УДК 681.3.062

О.Н. Половикова

Анализ семантических ошибок и ошибок выполнения, встречающихся в коде программ на языке Пролог*

O.N. Polovikova

The Analysis of Semantic Errors and Runtime Errors in a Prolog Language Program Code

В статье представлен анализ трех основных групп логических ошибок. Первая группа ошибок связана с некорректным использованием встроенных операторов языка Пролог. Вторая группа объединяет ошибки выполнения, которые возникают во время работы программы на некоторых начальных данных. Третья группа формируется ошибками некорректного построения порядка в описании фактов и правил процедур предложений. Проведенный анализ является одним из начальных этапов исследования по разработке и созданию программного модуля с функциями поиска некорректных семантических определений и предоставления вариантов их исправления.

Ключевые слова: логическое программирование, Пролог, декларативный уровень программирования, процедурный уровень программирования, семантические ошибки, ошибки выполнения программы.

The article presents the analysis of three basic groups of logic errors. The first group of errors is connected with incorrect use of the built in the Prolog language operators. The second group unites runtime errors which arise in a program operating time on some initial data. The third group is formed by errors caused by incorrect construction of an order in the description of the facts and rules of offers procedures. The analysis carried out by the authors is one of the research initial stages to work out and create the program module with functions to search incorrect semantic definitions and grant variants of their correction.

Key words: logic programming, Prolog, declarative level of programming, procedural level of programming, semantic errors, runtime errors

Декларативные языки можно разделить на функциональные и логические. Пролог является логическим языком, так как позволяет выполнить описание проблемы в терминах фактов и логических формул, а решение проблемы выполняет система с помощью механизмов логического вывода. Требуемый результат (цель, которую нужно достичь Пролог-системе) оформляется конъюнкцией утверждений, а механизм определения истинности каждого утверждения реализован посредством соотнесения утверждения с некоторым набором шаблонов и реализован в устройстве языка [1]. Декларативность языка приводит к различным логическим ошибкам в коде программ, на которые компилятор языка никаких предупреждений не выводит.

В данной статье представлен анализ трех основных групп логических ошибок. Первая группа ошибок связана с некорректным использованием встроенных операторов языка Пролог. Вторая группа объединяет ошибки выполнения, которые возникают во время работы программы на некоторых начальных данных (исключительные ситуации). Ошибки из этой группы могут привести к зацикливанию программы, переполнению стековой памяти, некорректному результату. Третья группа формируется ошибками некорректного построения порядка в описании фактов и правил процедур предложений.

Выделенные группы ошибок сложно диагностировать. Представленный анализ позволит усовершенствовать их поиск и исправление в практических разработках.

Предлагаемая классификация ошибок построена на основе анализа информационных источников [1–4], а также опыта программирования на языке Пролог в среде Strawberry Prolog.

Ниже представлены примеры программ или процедур предложений с ошибками по каждой группе, выделены причины возможного возникновения, а также предложены варианты их исправления.

Ошибки некорректного использования встроенных операторов языка Пролог. Выражения, построенные с использованием операторов (арифметических, бинарных, сравнения), являются структурами. Операторы языка – это особая форма синтаксиса соответствующих структур, где функтором является символ операции, компонентами выступают операнды. Для удобства записи и чтения таких выражений принято использовать их операторную форму. Поэтому только применение арифметических или бинарных операторов не вызывает выполнения каких-либо операций (необходимо использовать оператор “is” или “:=”).

* Данная работа выполнена при финансовой поддержке РФФИ (проект №10-01-98005 р_сибирь_а).

Например, *5 – 4* и *1* – это разные объекты, и между ними нельзя установить соответствие; между *X – 1* и *5*, даже если ранее переменной *X* присвоено значение 6, также нельзя установить соответствие. Утверждение *sum(5+4)* не согласуется с фактами *sum(9)*, *sum(X + 1)*, при любых значениях переменной *X* (при согласовании утверждения со вторым фактом невозможно установить соответствие между двумя структурами: *+(5,4)* и *+(X,1)*). Некорректное использование встроенных операторов языка Пролог приводит к несогласованности левой и правой частей оператора "=" (сравнение на равенство), как следствие этого, невозможно установить истинность соответствующего утверждения, корректно воспользоваться нужным предложением или процедурой.

Ошибки выполнения программы. Чтобы научиться определять (диагностировать) данную группу ошибок, следует проанализировать процедурный уровень программирования языка Пролог.

