентов, преподавателей
и разработчиков интеллектуальных информационных систем.

УДК 004.438 Visual Prolog
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*
Фото	*Кирилла Сергеева*

Подписано в печать 30.09.15.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 43,86.
Тираж 700 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

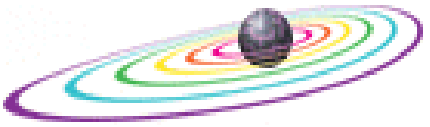
ISBN 978-5-9775-3487-1

Оглавление

Главы, помещенные в электронный архив

ЧАСТЬ VI. ПРОСТЫЕ ГРАФИЧЕСКИЕ ПРИЛОЖЕНИЯ

ГЛАВА 1



Лексика языка Visual Prolog

Лексика представляет собой описание неделимых сущностей языка программирования, из которых, как из кирпичиков, строятся все вычислительные конструкции этого языка. Неделимыми они являются потому, что воспринимаются компилятором языка как цельные фрагменты текста, хотя и состоят из последовательности символов алфавита. Такие неделимые сущности называются *лексемами*.

Некоторые лексемы — например, числа, ключевые слова и строки известны многим начинающим программистам. О других лексемах — двоичных данных, списках знают лишь опытные. В этой главе мы рассмотрим только лексемы Visual Prolog, а способы построения из них языковых конструкций будут описаны в следующих главах.

1.1. Алфавит

Алфавит Visual Prolog включает буквы английского и национального (русского) алфавитов, цифры, знаки пунктуации, операторы и ключевые слова. В языке есть различия между идентификаторами, начинающимися с заглавной и строчной буквы.

1.2. Комментарии

Для указания многострочных комментариев используются открывающие символы `/*` и закрывающие символы `*/`. Для указания однострочных комментариев служит знак процента `%`, действие которого распространяется до конца строки.

Пример комментариев:

```
/*   Многострочные
     комментарии     */
%    Однострочные комментарии
```

1.3. Символы разметки текста

Такими символами являются пробелы, табуляции и символы перехода на новую строку. Комментарии и символы разметки во время компиляции не анализируются.

1.4. Лексемы

В качестве лексем языка Visual Prolog выступают ключевые слова, знаки пунктуации, операции, идентификаторы и литералы.

1.4.1. Ключевые слова

Ключевые слова разделяются на старшие и младшие. Это деление условное и предназначено только для различения цвета ключевых слов при их отображении в редакторе.

К старшим ключевым словам относятся:

```
class          domains        inherits       predicates
clauses        end            interface      properties
constants      facts          monitor        resolve
constructors   goal           namespace      supports
delegate       implement      open
```

Младшие ключевые слова:

```
align      digits       failure      language     quot
and        div          finally      mod          rem
anyflow    do           foreach      multi        single
as         else         from         nondeterm    then
bitsize    elseif       guard        or           to
catch      erroneous    if           orelse       try
determ     externally   in           procedure
```

Все ключевые слова, кроме `as` и `language`, зарезервированы, и использовать их в качестве имен, вводимых программистом в исходный текст программы, не допускается. Ключевое слово `end` всегда комбинируется с другими ключевыми словами:

```
end class        end implement        end interface
end if           end foreach          end try
```

1.4.2. Знаки пунктуации

Знаками пунктуации являются:

```
;   !   ,   .   #   [   ]   |   (   )   {   }   :   :-   ::
```

Многосимвольные знаки `:-` и `::` не должны разделяться пробелами.

1.4.3. Операции

Операции определяют вычисления, которые должны быть выполнены над указанными данными. Все операции бинарные, но плюс и минус могут быть и унарными.

```
+    -    /    *    ^    =    div    mod    quot    rem
<    >    <>   ><   <=   >=   :=     ==
```

Не допускается разрывать пробелами многосимвольные имена операций:

```
<>    ><    <=    >=    :=    ==    div    mod    quot    rem
```

1.4.4. Идентификаторы

В языке различают четыре вида идентификаторов: строчные, заглавные, анонимные и эллипсис.

Строчный идентификатор — последовательность букв, цифр и знаков подчеркивания, начинающаяся со строчной буквы.

