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

к курсовой работе на тему:

«Система тестирования»

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

Руководитель _____ Строганов Ю.В.
(Подпись, дата)

Москва 2018

Содержание

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 возможности пошагового поиска решений;
- Изображение дерева 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

Для разработки синтаксического анализатора требовалось описать все возможные структуры языка Пролог.

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)

type Question = [Term]
```

Чтобы разобрать текст программы, необходимо описать, как разобрать любую его структуру. Например, при помощи пакета `parsec`, что-

бы проверить, является ли строка числом, необходимо написать следующий код, представленный в листинге 2.2.

Листинг 2.2 — Парсер целого числа

```
plus :: Parser String
plus = char '+' >> number

minus :: Parser String
minus = char '-' <:> number

number :: Parser String
number = many1 digit

integer :: Parser String
integer = plus <|> minus <|> number
```

В листинге 2.3 представлен код, необходимый, чтобы получить число, как описанную ранее структуру данных.

Листинг 2.3 — Парсер структуры языка

```
parseInt :: Parser Number
parseInt = do
  a <- integer
  return . Int $ read a

parseFloat :: Parser Number
parseFloat = do
  a <- integer
  b <- decimal
  c <- exponent
  return $ Float $ read (a ++ b ++ c)
 where
  decimal  = char '.' <:> number
  exponent = option "" $ oneOf "eE" <:> integer

parseNumber' :: Parser Number
parseNumber' = try parseFloat <|> parseInt
```

Подобные действия необходимо выполнить для каждой структуры данных.

2.2 Алгоритм унификации

Алгоритм унификации на языке Haskell представлен в листинге 2.4.

Листинг 2.4 — Алгоритм унификации

```
1  (VariableTerm x          , ConstTerm _          ) -> Just ([], Just (x := p))
2  (VariableTerm x          , VariableTerm y       ) -> Just ([], Just (x := p))
3  (VariableTerm x          , CompoundTerm f args) -> case find (t ==) args of
4    Just _  -> Nothing
5    Nothing -> Just ([], Just (x := p))
6  (_, VariableTerm _) -> unificateTerms p t
7  (CompoundTerm f1 args1, CompoundTerm f2 args2) ->
8    if f1 == f2 && length args1 == length args2
9      then Just (zipWith (:?) args1 args2, Nothing)
10     else Nothing
11 (Cut, _  ) -> error "cut unification"
12 (_  , Cut) -> error "cut unification"
13 (t  , p  ) -> if t == p then Just ([], Nothing) else Nothing
```

На вход функции подается два терма. В качестве возвращаемого значения, в случае успешной унификации будет возвращена новая цель (если унифицировались составные термы), и возможная подстановка (подстановка возможная, т.к. в случае унификации двух констант или составных термов, подстановка выработана не будет).

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 первоначальный 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 (func, 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 происходит сравнение текущего терма с другими стандартными термами. Если все сравнения прошли неудачно, то вызывается стандартный обработчик, представленный 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     unfications = zip (unificateClausesTerm clauses term substitution)
           clauses
11     f (Nothing, Rule p _) = return (Nothing, Fail (Just $ p :? term))
12     f (Nothing, Fact p) = return (Nothing, Fail (Just $ p :? term))
13     f (Just (func', terms', substitution'), _) = do
14       let resolvent'  = Resolvent (Just func') terms' resolvent
15           resolvent'' = updateResolvent resolvent' substitution'
16       (cutted, branches) <- search' sclauses resolvent'' substitution'
17       return
18         (cutted, Node (Just $ term :? func') substitution' resolvent''
               branches)
19   branches <- mapM f unfications
20   let
21     cutInfo   = termInfo term
22     branches' = map snd $ takeWhile' check branches
23     check (x, _) = fmap termInfo x /= Just cutInfo
24     needCut = if length branches /= length branches then Just term else
           Nothing
25   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 потребуется следующее решение, то вычисления продолжатся.

2.6 Пример работы программы

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

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

Заключение

В результате проделанной работы, было реализовано консольное приложение, с помощью которого можно выполнять Пролог программы, строить дерево поиска решений, и выполнять последовательный поиск решений.

Список использованных источников

1. Введение в системы баз данных: учебное пособие / Бураков П.В., Петров В.Ю. – Санкт-Петербург, 2010 – 128с.

2. Назначение и основные понятия БД. [Электронный ресурс] Режим доступа: http://life-prog.ru/1_32727_naznachenie-i-osnovnie-ponyatiya-bd.html , свободный.

3. Базы данных SQL, NoSQL и различия в моделях баз данных [Электронный ресурс]. Режим доступа: http://devacademy.ru/posts/sql-nosql/, свободный.

4. Информационные системы: Учебник для вузов. /Избачков Юрий Сергеевич, Петров Владимир Николаевич, Васильев Александр Алексеевич, Телина Ирина Сергеевна. — 3-е изд. — Питер, 2010. — 544с.

5. SQL или NoSQL — вот в чём вопрос [Электронный ресурс]. Режим доступа: https://habr.com/company/ruvds/blog/324936/, свободный.

6. SQL и NoSQL: разбираемся в основных моделях баз данных [Электронный ресурс]. Режим доступа: https://tproger.ru/translations/sql-nosql- database-models/, свободный.

7. О сервисе Яндекс.Контест. [Электронный ресурс] Режим доступа: https://contest.yandex.ru/about/ , свободный.

8. О сервисе Сертификация Mail.ru. [Электронный ресурс] Режим доступа: https://certification.mail.ru/about/ , свободный.

9. О сервисе Stepik URL. [Электронный ресурс] Режим доступа: https://welcome.stepik.org/ru/about , свободный.

10. Бэнкер Кайл MongoDB в действии /Бэнкер Кайл. — 1-е изд. — Москва: ДМК-пресс, 2017. — 394с.

11. Изучаем Haskell /Мена А.С. – Санкт-Петербург: Питер, 2015. – 464 pp.

12. servant: A family of combinators for defining webservices APIs [Электронный ресурс]. Режим доступа:

https://hackage.haskell.org/package/servant, свободный.

13. Persistent и работа с базами данных в Yesod [Электронный ресурс]. Режим доступа: https://eax.me/yesod-persistent/, свободный.

14. An Introduction to Elm. [Электронный ресурс]. Режим доступа: https://guide.elm-lang.org/, свободный.