

Государственное образовательное учреждение высшего
профессионального образования



«Московский государственный
технический университет
имени Н. Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ

«Информатика и системы управления»

КАФЕДРА

«Программное обеспечение ЭВМ и информационные
технологии»

О Т Ч Е Т П О П Р О И З В О Д С Т В Е Н Н О Й П РА К Т И К Е

Студент

Анисимов Никита Сергеевич

(фамилия, имя, отчество)

Группа

ИУ7-72

Тип практики

производственная

Название предприятия

МГТУ им. Баумана

Студент

(Подпись, дата)

Анисимов Н.С.

Руководитель

(Подпись, дата)

Толпинская Н.Б.

Оценка

Москва 2018

Содержание

Индивидуальное задание	3
Введение	4
Цель	4
Задачи	4
1 Аналитическая часть	5
1.1 Основные элементы языка Пролог	5
1.2 Особенности использования переменных	5
1.3 Структура программы	5
1.4 Понятие процедуры	6
1.5 Подстановка	6
1.6 Алгоритм унификации	6
1.7 Наиболее общий унификатор	7
1.8 Порядок работы	7
1.9 Резольвента	7
1.10 Список	8
2 Технологический раздел	9
2.1 Синтаксический анализатор	9
2.2 Алгоритм унификации	11
2.3 Резольвента	11
2.4 Доказательство	12
2.5 Последовательный поиск решение	14
2.6 Пример работы программы	14
Заключение	18
Список использованных источников	19

Индивидуальное задание

Разработка приложения для визуализации дерева поиска решений, получаемого в ходе работы программы на языке Пролог.

Введение

Цель

Целью данной работы является разработка приложения для визуализации дерева поиска решений, получаемого в ходе работы программы на языке Пролог.

Задачи

Поставлены следующие задачи:

- Разработка синтаксического анализатора языка Пролог;
- Реализация алгоритма унификации;
- Реализация алгоритма поиска решений;
- Разработка пошагового поиска решений.

1 Аналитическая часть

1.1 Основные элементы языка Пролог

Основным элементом языка является терм. Терм – это либо константа, либо переменная, либо составной терм. Составной терм показывает наличие отношения между аргументами. Константа – это символьный атом, начинающийся с маленькой буквы. Именованная переменная – это символьный атом, начинающийся с большой буквы. Символьный атом является неименованной переменной, если он начинается с символа `_`. Составной терм – это терм вида $f(t_1, t_2..t_n)$, где $t_1, t_2..t_n$ – термы-аргументы, f – главный функтор или имя. отношения. Количество n аргументов – арность.

1.2 Особенности использования переменных

Переменная конкретизирована, если в некий момент времени ей соответствует значение. Именованная переменная уникальна в рамках одного предложения. Анонимная переменная всегда уникальна. Она не может быть конкретизирована и не может передать значение на другой шаг доказательства.

1.3 Структура программы

Программа состоит из базы знаний и вопроса. База знаний – из фактов и правил. Правило – это «условная истина» или «теорема». Имеет следующий вид: `<заголовок>:-<тело>`. Факт – «безусловная истина», «аксиома», правило без тела. Факт – частный случай правила. Факт без переменных называется основной. Вопрос – это специальный тип предложений. Ответом на вопрос является либо да, либо нет. Побочный эффект – данные, при которых был получен ответ да. Факты, правила и вопрос представляются в виде терма.

1.4 Понятие процедуры

Процедура – совокупность правил, заголовки которых согласуются с одной и той же целью (одно сложно определенное знание). Процедуры нужны, когда знание не уместается в одно предложение.

1.5 Подстановка

Подстановкой называется множество пар вида $\{x_i = t_i\}$, где t_i это термы не содержащие переменных. Пусть $\Theta = \{x_1 = t_1, x_2 = t_2 \dots x_n = t_n\}$. Если A – терм, то результатом подстановки является $A\Theta$. Применение подстановки заключается в замене всех вхождений переменной x_i на соответствующий терм t_i .

Терм B является примером терма A , если существует подстановка Θ , такая что $B = A\Theta$. Терм C называется общим примером термов A и B , если существуют такие подстановки Θ_1 и Θ_2 , что $C = A\Theta_1$ и $C = B\Theta_2$.

1.6 Алгоритм унификации

Алгоритм унификации основной шаг доказательства. С помощью данного алгоритма происходит:

1. двунаправленная передача параметров процедурам,
2. неразрушающее присваивание,
3. проверка условий.