Примеры правильных строчных идентификаторов:

```
S       жук001       w_ю_       xMary       й9я8ы
```

Примеры неправильных строчных идентификаторов:

```
8ver       % начинается с цифры
soap#1     % содержит символ #
Soap       % начинается с заглавной буквы
my prog    % содержит пробел
facts      % ключевое слово
```

Заглавный идентификатор — последовательность букв, цифр и знаков подчеркивания, начинающаяся с заглавной буквы или со знака подчеркивания.

Примеры правильных заглавных идентификаторов:

```
S    Жук_001    _Mary    Facts
```

Примеры неправильных заглавных идентификаторов:

```
8ver       % начинается с цифры
Soap$1     % содержит символ $
soap       % начинается со строчной буквы
My prog    % содержит пробел
```

Анонимный идентификатор — знак подчеркивания: _.

Эллипсис — знак многоточия: ....

1.4.5. Литералы

В качестве литералов выступают целые и вещественные числа, символы, строки, двоичные данные, списки и составные термы.

Целые числа

Формат целых чисел зависит от системы счисления, в которой представлено число (табл. 1.1).

Таблица 1.1. *Представление десятичных, восьмеричных и шестнадцатеричных чисел*

Система счисления	Знак (опционально)	Префикс	Допустимые цифры	Примеры
Десятичное целое	±	Нет префикса	0123456789	355 +90 -897
Восьмеричное целое	±	0o	01234567	0o777 +0o121 -0o67
Шестнадцатеричное целое	±	0x	0123456789 AaBbCcDdEeFf	0xABC +0xc82 -0x3ff5

Вещественные числа

Вещественные числа представляются только в десятичной системе счисления (табл. 1.2, 1.3).

Таблица 1.2. *Представление вещественных чисел*

Мантисса				Порядок (опционально)		
Знак числа (опционально)	Цифры целой части	Десятичный разделитель	Цифры дробной части	Префикс показателя степени	Знак показателя степени	Цифры показателя степени
		опционально				
±	0123456789	.	0123456789	e или E	±	0123456789

Таблица 1.3. *Примеры вещественных чисел*

Число	Пояснение
-123.789	число без степени
+123.456E-5	$+123.456 \cdot 10^{-5}$
-0.123456e+12	$-0.123456 \cdot 10^{12}$

Символы

Символ — любой печатаемый на клавиатуре символ, заключенный в апострофы, или ESC-последовательность, также заключенная в апострофы. *ESC-последовательность* (ESCAPE-последовательность) — это последовательность управляющих символов, начинающаяся с обратной наклонной черты — знака шиллинга \ (иногда этот знак называют *обратным слешем*). Именно обратная наклонная черта и является признаком ESC-последовательности:

- `'\t'` — табуляция (tabulation);
- `'\n'` — переход на новую строку (new line);
- `'\r'` — возврат в начало строки (return);
- `'\uXXXX'` — Unicode-символ, имеющий шестнадцатеричный код `XXXX`.

Кодировку Unicode-символов можно посмотреть, например, в приложении MS Word, используя меню **Вставка** | **Символ**. ESC-последовательность может определять не только действия, но и символы. Примеры символов и ESC-последовательностей представлены в табл. 1.4.

Таблица 1.4. *Примеры символов*

Символ	Пояснение
`'f'`	Буква `f`
`'\u0066'`	Буква `f`, заданная двухбайтовым шестнадцатеричным кодом
`'Ж'`	Буква `Ж`
`'\u0416'`	Буква `Ж`, заданная двухбайтовым шестнадцатеричным кодом
`'\n'`	Переход на новую строку
`'\''`	Апостроф '
`'\\'`	Обратная наклонная черта \
`'\"'`	Двойная кавычка "
`'/'`	Наклонная черта /
`'\u2194'`	Символ ↔
`'\u203C'`	Символ ‼

С помощью Unicode-символов можно задавать символы, которых нет на клавиатуре.

Строки

Строка — один или более символов, заключенных в двойные кавычки. Строка не должна разрываться клавишей <Enter>. Строки могут содержать символы, представляемые ESC-последовательностями. Строки с префиксом `@` могут многократно разрываться клавишей <Enter> и, кроме того, выполнение ESC-последова-

тельностей в такой строке подавляется. В табл. 1.5 показаны особенности обработки строк с префиксом @.

