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 Основная часть

1.1 Характеристика предприятия

Московский государственный технический университет им. Н. Э. Баумана – российский национальный исследовательский университет, научный центр, особо ценный объект культурного наследия народов России.

Информация о кафедре ИУ-7.

— Заведующий кафедрой: к.т.н., доцент Рудаков Игорь Владимирович

— Год создания: 1989

— Назначение кафедры: Готовит специалистов широкого профиля в области проектирования и разработки программного обеспечения. С 2011 года выпускает бакалавров и магистров по направлениям подготовки 09.03.04 и 09.04.04 "Программная инженерия"

Основные направления обучения

— Программная инженерия: Принципы и методы проектирования и разработки информационных систем.

— Системное программирование: Низкоуровневое программирование, разработка драйверов устройств, программирование в режиме ядра ОС, вопросы проектирования ОС.

— Конструирование компиляторов: Теория формальных языков и практика создания компиляторов.

— Программирование баз данных: Математические основы баз данных, проектирование и разработка ПО, использующего базы данных.

— Сетевое программирование: Изучение сетевых протоколов, создание собственных реализаций сетевых стандартов, создание новых протоколов.

— Машинная и инженерная графика: Реализация алгортимов компьютерной графики, создание фотореалистичных изображений.

— Компьютерное моделирование: Моделирование непрерывных и дискретных систем, численные методы.

— Интеллектуальные системы: Математические основы и реализация экспертных систем, систем принятия решений, систем обработки естественного языка.

— Библиотечные информационные системы: Проектирование и разработка информационно-поисковых систем, классификация информации.

1.2 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.

1.3 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.

1.4 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.

1.5 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.

1.6 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 $C = B\Theta_2$.

1.7 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 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.

1.8 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.

1.9 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.

1.10 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.

1.11 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.

1.12 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 1.1 — 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 разобрать любую его структуру. Например, при помощи пакета `parsec`, чтобы проверить, является ли строка числом, необходимо написать следующий код, представленный в листинге 1.2.

Листинг 1.2 — 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 1.3 представлен код, необходимый, чтобы получить число, как описанный в листинге 1.2 тип.

Листинг 1.3 — 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.

1.13 Алгоритм унификации

Алгоритм унификации на языке Haskell представлен в листинге 1.4.

Листинг 1.4 — Алгоритм унификации

```
    Nothing -> Just ([], Just (x := p))
  (_, VariableTerm _) -> unificateTerms p t
  (CompoundTerm f1 args1, CompoundTerm f2 args2) ->
    if f1 == f2 && length args1 == length args2
      then Just (zipWith (:?) args1 args2, Nothing)
      else Nothing
  (Cut _, _  ) -> error "cut unification"
  (_  , Cut _) -> error "cut unification"
  (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.

1.14 Доказательство

В листинге 1.5 показан шаг обработки текущей цели.

Листинг 1.5 — Доказательство

```

search'
  :: [Syntax.Clause]
  -> Resolvent
  -> Substitution
  -> PrologM (Cutting, [SearchTree])
search' _ EmptyResolvent substitution =
  return ([], [Ok Nothing substitution])
search' sclauses (Resolvent func [] resolvent) substitution =
  search' sclauses resolvent substitution
search' sclauses (Resolvent func (Cut x : rest) resolvent) substitution =
    do
  let nr  = succCut (Resolvent func rest resolvent)
  (cs, branches) <- search' sclauses nr substitution
  cs' <- return $ case func of
    Just c -> predCut $ (c, x + 1):cs
    Nothing -> predCut $ cs

  return (cs', 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 "<" [_, _] ->
```

```
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        boolHandler term sclauses resolvent' substitution
33      CompoundTerm "\\=" [_, _] ->
34        boolHandler term sclauses 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 1.6.

Листинг 1.6 — Стандартный обработчик

```
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 ([], Fail (Just $ p :? term))
12     f (Nothing, Fact p) = return ([], Fail (Just $ p :? term))
13     f (Just (func', terms', substitution'), _) = do
14       let resolvent'  = Resolvent (Just func') terms' (succCut  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     branches' = takeWhile' check branches
```

```
22      check (cs,_) = find (\(_,x) -> x == 0) cs == Nothing
23      cutters'  = S.fromList (concat (map fst branches'))
24      cutters'' = S.toList cutters'
25      cutters   = let x = sortBy (\(_, x) (_, y) -> compare x y) cutters''
26        in if length x == 0
27          then []
28          else if (snd.head $ x) == 0 then tail x else x
```

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.

