Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 19

Обработка списков на Prolog

Студент Лучина Е.Д

Группа ИУ7-61Б

Преподаватель Толпинская Н.Б.

Москва.

апрель 2020 г.

Содержание отчета

Полный текст задания,

Ответы на вопросы,

Текст программы с комментариями обозначений и идентификаторов аргументов,

Таблица для 2-ого задания, демонстрирующая работу системы при одном из

успешных вариантов вопроса.

Выводы: за счет чего достигнута эффективность работы каждой программы.

Задание

Ответить на вопросы (коротко):

1. Что такое рекурсия? Как организуется хвостовая рекурсия в Prolog? Как можно организовать выход из рекурсии в Prolog?
2. Какое первое состояние резольвенты?
3. В каких пределах программы переменные уникальны?
4. В какой момент, и каким способом системе удается получить доступ к голове списка?
5. Каково назначение использования алгоритма унификации?
6. Каков результат работы алгоритма унификации?
7. Как формируется новое состояние резольвенты?
8. Как применяется подстановка, полученная с помощью алгоритма унификации – как глубоко?
9. В каких случаях запускается механизм отката?
10. Когда останавливается работа системы? Как это определяется на формальном уровне?

Используя хвостовую рекурсию, разработать эффективную программу, (комментируя назначение аргументов), позволяющую:

1. Найти длину списка (по верхнему уровню);
2. Найти сумму элементов числового списка
3. Найти сумму элементов числового списка, стоящих на нечетных позициях исходного списка (нумерация от 0)

Убедиться в правильности результатов

Для одного из вариантов ВОПРОСА и одного из заданий составить таблицу, отражающую конкретный порядок работы системы:

Т.к. резольвента хранится в виде стека, то состояние резольвенты требуется отображать в столбик: вершина – сверху! Новый шаг надо начинать с нового

состояния резольвенты! Для каждого запуска алгоритма унификации, требуется указать № выбранного правила и дальнейшие действия – и почему.

Текст процедуры, Вопрос:….

Шаг	Текущая резольвента (ТР)	ТЦ, выбираемые правила: сравниваемые термы, подстановка	Дальнейшие действия с комментариями
№!	...	...	...

Ответы на вопросы

1. Что такое рекурсия? Как организуется хвостовая рекурсия в Prolog? Как можно организовать выход из рекурсии в Prolog?

   Рекурсия - ссылка на себя. В Prolog это способ повторного использования процедуры. Хвостовая рекурсия - вид рекурсии, где рекурсивный “вызов” осуществляется в конце процедуры. Выход из рекурсии в prolog может организовываться с помощью предикатов отсечения и fail, заголовки и правил и факты должны быть сформулированы так, что система находит решение или входит в тупик и заканчивает работу.

2. Какое первое состояние резольвенты?

   Поставленный в goal вопрос.

3. В каких пределах программы переменные уникальны?

   Именованные переменные уникальны в пределах предложения. Анонимные уникальны во всей программе.

4. В какой момент, и каким способом системе удается получить доступ к голове списка?

   В Prolog используется специальный символ для разделения списка на голову и хвост - вертикальная черта |. Например: [1, 2, 3] или [1 | [2, 3]].

   В правиле func([X | Y]) :- .... при унификации X - будет хранить значение головы, Y - хвоста переданного списка.

5. Каково назначение использования алгоритма унификации?

   Назначение алгоритма унификации - подобрать в базе знаний подходящее знание, позволяющее на вопрос ответь да, доказать его истинность.

6. Каков результат работы алгоритма унификации?

“yes” в случае успеха, если удалось на основе БЗ доказать истинность утверждения в goal. Иначе no. Если вопрос содержит именованные переменные то в процессе унификации они конкретизируются и являются результатом

работы программы. Программа выведет все варианты конкретизации этих переменных и количество найденных решений. Или выведет “No Solution” - решения нет.

7. Как формируется новое состояние резольвенты?

Разрешенная подцель резольвенты заменяется на тело предложения, успешно унифицированное с ней. В случае факта тело пустое, поэтому подцелей в резольвенте становится на одну меньше. При этом эта подцель отправляется в стек для возврата, используемом при откате. В случае неудачи при поиске ответа или достижении конца базы знаний запускается механизм отката, который с помощью “мусорного” стека восстанавливает предыдущую подцель. Попутно происходит конкретизации и ре-конкретизация переменных.

8. Как применяется подстановка, полученная с помощью алгоритма унификации – как глубоко?

Все переменные в резольвенте конкретизируются значениями полученной подстановки (если для этих переменных найдено значение). Существуют переменные имеющие одинаковое имя, но разные области видимости соответственно они являются разными переменными.