Таблица 1.5. *Особенности обработки строк при наличии префикса* @

Строка	Вывод на экран
"Visual prolog"	Visual prolog
"Visual\tprolog"	Visual prolog
@"Visual prolog"	Visual prolog
@"Visual\tprolog"	Visual\tprolog
"Visual prolog"	Ошибка компиляции
@"Visual prolog"	Visual prolog

Префикс @ удобно использовать, например, при задании в программе пути к файлу, который обычно содержит несколько знаков обратной наклонной черты. С его помощью также удобно определять многострочный текст без многократного ввода ESC-последовательности \n.

Кроме того, префикс @ полезно применять для определения строк, заданных без кавычек и обрамленных парными символами: открывающим и закрывающим. В этом случае компилятор будет проверять наличие закрывающего символа, если был использован открывающий символ. В табл. 1.6 представлены допустимые открывающие и закрывающие символы.

Таблица 1.6. *Открывающие и закрывающие парные символы*

Парные символы		Парные символы	
Открывающий	**Закрывающий**	**Открывающий**	**Закрывающий**
@(	)	@)	(
@{	}	@}	{
@[	]	@]	[
@<	>	@>	<

Например, выровненный по центру абзац HTML-текста можно задать в виде строки:

```
Str = @[<p align="center">Некоторый текст</p>]
```

Двоичные данные

Двоичные данные — последовательность байтов, заключенная в квадратные скобки, перед которыми указывается знак $. Байты могут быть представлены десятич-

ными, восьмеричными или шестнадцатеричными целыми числами в диапазоне от 0 до 255, или арифметическими выражениями, значения которых может быть вычислено во время компиляции (табл. 1.7).

Таблица 1.7. *Примеры двоичных данных*

Двоичные данные	Вывод на экран
`$[10,20,30]`	`$[0A,14,1E]`
`$[10,0x20,0o30]`	`$[0A,20,18]`
`$[4+4,2*0x10]`	`$[08,20]`

Двоичные данные используются в Visual Prolog не только для связи с другими языками и для низкоуровневой обработки данных, но и в целях безопасного хранения и обмена данными произвольного типа, а также в качестве массива данных с произвольным доступом к элементам.

Списки

Синтаксически, список представляет собой заключенную в квадратные скобки последовательность элементов одного типа, разделенных запятой. Список может быть также выражен головой списка, роль которого играет первый элемент, и хвостом списка, отделенным от головы вертикальной чертой. Хвост списка является списком. Варианты записи списков приведены в табл. 1.8.

Таблица 1.8. *Варианты записи списков*

Список	Описание
`[]`	Пустой список
`[10,20,30]`	Список целых чисел
`["123","abcd"]`	Список строк
`[10,"abcd"]`	Неправильный список
`[3|W]`	Список, состоящий из головы — числа `3` и хвоста, выраженного переменной `W`
`[3|[45,12]]`	Список, состоящий из головы — числа `3` и хвоста, представленного списком `[45,12]`
`[[8,45],[],[90]]`	Список, состоящий из трех вложенных списков, один из которых пустой
`[[]]`	Непустой список, содержащий один пустой список

Составной терм

Составной терм определяется пользователем и служит для представления в программе структур данных. Составные термы описаны в *главе 2*.

ГЛАВА 2

Термы

2.1. Понятие термов

К *простым* термам относятся числа, символы, строки и идентификаторы, описанные в лексике языка.

Составные термы определяются через простые термы и составные термы, в том числе и через самих себя. Иерархия термов изображена на рис. 2.1.

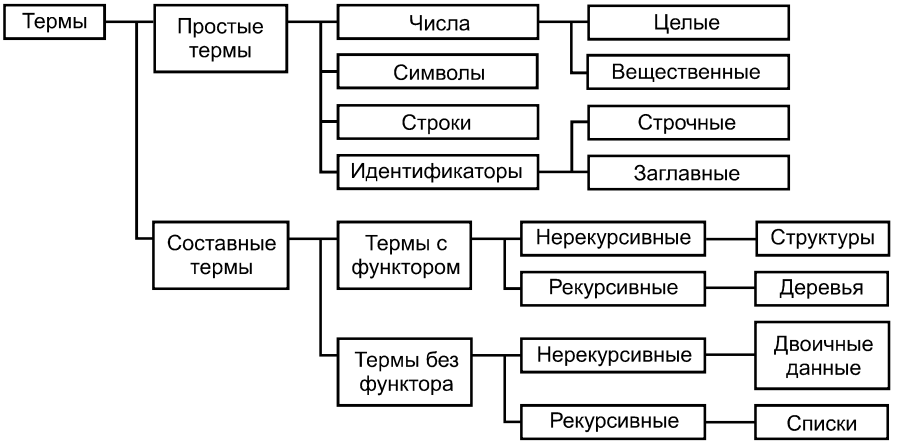


Рис. 2.1. Иерархия термов

Определение составного терма в общем случае рекурсивно. Согласно своему названию такой терм является именованной структурой, составленной из нескольких сущностей, как показано на рис. 2.2.