При унификации двух термов Θ_1 и Θ_2 возможны следующие случаи:

- если Θ_1 и Θ_2 константы, и они совпадают, то унификация успешна;
- если Θ_1 не конкретизированная переменная, а Θ_2 – константа, или составной терм, не содержащий в качестве аргумента Θ_1 , то унификация успешна, а Θ_1 конкретизируется значением Θ_2 ;
- если Θ_1 и Θ_2 не конкретизированные переменные, то их унификация всегда успешна, причем и становятся сцепленными, а при конкретизации

тизации одной из переменных, другая конкретизируется автоматически тем же значением.

- если Θ_1 и Θ_2 составные термы, то они успешно унифицируются если выполнены следующие условия:

- Θ_1 и Θ_2 имеют одинаковые главные функторы;
- Θ_1 и Θ_2 имеют равные аргументы;
- каждая пара соответствующих аргументов успешно унифицируется.

1.7 Наиболее общий унификатор

Терм S является более общим, чем терм T , если T является примером S , а S не является примером T . Терм S наиболее общий пример термов T_1 и T_2 , если S такой их общий пример, который является более общим по отношению к любому другому их примеру. Унификатор двух термов – подстановка, которая будучи применена к каждому терму даст одинаковый результат. Наиболее общим унификатором двух термов называется унификатор, соответствующий наиболее общему примеру термов.

1.8 Порядок работы

Работа начинается с задания вопроса. Сверху вниз система просматривает базу знаний и пытается унифицировать вопрос с заголовком правила. Возможна неудача и успех. В случае неудачи, система пытается унифицировать вопрос с заголовком следующего правила. Если унификация прошла успешно, то результатом является флаг и наибольший общий унификатор.

1.9 Резольвента

На каждом шаге доказательства имеется конъюнкция целей, называемая резольвентой. Если резольвента пуста, то достигнут однократный успех. Преобразование резольвенты происходит с помощью редукции.

Редукцией цели G с помощью программы P называется замена цели G телом того правила из P , заголовок которого унифицируется с целью G . Такие правила, заголовки которых унифицируются с целью, называются сопоставимыми с целью.

Новая резольвента получается в два этапа:

1. в текущей резольвенте выбирается одна из целей, и для нее выполняется редукция;
2. в полученной конъюнкции целей применяется подстановка, полученная как наибольший общий унификатор цели и заголовка, сопоставленного с ней правила.

1.10 Список

Список в языке Пролог организуется при помощи составного терма с главным функтором `'.'` и аргументностью равной двум, и с помощью специального терма `[]`, представляющим собой пустой список. Список целых чисел от 1 до 3 выглядит следующим образом `'.'(1, '.'(2, '.'(3, [])))`. Для упрощения был введен дополнительный синтаксис. Выражение `[1,2,3]` представляет собой «синтаксический сахар» для представления списка.

2 Технологический раздел

В качестве языка программирования был выбран язык Haskell.

2.1 Синтаксический анализатор

Парсер выполняет преобразование текста, записанного на языке Пролог, во внутреннее представление, удобное для дальнейшей работы. Синтаксический анализатор разработан с использованием библиотеки `parsec`.

Для разработки синтаксического анализатора требовалось описать грамматику языка при помощи типов.

Листинг 2.1 — Описание программы

```
1 data Program
2   = Program [Clause] Question
3   deriving (Show)
4
5 data Clause
6   = Fact Term
7   | Rule Term [Term]
8   deriving (Show, Eq)
9
10 data Term
11   = AtomTerm Atom
12   | NumberTerm Number
13   | VariableTerm Variable
14   | CompoundTerm Atom [Term]
15   | Cut
16   deriving (Show, Eq)
17
18 data Variable
19   = Named String
20   | Anonymous
21   deriving (Show, Eq)
22
23 data Atom
24   = Symbolic String
25   deriving (Show, Eq)
26
27 data Number
28   = Int Int
29   | Float Float
30   deriving (Show, Eq)
```

```
31
32 type Question = [Term]
```

Чтобы разобрать текст программы, необходимо описать, как разбирать любую его структуру. Например, при помощи пакета `parsec`, чтобы проверить, является ли строка числом, необходимо написать следующий код, представленный в листинге 2.2.