9. В каких случаях запускается механизм отката?

Найдено решение, но не все записи БЗ отмечены. Алгоритм унификации зашел в тупик, завершился неуспешно.

10. Когда останавливается работа системы? Как это определяется на формальном уровне?

Когда резольвента пуста и все знания в БЗ отмечены.

Листинг программы

Листинг 1 - длина списка (по верхнему уровню).

```
domains
     ilist = integer*.

predicates
     length(ilist, integer)

clauses
     length([], 0).
     length([_|Y], Length) :-
          length(Y, LY), Length = LY + 1.
```

Примеры работы. Тестирование.

`length([], 0).`	`yes`

length([], Len).	Len=1 1 Solution
length([1, 2, 3], 5).	no
length([1, 2, 3], Len).	Len=3 1 Solution

Если список пустой [] он унифицируется с первым фактом, переменная длины будет связана со значением 0 (если есть переменная, иначе в результате унификации установим истинность предполагаемой передаваемой длины). Если список не пустой, он будет унифицирован со вторым термом. Соответственно голова списка нас не интересует, обозначим первый элемент списка анонимной переменной, Y будет связана с хвостом списка (пустой или непустой список), Length будет конкретизирована с теле правила. В теле правила сначала запускается рекурсивная цель length(Y, Ly), чтобы получить длину хвоста - Ly. Передаем хвост списка и переменную, с которой будет связана длина хвоста. После чего прибавляем в Ly единицу и результат связываем в переменной Len. Если хвост (Y) пустой, произойдет унификация length(Y, Ly), с первым термом Ly станет равной 0. Иначе будет запускаться рекурсия пока хвост не станет пустым списком Первый терм является тупиком данной рекурсивной процедуры.

Листинг 2 - сумма элементов числового списка.

```
domains
     ilist = integer*

predicates
     sumelems(ilist, integer)

clauses
     sumelems([], 0).
     sumelems([X|Y], Sum):-
          sumelems(Y, Sy),
          Sum = X + Sy.
```

Примеры работы. Тестирование.

sumelems([], 0).	yes
sumelems([5], 5).	yes
sumelems([3, 4], 9).	no
sumelems([1, 2, 3, 4], S).	S = 10 1 Solution
sumelems([], S).	S = 0

	1 Solution

Вопрос первый аргумент которого пустой список будет унифицирован с первым термом: неуспешно, если второй аргумент не 0: успешно, если 0; если передана переменная длины, она будет конкретизирована нулем. Если список не пустой, произойдет унификация со вторым термом. X = голова списка, первый элемент. Y = хвост списка, переданное значение суммы или переменная = Sum. С помощью рекурсии вычислим сумму (Sy) элементов хвоста (Y): sumelems(Y, Sy). После чего прибавим к ней первый элемент X, результат унифицируем с Sum. Если была передана переменная, то она является связанной с Sum = сумме элементов, иначе программа попытается унифицировать переданное значение суммы и вычисленное. Если хвост пустой, то sumelems(Y, Sy) будет унифицировано с sumelems([], 0) => {Sy = 0}. Это является тупиком рекурсии.

Листинг 3.

поиск суммы элементов числового списка, стоящих на нечетных позициях исходного списка (нумерация от 0).

```
domains
     ilist = integer*

predicates
     sumodd(ilist, integer)

clauses
     sumodd([], 0).
     sumodd([_|[]], 0).
     sumodd([_, X| Z], S) :-
          sumodd(Z, Sz),
          S = X + Sz.
```

Примеры работы. Тестирование.

sumodd([], 0).	yes
sumodd([], S).	S = 0 1 Solution
sumodd([5], S).	S = 0 1 Solution
sumodd([2], 6).	no
sumodd([1, 2, 3], S).	S = 2 1 Solution

sumodd([3, 2], S).	S = 2 1 Solution
sumodd([1, 2, 3, 4, 5, 6, 7, 8, 9], S).	S = 20 1Solution

Пример вопроса с пустым список будет унифицирован с sumodd([], 0), в результате чего сумма будет равна нулю (или произойдет проверка равенства переданного значения нулю). Если список содержит только один элемент [X|[]], то сумма равна 0 - факт sumodd([_|[]], 0). Элемент имеет четный индекс 0. Оба эти терма являются тупиками рекурсии. Первый если список пустой или содержит четное количество элементов, второй - если нечетное количество элементов. Рекурсивное предложение - sumodd([_, X| Z], S) :- sumodd(Z, Sz), S = X + Sz - заключается в том, чтобы идти по списку парами, добавляя в сумму второй элемент и пропуская первый. [_, X | Z] - _ - первый элемент, который не имеет значения, X- второй элемент, Z - хвост. Sz - сумма элементов хвоста, имеющие нечетные индексы.