1.15 Отсечение

Для работы алгоритма, в качестве хранилища контрольных точек, было выбрано дерево. Такой выбор был сделан для простоты визуализации дерева поиска решений. Позже оказалось, что данная структура не подходит для работы алгоритма. Если отсечения стоит в середине тела правила, то откат может быть совершен в обратную сторону по отсечению. В результате этого корректная работа программы возможно только в случае, если отсечение стоит в конце предложения или если его нет. В противном случае, программа может работать некорректно.

Был сделан вывод, что для дальнейшей разработки лучше использовать стек, который хранит текущую подстановку, резольвенту и возможные варианты ветвления. Но при данной конфигурации значительно усложняется алгоритм построения дерева поиска решений.

1.16 Генерация изображений

Graphviz – пакет утилит по автоматической визуализации графов, заданных в виде описания на языке DOT, а также дополнительных текстовых и графических программ, виджетов и библиотек, используемых при разработке программного обеспечения для визуализации структурированных данных. В данной работе использовался Graphviz для генерации изображний.

1.17 Примеры работы программы

Для программы нахождения факториала, приведенной в листинге 1.7, построено дерево, изображенное на рисунке 1.1.

Листинг 1.7 — 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).
```

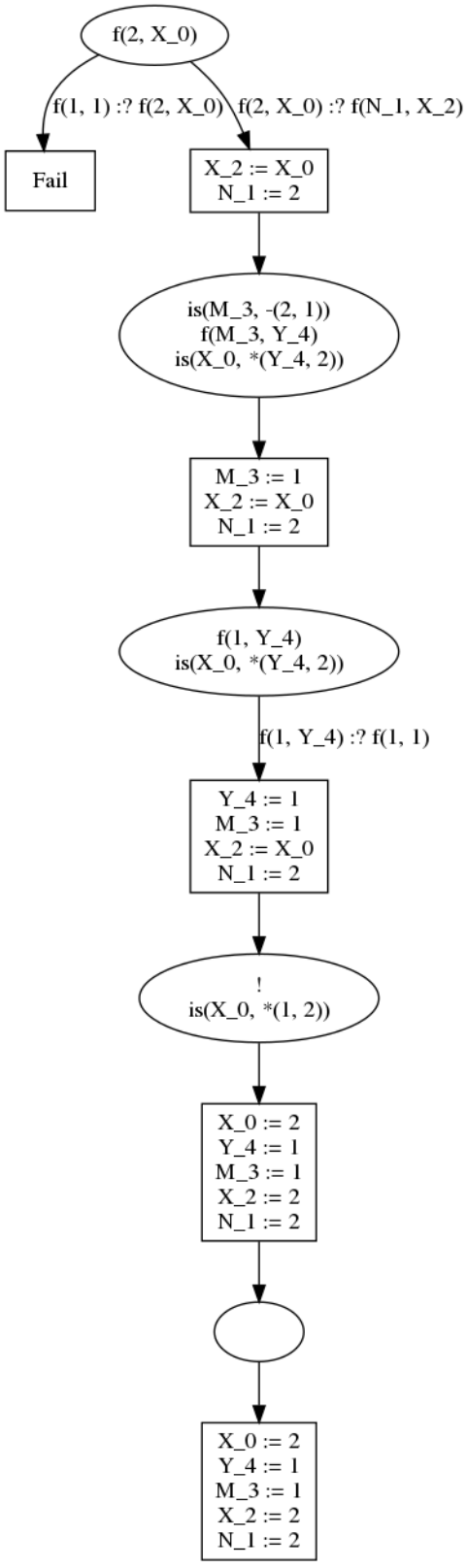

Рисунок 1.1 — Дерево поиска решения для факториала

Для программы соединения двух списков, приведенной в листинге 1.8, построено дерево, изображенное на рисунке 1.2.

Листинг 1.8 — Append

```
clauses

  append([], L, L).
    append([H|T], L, [H|X]):-
    append(T, L, X).