Листинг 2.2 — Парсер целого числа

```
1
2 minus :: Parser String
3 minus = char '-' <:> number
4
5 number :: Parser String
6 number = many1 digit
7
8 integer :: Parser String
9 integer = plus <|> minus <|> number
```

В листинге 2.3 представлен код, необходимый, чтобы получить число, как описанный в листинге 2.2 тип.

Листинг 2.3 — Парсер структуры языка

```
1 parseInt :: Parser Number
2 parseInt = do
3   a <- integer
4   return . Int $ read a
5
6 parseFloat :: Parser Number
7 parseFloat = do
8   a <- integer
9   b <- decimal
10  c <- exponent
11  return $ Float $ read (a ++ b ++ c)
12 where
13   decimal = char '.' <:> number
14   exponent = option "" $ oneOf "eE" <:> integer
15
16 parseNumber' :: Parser Number
17 parseNumber' = try parseFloat <|> parseInt
```

Подобные действия необходимо выполнить для каждого типа.

2.2 Алгоритм унификации

Алгоритм унификации на языке Haskell представлен в листинге 2.4.

Листинг 2.4 — Алгоритм унификации

```
1 updateResult t p results = concat <$> mapM f results
2   where
3     f x@(t' := p') =
4       let p'' = replaceOccurrence t p p'
5       in if p' == p'' then Just [x] else unification [VariableTerm t' :=
6         p''] []
7
8 replaceOccurrence :: Variable -> Term -> Term -> Term
9 replaceOccurrence t q (CompoundTerm f args) = CompoundTerm f (map update
10   args)
11   where update = replaceOccurrence t q
12
13 replaceOccurrence t q p@(VariableTerm p') = if t == p' then q else p
14 replaceOccurrence _ _ p = p
15
16 unificateTerms :: Term -> Term -> Maybe (Target, Maybe OneSubstitution)
17 unificateTerms t p = case (t, p) of
18   (VariableTerm Anonymous, _ ) -> Just ([], Nothing)
19   (VariableTerm x , ConstTerm _ ) -> Just ([], Just (x := p))
20   (VariableTerm x , VariableTerm y ) -> Just ([], Just (x := p))
21   (VariableTerm x , CompoundTerm f args) -> case find (t ==) args of
```

Функция принимает в качестве аргументов два термина. Возвращаемым значением, в случае успешной унификации, будет новая цель или подстановка.

2.3 Резольвента

Резольвента описывается при помощи типа данных, приведенной в листинге 2.5.

Листинг 2.5 — Резольвента

```
1 data Resolvent
2   = Resolvent (Maybe Term) [Term] Resolvent
3   | EmptyResolvent
```

Она представляет собой список, каждый элемент которого хранит два значения: заголовок и тело правила. Необходимость хранения

заголовка обусловлено работой алгоритма отсечения. Заголовка может не быть, в случае если это вопрос.

2.4 Доказательство

В листинге 2.6 показан шаг обработки текущей цели.

Листинг 2.6 — Доказательство

```

1 search '
2   :: [Syntax.Clause]
3   -> Resolvent
4   -> Substitution
5   -> PrologM (Cutting, [SearchTree])
6 search ' _ EmptyResolvent substitution =
7   return (Nothing, [Ok Nothing substitution])
8 search ' sclauses (Resolvent func [] resolvent) substitution =
9   search ' sclauses resolvent substitution
10 search ' sclauses (Resolvent func (Cut : rest) resolvent) substitution = do
11   (_, branches) <- search ' sclauses (Resolvent func rest resolvent)
12   substitution
13   return (func, branches)
14 search ' sclauses (Resolvent func (term : rest) resolvent) substitution = do
15   let resolvent' = (Resolvent func rest resolvent)
16   case term of
17     CompoundTerm "is" [_, _] -> isHandler term sclauses resolvent'
18     substitution
19     CompoundTerm "<" [_, _] ->
20       boolHandler term sclauses resolvent' substitution
21     CompoundTerm ">" [_, _] ->
22       boolHandler term sclauses resolvent' substitution
23     CompoundTerm "<=" [_, _] ->
24       boolHandler term sclauses resolvent' substitution
25     CompoundTerm ">=" [_, _] ->
26       boolHandler term sclauses resolvent' substitution
27     CompoundTerm "==" [_, _] ->
28       boolHandler term sclauses resolvent' substitution
29     CompoundTerm "\\=" [_, _] ->
30       boolHandler term sclauses resolvent' substitution
31     CompoundTerm "=" [_, _] ->
32       explicitUnification term sclauses resolvent' substitution
33     CompoundTerm "trace" [x] ->
34       traceHandler term sclauses resolvent' substitution
35     _ ->
36       defaultHandler term sclauses (Resolvent func rest resolvent)
37       substitution