Имя терма называется *функтором* и обозначается строчным идентификатором. Количество аргументов называется *арностью*. Терм на рис. 2.2 является *N-арным* термом, иногда его называют *N-местным* термом, что есть то же самое. Каждый аргумент, в свою очередь, сам является простым или составным термом. Если один или несколько аргументов терма выражаются этим же термом, то говорят, что такой терм определен *рекурсивно*. В комментариях к логическим программам термы

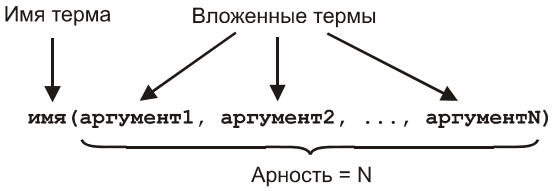


Рис. 2.2. Составной терм

иногда обозначают в виде имени и арности `имя/N`, опуская для краткости аргументы.

Составные термы могут не иметь аргументов. Такие термы содержат пустые скобки и называются *нульарными* термами.

Например, терм `f(999, arm)` имеет функтор `f`, является двухарным, первый аргумент — число `999`, второй аргумент — строчный идентификатор `arm`, терм определен нерекурсивно и имеет сокращенное обозначение `f/2`.

Терм `spice()` является нульарным термом и в документации имеет сокращенное обозначение `spice/0`.

Рекурсивный терм `z(babilon, z(inter,Near,8), 43)` является трехарным. Первый аргумент — строчный идентификатор `babilon`, второй аргумент — *вложенный* терм с этим же функтором и арностью, третий аргумент — число `43`. Вложенный терм `z(inter,Near,8)` содержит в качестве второго аргумента переменную `Near`, т. к. это заглавный идентификатор.

2.2. Переменные

Переменные в Прологе пишутся заглавными идентификаторами и могут быть либо свободными, либо связанными. Пока переменная *не связана* каким-либо значением, она называется *свободной*. Как только свободная переменная *связывается* каким-либо значением, она становится *связанной* и после этого не может изменить своего значения, будто является константой. Такой механизм связывания переменных называется *неразрушающим* присваиванием, в отличие от *разрушающего* присваивания, используемого в императивных языках и допускающего многократное изменение значения переменной.

Термин «переменная» в применении к Прологу достаточно условен и является данью традиции, привнесенной из императивных языков. Переменная в Прологе не меняет свое значение и, будучи свободной, она приобретает его только один раз. Чтобы связанная переменная приобрела другое значение, ее надо «отвязать» от текущего значения с помощью процесса, называемого *откатом*, т. е. произвести своеобразное «развоплощение». Механизм отката будет описан в *главе 6*. К настоящему времени в теории языков программирования не придуман такой термин, который бы адекватно отражал суть «переменных» Пролога с однократным связыванием. Поэтому мы будем использовать термин «переменная», надеясь на правильное его понимания читателем.

Термы, в которые не входят свободные переменные, называются *основными*. Термы, содержащие свободные переменные, называются *неосновными*.

Терм `f(999, arm)` является основным, а терм `z(inter,Near,8)` — неосновным, терм `z(babilon, z(inter,Near,8), 43)` также является неосновным, поскольку один из его аргументов — неосновной терм.

В Прологе используются и *анонимные* переменные — в тех случаях, где согласно синтаксису должна находиться переменная, но ее значение в вычислениях не используется. Анонимные переменные выражаются знаком подчеркивания или идентификатором, начинающимся со знака подчеркивания.

