2.9. Согласование объектов

Согласование объектов – это процесс, в котором в качестве исходных данных берутся два разных объекта, и выполняется проверка того, являются ли они идентичными.

Когда объекты не идентичны, процесс согласования завершается неудачей.

В противном случае процесс завершается успешно, и происходит конкретизация переменных в обоих объектах такими значениями, что оба объекта становятся идентичными.

Существуют следующие правила согласования объектов:

1. *Константа* согласуется с другой константой, только если они представляют собой одинаковый объект.
2. *Свободная переменная* согласуется с любым объектом. При согласовании свободной переменной с другой свободной переменной они становятся сцепленными (но свободными). При согласовании с константой или другой связанной переменной, она конкретизируется значением константы или переменной и становится связанной.
3. *Связанная переменная* согласуется с константой или другой связанной переменной, если они идентичны.
4. *Анонимная переменная* согласуется с чем угодно, причём её конкретизация не осуществляется.
5. *Структура* согласуется с другой структурой, только если у них главные функторы одинаковые, а аргументы поддаются согласованию. Результирующая конкретизация определяется путём согласования аргументов структур.

2.10. Структура правила

Правила – это предложения, которые можно получить из заданных фактов и других правил.

Правило соответствуют зависимым связям, и позволяет выводить один элемент информации из другого.

Правило отличается от факта наличием абстрактных объектов и некоторого условия, от выполнения которого зависит истинность этого правила.

Все правила имеют общий синтаксис:

Head :– Body.

, где **Head** – заголовок правила,
:– – логическая операция импликация **(ЕСЛИ И ТОЛЬКО ЕСЛИ)**,
Body – тело правила.

Заголовок правила соответствует заключению (логическому следствию) и является утверждением, которое может быть истинным, если и только если выполнено некоторое условие.

Тело правила содержит условие (допущения или предпосылки), истинность которого требуется доказать. Условие задаётся одной целью или конъюнкцией целевых утверждений (подцелей). В теле правила подцели разделяются запятыми, а последняя подцель заканчивается точкой. Каждая подцель в правиле – это запрос к какому-либо предикату.

Импликация – логическая операция, которая используется при доказательстве правила и играет важную роль. Если доказана истинность импликации, то доказывается истинность заключения.

Заголовок и подцели – это отношения между объектами, описанные предикатами, поэтому правило в программе соответствует следующему формату:

relation (object$_1$, object$_2$, …, object$_N$) :–
relation (object$_1$, …, object$_M$),
…
relation (object$_1$, …, object$_K$).

, где **relation** – предикаты.

Объекты в отношениях правила задаются чаще всего переменными, но могут использоваться и константы.

Разрешается описание нескольких правил для одного и того же предиката, что соответствует логической связке **ИЛИ**. В подобном случае говорят о наличии процедуры в программе.

Процедура – это множество правил об одном и том же отношении.

Анализ структуры правила позволяет утверждать:

1. Факт – правило, у которого пустое тело.
2. Цель – правило без заголовка.

В сравнении с условным оператором **if** из традиционных языков программирования правило *Пролога* имеет другой смысл:

С: если (**заголовок** истинен),
{ тогда **выполнить тело** }

Пролог: если (**тело** может быть выполнено),
{ тогда **заголовок истинен** }

2.11. Согласование правил и целей

Если тело правила истинно, то истинно правило и его заголовок.

Для успешного разрешения правила *Пролог* должен разрешить все составляющие правило подцели. *Пролог*-система создаёт последовательный список переменных и должным образом конкретизирует их согласно телу правила и в соответствии с имеющимися фактами и другими правилами.

Цель **Goal** истинна, ***если и только если***

1. в программе имеется предложение **Clause** такое, что
2. существует экземпляр **I** предложения **Clause** такой, что
    a. заголовок **I** идентичен заголовку цели **Goal** и
    b. все цели в теле **I** истинны.

Список целей является истинным, если все цели в списке являются истинными **для одной и той же конкретизации переменных**. При успешном выполнении операции согласования её результатом становится наиболее общая конкретизация переменных.

Для представления целей в программе часто используются предикаты нулевой арности. Например:

```
predicates  run
clauses     run :– mather (Mam, Child).
goal        run.
```

2.12. Поиск решений в Прологе и обратный просмотр

Допустим, надо создать турнирную таблицу соревнований по пинг-понгу между девятилетними мальчиками теннисного клуба.