Таблица

Таблица для 2-ого задания, демонстрирующая работу системы при одном из успешных вариантов вопроса.

Текст процедуры и рассматриваемый вопрос

```
...
clauses
   sumelems([], 0).
   sumelems([X|Y], Sum):-
      sumelems(Y, Sy),
      Sum = X + Sy.
goal
   sumelem([4, 5], S).
```

Таблица 1.

Шаг	Текущая резольвента (ТР)	ТЦ, выбираемые правила: сравниваемые термы, подстановка	Дальнейшие действия с комментариями
№!	sumelem([4, 5], S).	ТЦ = sumelem([4, 5], S). унификация с начала базы sumelem([4, 5], S) != sumelems([], 0).	унификация невозможна возврат к ТЦ, метка переносится ниже
		ТЦ = sumelem([4, 5], S). поиск зания от метки sumelem([4, 5], S). = sumelems([X|Y], Sum) Успешная унификация Подстановка = {X = 4, Y = [5], Sum = S}	Прямой ход. Проверка тела правила. Изменение резольвенты - замена цели на тело правило. Применение подстановки к резольвенте.

№2	sumelems([5], Sy1), S = 4 + Sy1.	ТЦ = sumelem([5], Sy1). унификация с начала базы sumelem([5], Sy1) != sumelems([], 0).	унификация невозможна возврат к ТЦ, метка переносится ниже
		ТЦ = sumelem([5], Sy1). поиск зания от метки sumelem([5], Sy1). = sumelems([X|Y], Sum) Успешная унификация Подстановка = {X = 5, Y = [], Sum = Sy1}	Прямой ход. Проверка тела правила. Изменение резольвенты - замена цели на тело правило. Применение подстановки к резольвенте.
№3	sumelems([], Sy2), Sy1 = 5 + Sy2. S = 4 + Sy1.	ТЦ = sumelem([], Sy2). поиск зания с начала базы sumelem([], Sy1). = sumelems([], 0) Успешная унификация Подстановка = {Sy2 = 0}	Подобрано знаний - факт. Замена ТЦ на пустое тело правила. Конкретизирована Sy2, применение подстановки к резольвенте.
№4	Sy1 = 5 + 0. S = 4 + Sy1.	ТЦ = Sy1 = 5 + 0. Подстановка = {Sy1 = 5}	Цель успешно унифицирована, удаление ТЦ из стека, представляющую резольвенту. Применение подстановки.
№5	S = 4 + 5.	ТЦ = S = 4 + 5 Подстановка = {S = 9}	Цель успешно унифицирована, удаление ТЦ из стека, представляющую резольвенту.
№6	пуста	Решение найдено	Вывод решение, откат (в несколько этапов) в поисках другого решения. Развязывание переменных.
№7	sumelems([], Sy2), Sy1 = 5 + Sy2. S = 4 + Sy1.	ТЦ = sumelem([], Sy2). поиск зания от метки sumelem([], Sy2). = sumelems([X|Y], Sum) неуспешная унификация	Неуспех, откат. Отмечены все знания БЗ, откат.
№8	sumelems([5], Sy1), S = 4 + Sy1.	ТЦ = sumelem([5], Sy1). Отмечены все знания БЗ	откат
№9	sumelem([4, 5], S).	ТЦ = sumelem([4, 5], S). Найдены все возможные решения для ТЦ	Разрешена, удаление из резольвенты
№10	пуста	Больше целей нет	Завершение работы программы

Используя предикат отсечения в первом терме, можно было бы пропустить шаг №7, включающий в себя откат и лишние (заведомо неуспешные) попытки унификации. Это бы сделало программу эффективнее.

Выводы

В Prolog рассматривая вариант работы программы с поиском всех возможных решений, алгоритм поиска решений осуществляет полный перебор всех знаний. Для достижения большей эффективности стоит использовать предикаты отсечения ! и fail. Они позволяют не рассматривать бесперспективные ветви поиска и завершают поиск для данного примера вопроса либо успехом (!) либо неудачей (fail). Также на эффективность влияет формулировка правил и фактов. Если тело правило содержит проверки его применимости, их стоит внести в заголовок в качестве аргументов. Например list(L) :- L = []. Заменить на list([]). Все это ускоряет процесс унификации, избавляет от ненужных переменных, уменьшает количество работы по изменению резольвенты.