Мы рассмотрели *префиксные* термы, т. е. термы, функтор которых расположен слева от аргументов, перечисленных в скобках. В Прологе используются также *инфиксные* двухарные термы, у которых функтор располагается между аргументами и обозначается не строчным идентификатором, а символом арифметической или логической операции. Такие термы призваны повысить читабельность исходного текста программы. Например, вместо префиксного терма проверки неравенства `>(A,5)` в Прологе используют инфиксную запись этого терма `A>5`. Такая запись более привычна для человека и, несмотря на отсутствие скобочной нотации и строчного идентификатора в качестве функтора, является допустимым в Прологе термом.

2.3. Списки

Наряду с префиксными и инфиксными термами в Пролог для обозначения односвязных списков введен безфункторный терм, описанный в *главе 1*. *Список* есть специальный вид двоичного безфункторного рекурсивного терма, у которого первый аргумент — это голова списка, а второй аргумент является хвостом списка. Разделителем между ними служит вертикальная черта. Все элементы списка должны принадлежать к одному типу.

Списочный терм правоассоциативен. Это означает, что список, содержащий три элемента, — например, `10`, `11` и `12`, с теоретической точки зрения представляет собой терм `[10 |[11 |[12 |[]]]]`. Такая запись списка трудна для восприятия, поэтому для упрощения записи и облегчения восприятия списки обозначаются кратко — в виде перечисления через запятую элементов, — например, `[10,11,12]`. Подобные упрощения синтаксиса в теории языков программирования называют «синтаксическим сахаром». Так, синтаксическим сахаром являются следующие варианты записи этого списка:

```
[10 | [11,12]]
[10, 11 | [12]]
[10, 11, 12 | []]
```

Далее представлены неправильные записи списков:

- `[1,2|3]` — хвост списка должен быть списком `[3]`, а не элементом `3`;
- `[1,"aaa",7]` — элементы списка должны принадлежать одному типу;
- `[1 |[2] |[3]]` — список имеет два хвоста, а допускается только один.

Если элементами списка являются тоже списки, то они называются *вложенными* списками, или списками первого уровня вложенности. Список может иметь произвольное число уровней вложенности.

Вертикальная черта, обозначающая разделитель между головой и хвостом, является операцией префикса списка. Префикс списка может быть как конструктором списка, так и селектором списка.

Конструктор собирает новый список из данного ему элемента, служащего головой нового списка, и данного ему списка, служащего хвостом нового списка. Например, если имеется элемент `5` и список `[10,20,30]`, то новый список можно построить так: `[5|[10,20,30]]`. Эту же операцию можно выразить формально:

```
A = 5,
X = [10,20,30],
NewList = [A|X].
```

В результате:

```
NewList = [5,10,20,30].
```

Селектор разделяет заданный список на первый элемент, т. е. голову списка, и остаток списка, т. е. хвост. Например, если задан список `[5,10,20,30]`, то его можно разделить на голову и хвост, используя две свободные переменные, следующим образом:

```
[A|X] = [5,10,20,30].
```

Результатом будет являться голова `5` и остаток списка `[10,20,30]`:

```
A = 5,
X = [10,20,30].
```

Селектор не применим к пустому списку `[]`, т. к. пустой список не разделяется на голову и хвост. Однако список, содержащий лишь один элемент, разделяется на голову — этот самый единственный элемент, и хвост — пустой список.

И конструктор, и селектор синтаксически обозначаются одинаково — вертикальной чертой. Различие заключается в том, связаны или свободны переменные, между которыми стоит вертикальная черта. Если переменные связаны, то конструируется новый список. Если переменные свободны, то разделяется тот список, который приравнивается к этим свободным переменным, разделенным вертикальной чертой. Такое приравнивание в Прологе выполняет операцию сопоставления с образцом. Эта же операция сопоставления с образцом обрабатывает те случаи сопоставления, когда список содержит как свободные, так и связанные переменные.

2.4. Операция сопоставления с образцом

Сопоставление с образцом (pattern matching) — это операция сопоставления двух термов с целью поиска множества подстановок. Под *подстановкой* понимается замена всех вхождений переменной в терме каким-либо основным термом. *Множество подстановок* — это замена всех переменных терма основными термами.

Общий терм, полученный после замены переменных их значениями, должен быть основным, т. е. не должен содержать свободных переменных. Такое ограничение, наряду с отсутствием проверки вхождения переменной в терм, с которым сопоставляется эта переменная, не только повышает скорость выполнения кода, но и увеличивает его надежность за счет глобального анализа потока данных на этапе компиляции.