```

В первую очередь проверяется резольвента. Если она пуста, то возвращается выработанная подстановка. Если все цели из текущего правила закончились, то переход к целям из тела следующего правила. Если встречено отсечение, то продолжается поиск решений в данной ветке, но вместе с найденными результатами возвращается метка отсечения. Далее вызывается обработчик для текущего терма. Стандартный обработчик, представлен в листинге 2.7.

Листинг 2.7 — Доказательство

```

1 defaultHandler
2   :: Term
3   -> [Syntax.Clause]
4   -> Resolvent
5   -> Substitution
6   -> PrologM (Cutting, [SearchTree])
7 defaultHandler term sclauses resolvent substitution = do
8   clauses <- mapM semanticsClause_ sclauses
9   let
10    unifications = zip (unificateClausesTerm clauses term substitution)
11                      clauses
12    f (Nothing, Rule p _) = return (Nothing, Fail (Just $ p :? term))
13    f (Nothing, Fact p) = return (Nothing, Fail (Just $ p :? term))
14    f (Just (func', terms', substitution'), _) = do
15      let resolvent' = Resolvent (Just func') terms' resolvent
16          resolvent'' = updateResolvent resolvent' substitution'
17      (cutted, branches) <- search' sclauses resolvent'' substitution'
18      return
19        (cutted, Node (Just $ term :? func') substitution' resolvent''
20                     branches)
21  branches <- mapM f unifications
22  let
23    cutInfo = termInfo term
24    branches' = mapM snd $ takeWhile' check branches
25    check (x, _) = fmap termInfo x /= Just cutInfo
26    needCut = if length branches /= length branches' then Just term else
27              Nothing
28  return (needCut, branches')
```

В данном обработчике выполняется попытка унификации текущей цели со всеми возможными правилами. В случае успешной унификации происходит преобразование резольвенты для каждой отдельной

ветки. Если в какой-то момент встречается метка отсечения, и она соответствует текущему терму, то следующие ветки в списке отбрасываются.

2.5 Последовательный поиск решение

Последовательный поиск решений реализуется автоматически, благодаря ленивой природе вычислений в языке Haskell. Если необходимо только первое решение, то будет найдено только оно, и вычисления прекратятся. Если потребуются следующие решение, то вычисления продолжатся.

2.6 Пример работы программы

Для программы нахождения факториала, приведенной в листинге 2.8, построено дерево, изображенное на рисунке .

Листинг 2.8 — Факториал

```
1 clauses
2
3 f(1,1) :- !.
4 f(N,X) :- M is N - 1, f(M,Y), X is Y * N.
5
6 goal
7
8 f(2,X).
```