Пример 2-3. Турнирная таблица теннисного клуба.

```
domains
        child, sex = symbol         % фамилия и пол игрока
        age = integer               % возраст игрока

predicates
        nondeterm player (child, sex, age)

clauses
        player      (yin,       m,  9).     %   1
        player      (solomon,   m, 10).     %   2
        player      (marcellus, m,  9).     %   3
        player      (doores,    m,  9).     %   4

goal
        player (X, m, 9),       %   1-ая подцель
        player (Y, m, 9),       %   2-ая подцель
        X <> Y.                 %   3-я подцель
```

На естественном языке целевое утверждение звучит так:

*Найти 1-го игрока **X** мужского пола **m** в возрасте девяти лет и 2-го игрока **Y** мужского пола **m** в возрасте девяти лет, отличного от 1-го игрока.*

В логическом программировании одним из основных механизмов поиска решения является *обобщённое сопоставление с образцом*.

Пролог-система будет искать решения следующим образом:

1. Попытка найти решение для 1-ой подцели **player (X, m, 9).** Сопоставление с 1-ым предложением **player (yin, m, 9).** В результате подстановки **X = yin**. 1-ая подцель удовлетворена.
2. Попытка найти решение для 2-ой подцели **player (Y, m, 9).** Сопоставление с 1-ым предложением **player (yin, m, 9).** В результате подстановки **Y = yin**. 2-ая подцель удовлетворена.
3. Попытка удовлетворить 3-ью подцель **X <> Y (yin <> yin)** заканчивается неудачей, и 3-ья подцель не удовлетворяется. В этом случае *Пролог* возвращается к предыдущей подцели и ищет для неё другое решение. Такой механизм называется *обратный просмотр (back-tracking)*, или *откат*.

Если текущая цель не согласуется, то происходит возврат (откат) к той цели, где произошла конкретизация переменной, из-за которой текущая цель не удовлетворяется. Далее следует попытка применить к этой цели альтернативное предложение, чтобы переменная получила другую конкретизацию. После этого продолжается последовательное согласование целевых утверждений.

4. Попытка найти альтернативное решение для 2-ой подцели. Сопоставление с 3-им предложением **player (marcellus, m, 9).** В результате подстановки **Y = marcellus**. 2-ая подцель удовлетворена. Не срабатывает подстановка **Y = solomon**, так как не проходит сопоставление.
5. 3-я подцель удовлетворяется, так как **yin <> marcellus** истина. Получается первое решение: **X = yin Y = marcellus**
6. *Пролог*-система пытается найти **все** возможные решения, поэтому будет возврат ко 2-ой подцели. Сопоставление с 4-ым предложением, и подстановка **Y = doores**.
7. 3-я подцель удовлетворяется, следовательно, будет получено второе решение: **X = yin Y = doores**
8. 2-ая подцель не имеет других решений, а значит возврат к 1-ой подцели. Сопоставление **solomon** не проходит, так как его возраст 10 лет.

9. 1-ая подцель сопоставляется с 3-ьим предложением, происходит подстановка **X = marcellus**.
10. 2-ая подцель в результате сопоставления и подстановки даёт **Y = yin**.
11. 3-я подцель даёт третье решение: **X = marcellus Y = yin**
12. Возврат ко 2-ой подцели приводит к результату **Y = marcellus**, минуя **solomon**.
13. Неудача 3-ей подцели, так как **marcellus <> marcellus** ложь.
14. Обратный просмотр приводит к сопоставлению 2-ой подцели с 4-ым предложением, а подстановка связывает **Y = doores**.
15. 3-ья подцель даёт четвёртое решение: **X = marcellus Y = doores**

По аналогии будут найдены ещё два решения:

X = doores Y = yin **X = doores Y = marcellus**

В результате всего будет получено шесть решений.

Пример лабораторной работы № 2 приведён в [1].

