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»

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент Анисимов Никита Сергеевич
(фамилия, имя, отчество)

Группа ИУ7-72

Тип практики производственная

Название предприятия МГТУ им. Баумана

Студент _____ Анисимов Н.С.
(Подпись, дата)

Руководитель _____ Толпинская Н.Б.
(Подпись, дата)

Оценка _____

Москва 2018

Содержание

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.

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 $f(t1,t2..tn)$, где $t1,t2..tn$ – 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 уникальная 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 умещается в одно предложение.

1.5 Подстановка

Подстановкой называется множество пар вида $\{x_i = t_i\}$, где t_i это термы не содержащие переменных. Пусть $\Theta = \{x_1 = t_1, x_2 = t_2 ... x_n = t_n\}$. 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 существует 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 конкре-

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 арности;
  - 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

Работы 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 успешна, 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 арностью 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 упрощение 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 использование библиотеки `parsec`.

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

Листинг 2.1 — Описание программы

```haskell
data Program
  = Program [Clause] Question
  deriving (Show)

data Clause
  = Fact Term
  | Rule Term [Term]
  deriving (Show, Eq)

data Term
  = AtomTerm Atom
  | NumberTerm Number
  | VariableTerm Variable
  | CompoundTerm Atom [Term]
  | Cut
  deriving (Show, Eq)

data Variable
  = Named String
  | Anonymous
  deriving (Show, Eq)

data Atom
  = Symbolic String
  deriving (Show, Eq)

data Number
  = Int Int
  | Float Float
  deriving (Show, Eq)
```

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

Чтобы разобрать текст программы, необходимо описать, как разобрать 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  unificateTerms :: Term -> Term -> Maybe (Target, Maybe OneSubstitution)
2  unificateTerms t p = case (t, p) of
3    (VariableTerm Anonymous, _                    ) -> Just ([], Nothing)
4    (VariableTerm x        , ConstTerm _          ) -> Just ([], Just (x := p))
5    (VariableTerm x        , VariableTerm y       ) -> Just ([], Just (x := p))
6    (VariableTerm x        , CompoundTerm f args) -> case find (t ==) args of
7      Just _  -> Nothing
8      Nothing -> Just ([], Just (x := p))
9    (_, VariableTerm _) -> unificateTerms p t
10   (CompoundTerm f1 args1, CompoundTerm f2 args2) ->
11     if f1 == f2 && length args1 == length args2
12       then Just (zipWith (:?) args1 args2, Nothing)
13       else Nothing
14   (Cut, _  ) -> error "cut unification"
15   (_  , Cut) -> error "cut unification"
16   (t  , p  ) -> if t == p then Just ([], Nothing) else Nothing
```

Функция принимает в качестве аргументов два терма. 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. 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

```
search'
  :: [Syntax.Clause]
  -> Resolvent
  -> Substitution
  -> PrologM (Cutting, [SearchTree])
search' _ EmptyResolvent substitution =
  return (Nothing, [Ok Nothing substitution])
search' sclauses (Resolvent func [] resolvent) substitution =
  search' sclauses resolvent substitution
search' sclauses (Resolvent func (Cut : rest) resolvent) substitution = do
  (_, branches) <- search' sclauses (Resolvent func rest resolvent)
      substitution
  return (Just Cutted, branches)
search' sclauses (Resolvent func (term : rest) resolvent) substitution = do
  let resolvent' = (Resolvent func rest resolvent)
  case term of
    CompoundTerm "is" [_, _] -> isHandler term sclauses resolvent'
        substitution
    CompoundTerm "<" [_, _] ->
      boolHandler term sclauses resolvent' substitution
    CompoundTerm ">" [_, _] ->
      boolHandler term sclauses resolvent' substitution
    CompoundTerm "=<" [_, _] ->
      boolHandler term sclauses resolvent' substitution
    CompoundTerm ">=" [_, _] ->
      boolHandler term sclauses resolvent' substitution
    CompoundTerm "==" [_, _] ->
      boolHandler term sclauses resolvent' substitution
    CompoundTerm "\\=" [_, _] ->
      boolHandler term sclauses resolvent' substitution
    CompoundTerm "=" [_, _] ->
      explicitUnification term sclauses resolvent' substitution
    CompoundTerm "trace" [x] ->
      traceHandler term sclauses resolvent' substitution
    _ ->
      defaultHandler term sclauses (Resolvent func rest resolvent)
          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

```
defaultHandler
  :: Term
  -> [Syntax.Clause]
  -> Resolvent
  -> Substitution
  -> PrologM (Cutting, [SearchTree])
defaultHandler term sclauses resolvent substitution = do
  clauses <- mapM semanticsClause_ sclauses
  let
    unfications = zip (unificateClausesTerm clauses term substitution)
        clauses
    f (Nothing, Rule p _) = return (Nothing, Fail (Just $ p :? term))
    f (Nothing, Fact p) = return (Nothing, Fail (Just $ p :? term))
    f (Just (func', terms', substitution'), _) = do
      let resolvent' = Resolvent (Just func') terms' resolvent
          resolvent'' = updateResolvent resolvent' substitution'
      (cutted, branches) <- search' sclauses resolvent'' substitution'
      return
        (cutted, Node (Just $ term :? func') substitution' resolvent''
            branches)
  branches <- mapM f unfications
  let
    branches' = takeWhile' check branches
    check (x, _) = x == Nothing
    cut = (fst . last $ branches')
  return (cut, map snd 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 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(10,X).
```