Процесс создания программы на языке Пролог включает декларативный и процедурный уровни программирования. Только декларативный уровень программирования на языке Пролог не позволит реализовать эффективный поиск альтернативы решения в случае их комбинаторного перебора (оптимизировать вычислительный процесс), а также выявить логические ошибки в программе.

На процедурном уровне программирования важен порядок предложений внутри совокупности одноименных правил и фактов (в процедуре предложений), а также порядок хвостовых целей в теле предложений. От порядка предложений зависит порядок поиска решений и порядок, в котором будут находиться ответы на поставленные вопросы, так как при попытке повторного согласования цели (в случае неудачного сопоставления какой-либо подцели) система возобновляет просмотр базы с предложения, непосредственно следующего за тем, которое обеспечивало согласование цели ранее. Порядок целей (подцелей) влияет на количество проверок, выполняемых программой при решении.

Рассмотрим пример, демонстрирующий зависимость количества выполняемых системой проверок от порядка хвостовых подцелей в вопросе.

База данных электронной библиотеки в Пролог-программе состоит из трех видов фактов. Факт *production(N,Y,…)* описывает литературное произведение: его номер, автор, другие свойства (*n* фактов). Факт *reader(R,F,…)* определяет читателя следующими характеристиками: его номер, фамилия и другие параметры (*m* фактов). Предложение *get_book(N,R,…)* описывает факт выдачи произведения читателю и характеризуется свойствами: номер произведения, номер читателя (*k* фактов). Необходимо построить ответ на вопрос: кому из читателей (фамилия читателя) было выдано произведение Л.Н. Толстова?

Первый вариант вопроса:

?- production (N,"Толстой_Л_Н",…), get_book(N,R,…), reader(R, F,…), вывод(M).

Второй вариант вопроса:

?- get_book(N,R,…), reader(R, F,…), production (N,"Толстой_Л_Н",…), вывод(M).

Предположим, что в базе данных факты, соответствующие сформированной для Пролог-системы цели, расположены в конце списка предложений, тогда поиск ответа на второй вариант вопроса требует выполнить *k*m*n* проверок, на первый – *(n + k + m)* проверок.

Данный пример наглядно показывает необходимость процедурного уровня программирования при решении задач на языке Пролог. Рассмотрим следующий пример, демонстрирующий программный код с ошибкой выполнения.

Требуется создать программу на языке Пролог для нахождения суммы элементов списка.

```
/*код программы с ошибкой выполнения (строки программы пронумерованы с использованием комментариев)*/
/*1*/?-ввод(L,"Input list:"), summa(L,Sum), вывод(Sum).
/*2*/summa([H|T],Sum):-Sum is H+Sum1, summa(T,Sum1).
/*3*/summa([],0).
```

Во второй строке кода программы некорректно построен порядок целей в хвостовой части правила *summa([H|T],Sum)*: переменная *Sum1* используется в выражении с неопределенным значением (переменная конкретизируется в следующем выражении). Правильный порядок должен быть следующим:

Sum is H + Sum1, summa(T,Sum1).

Пренебрежение правилами повышения эффективности работы программы (уменьшение количества просмотра возможных альтернатив в процессе согласования цели) может привести не только к снижению скорости выполнения программы, но и к серьезным ошибкам, в том числе и к переполнению стека памяти Пролог-системы (комбинаторные взрывы).

Ошибки неполного описания условий выполнения факта или правила. Сложность поиска подобных ошибок заключается в том, что некорректную работу программы можно обнаружить только в случае, если произойдет повторное согласование утверждения, в описании процедуры предложений которого допущена неточность. Поиск такого рода ошибок возможен только в том случае, если разработчик знает не только декларативный, но и процедурный уровень программирования на Прологе. Рассмотрим пример программы с ошибкой.

Постановка задачи: определить, является ли сумма моделей двух заданных чисел меньше некоторой числовой константы (например, *12*).

Пример программы для решения поставленной задачи:

```
/*1*/?- ввод(X,"X= "), ввод(Y,"Y="), abs(X,A), вывод(A),abs(Y,B),вывод(B),(A+B)<12.
/*2*/abs(X, X):- X>0.
/*3*/abs(X, -1*X).
```

В первой строке указана цель, в случае успешного согласования которой Пролог система возвратит ответ «Да», в противном случае – «Нет». Во второй

и третьей строках описана процедура из двух предложений для определения модуля числа: во второй строке указано правило вычисления модуля для положительного значения переменной *X*, в третьей строке описано правило (факт), которое выполняется, если не выполнено правило из второй строки. Если не выполняется условие $X > 0$, то второй параметр отношения *abs(X,A)* будет сопоставлен со значением: *–1*X*.