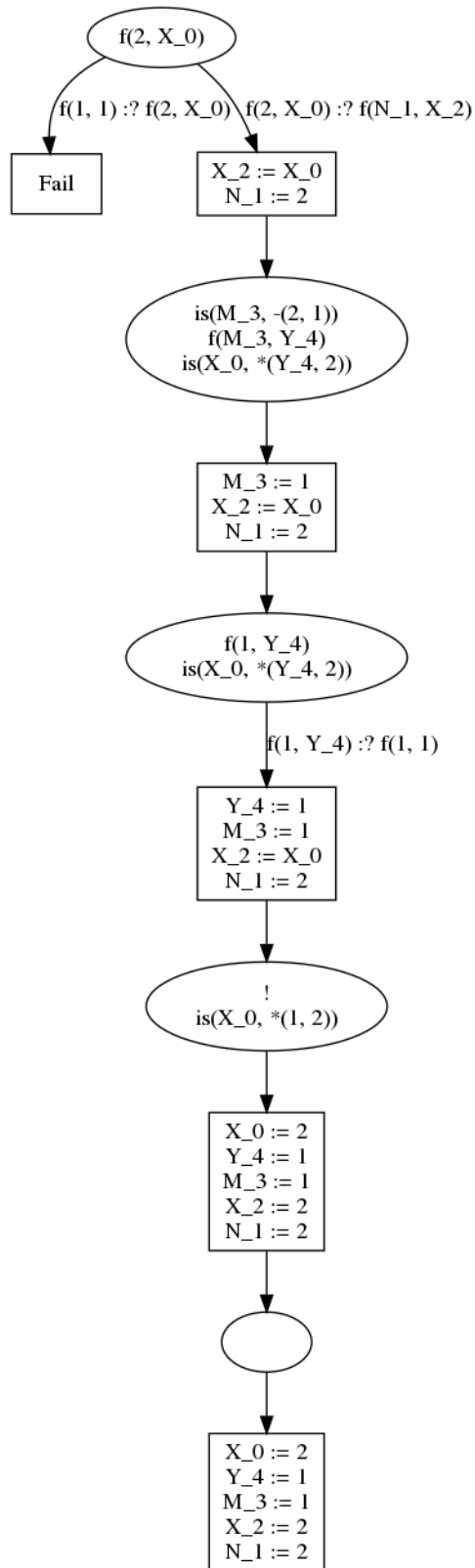


Рисунок 2.1 — Дерево поиска решения для факториала

Для программы нахождения совершенных чисел, приведенной в листинге 2.9, дерево построено быть не может, т.к. программа может выполняться бесконечно, но решения могут быть найдены последовательно.

Листинг 2.9 — Совершенные числа

```

1 clauses
2
3 ints(0).
4 ints(X) :- ints(Y), X is Y + 1.
5
6 perfect_number(X) :- ints(X), Y is X - 1, calculatesum_divisors_till(Sum,
    X, Y), Sum = X.
7
8 calculatesum_divisors_till(0, _, 0).
9 calculatesum_divisors_till(Sum, NumberToDivide, Till) :- Till > 0,
10     Rem is NumberToDivide mod Till, Rem = 0, Ts is Till - 1,
11     calculatesum_divisors_till(SumPrev, NumberToDivide, Ts),
12     Sum is SumPrev + Till.
13
14 calculatesum_divisors_till(Sum, NumberToDivide, Till) :- Till > 0,
15     Rem is NumberToDivide mod Till, Rem > 0, Ts is Till - 1,
16     calculatesum_divisors_till(Sum, NumberToDivide, Ts).
17
18
19 goal
20
21 perfect_number(X).
```

Были найдены первые два решения, 6 и 28. Следующее решение, 496, не было найдено, т.к. требовало очень много времени.

Дерево для быстрой сортировки даже 3 элементов получается слишком большим, чтобы была возможность его вставить сюда. Тем не менее, для программы, приведенной в листинге 2.10, результат сортировки верный.

Листинг 2.10 — Быстрая сортировка

```

1 clauses
2
3 append([], List2, List2).
4 append([Head|Tail], List2, [Head|TailResult]) :-
5     append(Tail, List2, TailResult).
6
7 divide([], _, [], []) :- !.
8 divide([Head|Tail], Pivot, [Head|GreaterList], SmallerList) :- Head >=
9     Pivot,
10     !,
11     divide(Tail, Pivot, GreaterList, SmallerList).
```



```

12         divide(Tail , Pivot , GreaterList , SmallerList) .
13
14     qsort ( [] , [] ) .
15     qsort ([Elem] , [Elem] ) .
16     qsort ([Pivot | Tail] , SortedList) :-
17         divide(Tail , Pivot , GreaterList , SmallerList) ,
18         qsort ( GreaterList , SortedGreaterList ) ,
19         qsort ( SmallerList , SortedSmallerList ) , ! ,
20         append ( SortedSmallerList , [Pivot | SortedGreaterList] , SortedList ) .
21
22
23 goal
24
25     qsort ([1 , 2 , 3 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1] , X) .

```

Заключение

В результате проделанной работы, был разработан синтаксический анализатор языка Пролог, реализованы алгоритм унификации и алгоритм поиска решений, разработан пошаговый поиск решений.

Список использованных источников

1. Изучаем Haskell / Мена А.С. – Санкт-Петербург: Питер, 2015. – 464 pp.
2. Создаём парсер для ini-файлов на Haskell [Электронный ресурс]. Режим <https://habr.com/post/50337/>, свободный.
3. ISO Prolog: A Summary of the Draft Proposed Standard [Электронный ресурс]. Режим доступа: <http://fsl.cs.illinois.edu/images/9/9c/PrologStandard.pdf>, свободный.
4. parsec: Monadic parser combinators [Электронный ресурс]. Режим доступа: <http://hackage.haskell.org/package/parsec>, свободный.