2.4.1. Сопоставление термов

Сопоставление термов выполняется за три этапа:

1. Проверка равенства функторов сопоставляемых термов.
2. Проверка равенства арности сопоставляемых термов.
3. Поиск множества подстановок.

Если подстановки всех переменных найдены и являются основными термами, то сопоставление с образцом считается успешным, иначе — сопоставление с образцом считается неуспешным, и замена переменных найденными значениями не производится.

Примеры сопоставления с образцом двух термов приведены в табл. 2.1.

Таблица 2.1. *Примеры сопоставления с образцом*

№	Терм 1	Терм 2	Сопоставление	Подстановки	Общий терм
1	`X`	`24`	Успешно	`X=24`	`24` — основной
2	`X`	`f(99)`	Успешно	`X=f(99)`	`f(99)` — основной
3	`ace()`	`X`	Успешно	`X=ace()`	`ace()` — основной
4	`ace()`	`ace(88)`	Ошибка этапа компиляции	Нет	Нет
5	`ace(88)`	`face(88)`	Неуспешно	Нет	Нет
6	`X`	`Y`	Ошибка этапа компиляции	Нет	Нет
7	`t(X,45,Y)`	`t(2,Z,7)`	Успешно	`X=2, Y=7, Z=45`	`t(2,45,7)` — основной
8	`t(X,45,Y)`	`t(Z,Z,8)`	Ошибка этапа компиляции	Нет	Нет
9	`t(X,45,Y)`	`t(Z,45,8)`	Ошибка этапа компиляции	Нет	Нет
10	`t(X,45,Y)`	`t(Z,X,W)`	Ошибка этапа компиляции	Нет	Нет
11	`t(X,45,Y)`	`g(2,45,7)`	Неуспешна	Нет	Нет
12	`t(X,45,Y)`	`t(X,45)`	Ошибка этапа компиляции	Нет	Нет

Таблица 2.1 *(окончание)*

№	Терм 1	Терм 2	Сопоставление	Подстановки	Общий терм
13	`t(X,45,8)`	`t(2,Z,Z)`	Неуспешно	Нет	Нет
14	`X`	`s(X)`	Ошибка этапа компиляции	Нет	Нет
15	`ace(88)`	`ace(88)`	Успешно	Нет	`ace(88)` — основной

В строке 4 табл. 2.1 Visual Prolog допускает использование в программе двух термов с одинаковым именем, но с разной арностью. Такие термы считаются разными, что видно в строках 4 и 12.

Сопоставление двух термов, указанных в строке 8, не будет скомпилировано, т. к. в ходе глобального анализа потока данных компилятор обнаружит, что одной из промежуточных подстановок является подстановка `X=Z`, т. е. связывание двух свободных переменных, что не допускается алгоритмом сопоставления с образцом.

Сопоставление терма с анонимной переменной всегда успешно.

Операция сопоставления с образцом выражается явно знаком равенства — например, `t(X,45,Y) = t(2,Z,7)`. С другой стороны, сопоставление с образцом выполняется без явного указания знака равенства — например, при вызове функции с заданными параметрами. Параметры вызова сопоставляются с соответствующими параметрами, заданными в определении функции.

Операция сопоставления с образцом безразлична к расположению свободных переменных относительно знака равенства. Поэтому операции `X=7` и `7=X` эквиваленты.

2.4.2. Сопоставление списков

Списки являются безфункторными термами, поэтому сопоставление списков выполняется аналогично сопоставлению термов — последовательным сопоставлением элементов двух списков между собой. Если в списках есть вложенные списки, то выполняется сопоставление их элементов. В табл. 2.2 приведены примеры сопоставления двух списков.

Таблица 2.2. *Примеры сопоставления с образцом для списков*

№	Список 1	Список 2	Сопоставление с образцом	Подстановки	Общий терм
1	`X`	`[10,20,30]`	Успешно	`X=[10,20,30]`	`[10,20,30]`
2	`[X,20,30]`	`[10,20,30]`	Успешно	`X=10`	`[10,20,30]`
3	`[X,20,30]`	`[10,Y,30]`	Успешно	`X=10, Y=20`	`[10,20,30]`
4	`[X,20,30]`	`[10,Y,50]`	Неуспешно	Нет	Нет
5	`[X,20,30]`	`[X,20]`	Неуспешно	Нет	Нет