В данных рассуждениях нет ошибок. Пролог-система в процессе поиска решения (при согласовании цели программы, заданной в первой строке) будет пытаться согласовать каждую подцель с правилом или фактом из соответствующей процедуры предложений, рассматривая эти предложения в строгом порядке (в порядке следования в тексте программы). Но при решении поставленной задачи не учитывается вся специфика работы механизма возврата, на котором основан поиск решения Пролог-системой. Если утверждение $(A + B) < 12$ ложное, Пролог-система в цепочке конъюнкции подцелей будет заново пытаться согласовывать каждую подцель, пока либо все подцели не будут согласованы (ответ «Да»), либо все возможные варианты согласования подцелей будут рассмотрены, но решения не найдено (ответ «Нет»). Поэтому если утверждение $(A + B) < 12$ ложное, то Пролог-система будет пытаться повторно согласовывать подцель *abs(Y,B)* или подцели *abs(X,B)* и *abs(X,A)*. Если переменная *Y* (или *X*) конкретизирована положительным значением, то возникнет логическая ошибка. Пролог-система отношение *abs(Y,B)* (или *abs(X,A)*) согласует с фактом *abs(X, –1*X)*.

Рассматриваемая программа на значениях $X = –7$ и $Y = –6$ выдает корректный результат: «Нет», так как подцели *abs(Y,B)* и *abs(X,B)* согласуются только с фактом *abs(X, –1*X)*, механизм возврата не будет заново просматривать предложения процедуры, потому что все возможные варианты согласования подцелей уже рассмотрены. А вот на значениях $X = 7$ и $Y = –6$ (или $X = –7$ и $Y = 6$, или $X = 7$ и $Y = 6$) программа выдаст некорректный результат «Да». Так как утверждение $(A + B) < 12$ на значениях $A = 7$ и $B = 6$ будет ложным, произойдет возврат и повторное согласование всех указанных подцелей (переменной *X* с положительным значением процедура *abs(X,A)* поставит в соответствие переменную *A* с отрицательным значением).

Существует несколько способов исправления подобных ошибок. Например, можно воздействовать на процесс возврата, используя предикат «отсечение». Данный встроенный предикат позволяет запретить просматривать в процессе возврата все альтернативы предложений при доказательстве подцели, если ранее она была согласована с правилом, где есть «отсечение» (первый вариант исправления ошибки).

Второе предложение процедуры *abs(,)* можно записать в виде правила (второй вариант устранения ошибки), чтобы гарантировать его выполнение только для неположительных значений первого параметра.

```
/*Устранение ошибки, воздействием на процесс возврата*/
/*2*/abs(X, X):- X>0, !.
/*3*/abs(X, -1*X).
/*Устранение ошибки, заменой факта правилом*/
/*2*/abs(X, X):- X>0.
/*3*/abs(X, -1*X):- X=<0.
```

Необходимо заметить, что выделенные группы не перекрывают всю совокупность возможных семантических ошибок или ошибок выполнения программы. Проведенный анализ является одним из начальных этапов исследования по разработке и созданию программного модуля со следующими функциями:

1) обнаружение (поиск) некорректных семантических определений в коде программ на языке Пролог;

2) предоставление вариантов их исправления;

3) построение рекомендаций по изменению порядка целей в хвостовых частях предложений или в последовательности фактов и правил созданных процедур (минимизация количества выполняемых Пролог-системой проверок).

Формализация возможных ошибок позволит определить множество шаблонов их декларативного описания для последующего использования в программном модуле.

Существующие компиляторы языка Пролог способны обнаруживать только синтаксические ошибки в коде. Использование данного модуля предоставит возможность разработчикам до этапа тестирования выполнить диагностику программы на некорректные декларативные определения, а также получить рекомендации по оптимизации кода.

Библиографический список

1. Братко И. Программирование на языке Пролог для искусственного интеллекта: пер. с англ. – М., 1990.

2. Кучуков А.М. Логическое программирование и Visual Prolog. – СПб., 2003.

3. Малпас Дж. Реляционный язык Пролог и его применение: пер. с англ. – М., 1990.

4. Шрайнер П.А. Основы программирования на языке Пролог. – М., 2005.