goal

  append([1,2],[3],X).
```

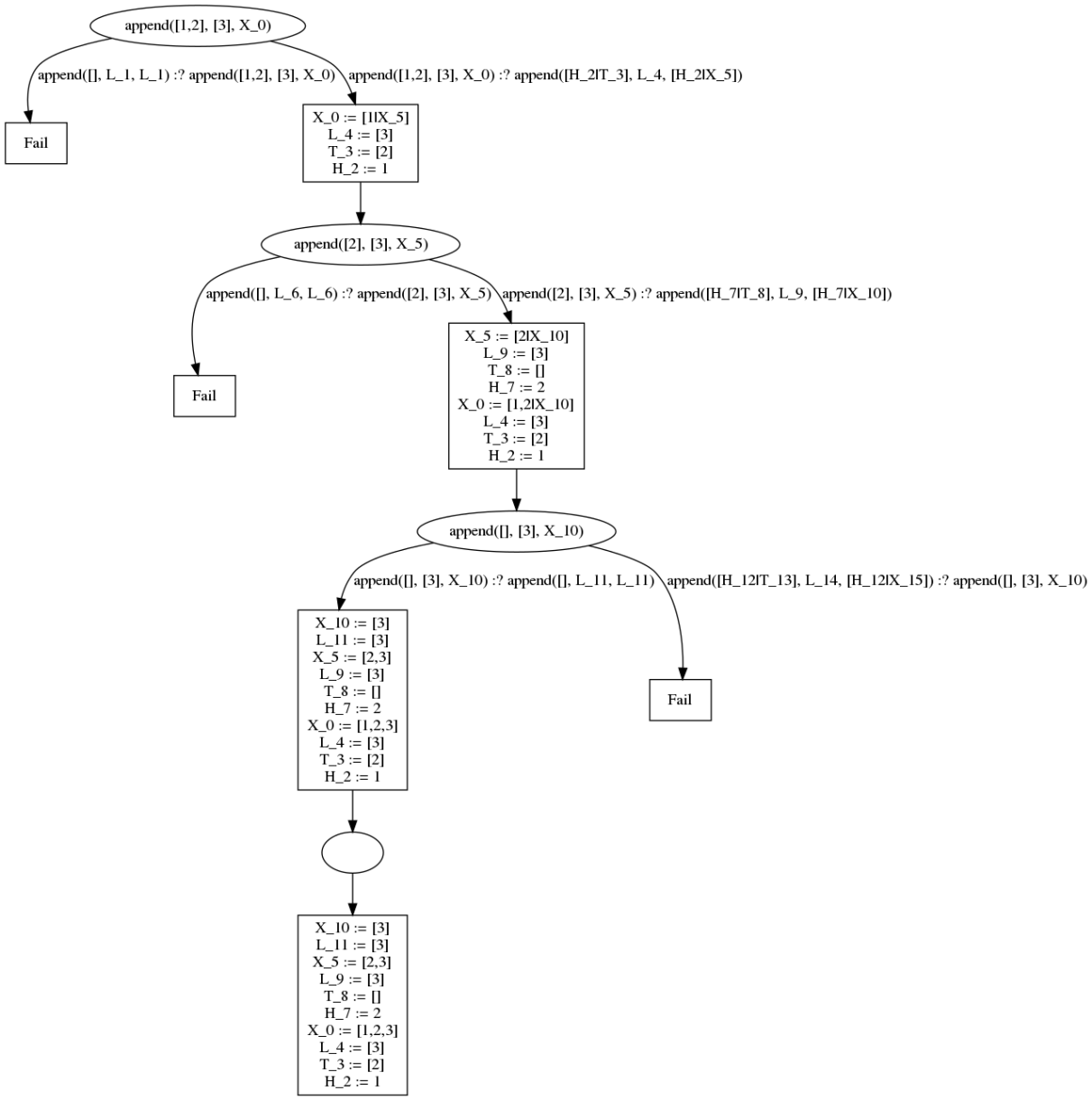

Рисунок 1.2 — Дерево поиска решения для соединения списков

Заключение

В результате проделанной работы, был разработан синтаксический анализатор языка Пролог, реализованы алгоритм унификации и частично алгоритм поиска решений. Алгоритм поиска решений работает корректно, только в случае если отсечения нет, или оно расположено в конце тела правила.

Список использованных источников

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

2. Основы программирования на языке Пролог. Курс лекций /Павел Шрайнер – Интернет-университет информационных технологий, 2005. – 176 pp.

3. Создаём парсер для ini-файлов на Haskell [Электронный ресурс]. Режим https://habr.com/post/50337/, свободный.

4. ISO Prolog: A Summary of the Draft Proposed Standard [Электронный ресурс]. Режим доступа: http://fsl.cs.illinois.edu/images/9/9c/PrologStandard.pdf, свободный.

5. parsec: Monadic parser combinators [Электронный ресурс]. Режим доступа: http://hackage.haskell.org/package/parsec, свободный.