Для успешного разрешения цели *Пролог* должен разрешить все составляющие правило подцели. Для согласования каждой подцели *Пролог*-система выполняет следующие действия:

1. Название подцели сравнивается с названием отношения из секции **clauses**. Если сравнение неудачно, то решений нет. Если же выявлено предложение, оно выбирается для последующего согласования.
2. Сравниваются количество и типы объектов в подцели и выбранном предложении. Если сравнение неудачно, то решений нет.
3. Последовательно согласуются объекты подцели и выбранного предложения. Если объекты не согласуются, то возможен откат к предыдущей подцели, дающей альтернативные решения. После возврата поиск решения продолжается с новыми значениями. Откаты происходят по направлению к заголовку правила до тех пор, пока не будут исчерпаны все возможные варианты конкретизации переменных. Когда подцель для отката отсутствует, решений нет. Если согласование удачно, то имеющиеся свободные переменные конкретизируются.

2.13. Простой ввод-вывод в Прологе

Под простым вводом обычно понимают ввод с клавиатуры, а под простым выводом – вывод на экран дисплея.

2.13.1. Предикаты чтения

Существует четыре встроенных предиката чтения, формат которых показан в табл. 5.

Каждый предикат читает данные своего типа.

Таблица 5

Предикаты чтения

Формат предиката	Назначение
readln (String)	ввод строки
readint (Integer)	ввод целого числа
readreal (Real)	ввод вещественного числа
readchar (Char)	ввод символа

Все используемые в предикатах чтения *переменные должны быть свободными*.

Неудачу предикаты терпят по следующим причинам:

- нажата клавиша **<Esc>** для предиката **readln**;
- неправильный символ (не цифра, не знак числа, не точка) для предикатов **readint** и **readreal**.

2.13.2. Предикаты записи

Для записи существует три встроенных предиката: **nl**, **write**, **writef**:

1. *Переход на новую строку (new line)*

   nl

2. *Обычный вывод*

   write (Arg_1, Arg_2, ..., Arg_N)

   В качестве аргументов Arg_j могут быть использованы константы или переменные. Количество аргументов этого предиката произвольно.

   Если параметр предиката записи – это переменная, то она обязательно должна быть связанной.

   Пример 2-4. Обычный вывод.

```
predicates
        address (real, symbol, symbol, integer)
        writeConst
        writeVar
goal    writeConst, nl, writeVar, exit.
clauses
        address (190031,"СПб", "Московский пр.", 9).
        writeConst :–
                write (190031, ", СПб"),
                nl, write ("Московский пр., ", 9).
        writeVar :–
                address (Index,Town, Street, House),
                write ("Индекс:\t", Index),
```

```
        nl, write ("Город:\t", Town),
        nl, write ("Улица:\t", Street),
        nl, write ("Дом:\t", House).
```

Решение

```
190031, СПб
Московский пр., 9
Индекс:    190031
Город:     СПб
Улица:     Московский пр.
Дом:       9
```

3. *Форматный вывод*

writef (FormatString, Arg_1, Arg_2, ..., Arg_N)

Предикат выводит значения аргументов Arg_j в соответствии с заданным при помощи **FormatString** шаблоном вывода.

Форма шаблона **%–m.pf**

- *дефис* указывает, что поле вывода выравнивается слева (по умолчанию выравнивание справа);
- **m** определяет минимальную длину поля;
- **p** определяет или точность представления вещественного числа, или максимальное число печатаемых в строке символов;
- **f** аналогично языку C представляет символ формата вывода; допустимые символы показаны в табл. 6.

Таблица 6

Символы формата вывода

Символ	Формат	Символ	Формат
f	вещественный	**e**	экспоненциальная форма
g	вещественный по умолчанию		
d	целый	**s**	строка
u	без знака	**c**	символ

Пример 2-5. Форматный вывод.

```
writef ("A = '%-7s'\nB = %8.1e\nB = %8.2e\n", first, 33.17, 33.17)
writef ("char = %c\nint  = %d\n", 65, 65),
writef ("hex = %x\t uns = %u\n", 65, 65)
```

Решение:

```
A = 'first  '
B =   3.3E+01
B =  3.32E+01
char = A
int  = 65
```

hex = 41 uns = 65

2.14. Простые объекты

Любой объект, который представляет сам себя, называется простым объектом.

Простой объект данных может быть или переменной, или константой:

- Константа определяет конкретный объект, не подлежащий изменениям.
- Переменная представляет неопределённый объект, значение которого устанавливается в процессе согласования с любым допустимым конкретным объектом.

Методические указания

1. *Довбуш Г. Ф.* Лабораторные работы по логическому программированию. Методические указания. – СПб: ПГУПС, 2005.