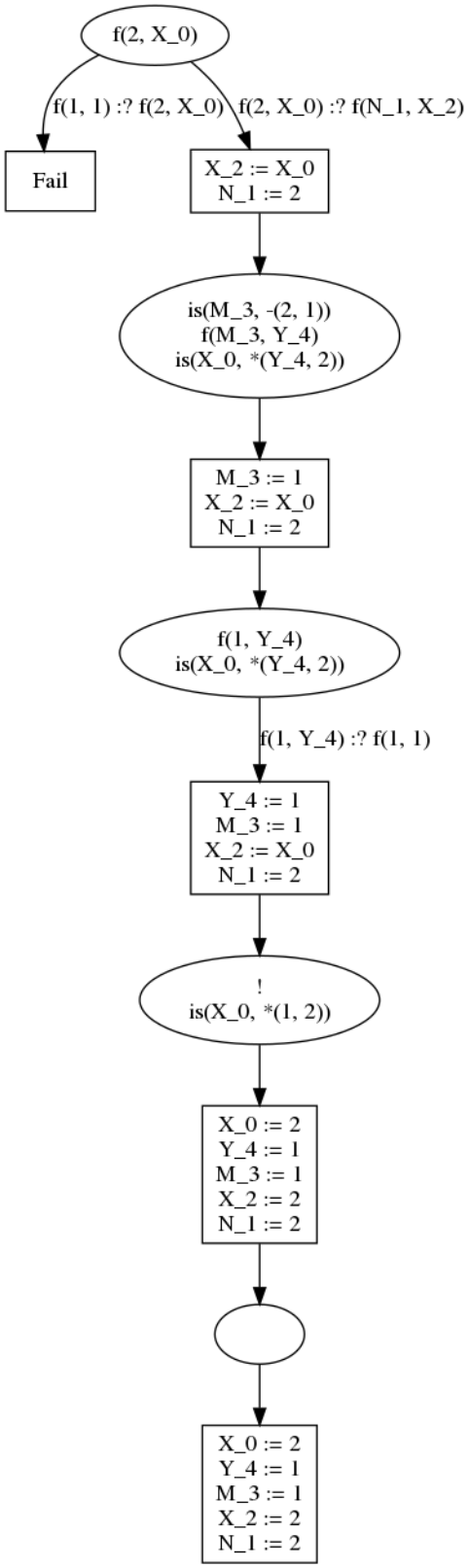

Рисунок 2.1 — Дерево поиска решения для факториала

Дерево для быстрой сортировки даже 3 элементов получается слишком большим, чтобы была возможность его вставить сюда. Тем не

менее, для программы, приведенной в листинге 2.9, результат сортировки верный.

Листинг 2.9 — Быстрая сортировка

```
clauses

  append([], List2, List2).
    append([Head|Tail], List2, [Head|TailResult]):-
        append(Tail, List2, TailResult).

  divide([], _, [], []):-!.
    divide([Head|Tail], Pivot, [Head|GreaterList], SmallerList):-Head >
        Pivot,
          !,
        divide(Tail, Pivot, GreaterList, SmallerList).
    divide([Head|Tail], Pivot, GreaterList, [Head|SmallerList]):-
        divide(Tail, Pivot, GreaterList, SmallerList).

    qsort([], []).
    qsort([Elem], [Elem]).
    qsort([Pivot|Tail], SortedList):-
        divide(Tail, Pivot, GreaterList, SmallerList),
        qsort(GreaterList, SortedGreaterList),
        qsort(SmallerList, SortedSmallerList),!,
        append(SortedSmallerList, [Pivot|SortedGreaterList], SortedList).


goal

  qsort([44, 15, 62, 43, 28, 93, 76, 21, 43, 37], X).
```

Заключение

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