

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 Р А К Т И К Е

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 | 3 |
| Введение | 4 |
| Цель | 4 |
| Задачи | 4 |
| 1 Основная часть | 5 |
| 1.1 Характеристика предприятия | 5 |
| 1.2 Основные элементы языка Пролог | 6 |
| 1.3 Особенности использования переменных | 6 |
| 1.4 Структура программы | 6 |
| 1.5 Понятие процедуры | 7 |
| 1.6 Подстановка | 7 |
| 1.7 Алгоритм унификации | 7 |
| 1.8 Наиболее общий унификатор | 8 |
| 1.9 Порядок работы | 8 |
| 1.10 Резольвента | 9 |
| 1.11 Список | 9 |
| 1.12 Синтаксический анализатор | 10 |
| 1.13 Алгоритм унификации | 11 |
| 1.14 Доказательство | 12 |
| 1.15 Отсечение | 14 |
| 1.16 Пример работы программы | 14 |
| Заключение | 17 |
| Список использованных источников | 18 |

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 алгоритмов 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 именованной 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.

1.3 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.

1.4 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 уместается в одно предложение.

1.6 Подстановка

Подстановкой называется множество пар вида $\{x_i = t_i\}$, где t_i это термы не содержащие переменных. Пусть $\Theta = \{x_1 = t_1, x_2 = t_2 \dots x_n = t_n\}$. 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 $C = B\Theta_2$.

1.7 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 аргументы;
- 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

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.

1.10 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 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.

1.12 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 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 разбирать 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

```

1 unifyTerms :: Term -> Term -> Maybe (Target, Maybe OneSubstitution)
2 unifyTerms 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 _) -> unifyTerms 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.

1.14 Доказательство

В листинге 1.5 показан шаг обработки текущей цели.

Листинг 1.5 — Доказательство

```

1
2 search '
3   :: [Syntax.Clause]
4   -> Resolvent
5   -> Substitution
6   -> PrologM (Cutting, [SearchTree])
7 search ' _ EmptyResolvent substitution =
8   return ([], [Ok Nothing substitution])
9 search ' sclauses (Resolvent func [] resolvent) substitution =
10  search ' sclauses resolvent substitution
11 search ' sclauses (Resolvent func (Cut x : rest) resolvent) substitution =
12  do
13    let nr = succCut (Resolvent func rest resolvent)
14    (cs, branches) <- search ' sclauses nr substitution
15    cs' <- return $ case func of
16      Just c -> predCut $ (c, x + 1):cs
17      Nothing -> predCut $ cs

```

```

18   return (cs', branches)
19 search' sclauses (Resolvent func (term : rest) resolvent) substitution = do
20   let resolvent' = (Resolvent func rest resolvent)
21   case term of
22     CompoundTerm "is" [_, _] -> isHandler term sclauses resolvent'
23     substitution
24     CompoundTerm "<" [_, _] ->
25     boolHandler term sclauses resolvent' substitution
26     CompoundTerm ">" [_, _] ->
27     boolHandler term sclauses resolvent' substitution
28     CompoundTerm "<=" [_, _] ->
29     boolHandler term sclauses resolvent' substitution
30     CompoundTerm ">=" [_, _] ->
31     boolHandler term sclauses resolvent' substitution
32     CompoundTerm "==" [_, _] ->
33     boolHandler term sclauses resolvent' substitution
34     CompoundTerm "\\=" [_, _] ->
35     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 — Доказательство

```

1 boolHandler bool sclauses resolvent substitution = do
2   let r = evalBool bool
3   case r of
4     Just True -> search' sclauses resolvent substitution
5     _          -> return ([], [])
6
7 defaultHandler
8   :: Term
9   -> [Syntax.Clause]
10  -> Resolvent
11  -> Substitution
12  -> PrologM (Cutting, [SearchTree])
13 defaultHandler term sclauses resolvent substitution = do
14   clauses <- mapM semanticsClause_ sclauses
15   let

```

```

16     unifications = zip (unificateClausesTerm clauses term substitution)
           clauses
17   f (Nothing, Rule p _) = return ([], Fail (Just $ p :? term))
18   f (Nothing, Fact p) = return ([], Fail (Just $ p :? term))
19   f (Just (func', terms', substitution'), _) = do
20     let resolvent' = Resolvent (Just func') terms' (succCut resolvent)
21     resolvent'' = updateResolvent resolvent' substitution'
22     (cutted, branches) <- search' sclauses resolvent'' substitution'
23     return
24       (cutted, Node (Just $ term :? func') substitution' resolvent''
           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.

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 Пример работы программы

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

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).
```

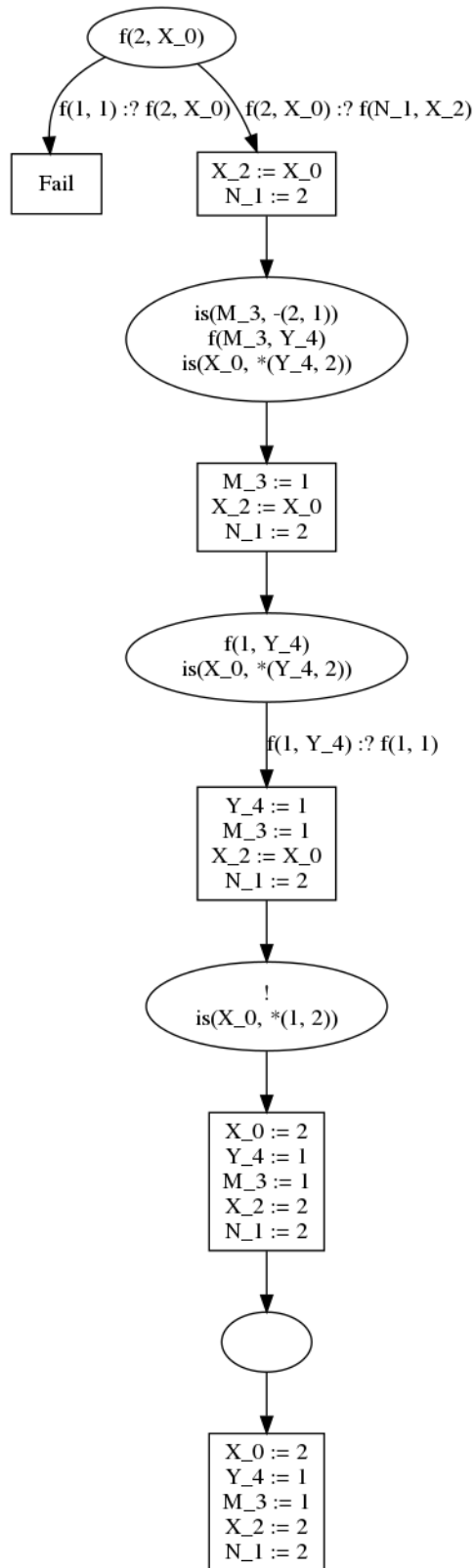


Рисунок 1.1 — Дерево поиска решения для факториала

Заключение

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