Лабораторная работа №1

Тема: экспертные системы

Цель работы: получение практических навыков применения специализированных оболочек для построения продукционных экспертных систем.

Задание: На языке COOL оболочки экспертных систем CLIPS реализовать экспертную систему согласно полученному варианту. Использовать классы, объекты, продукции при написании программы.

Реализовать макет ЭС должен включать в себя:

- Организацию ввода информации (опрос системой пользователя).
- Структуру правил системы (2-3 реально работающих правила, для остальных – описание работы на естественном языке).
- Структуру функций системы (2-3 реально работающих функции, для остальных – описание работы на естественном языке).
- Организацию вывода информации на экран и в файл.

Общее количество правил, функций и хэндлеров в сумме должно составлять не менее 20.

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание
2. Краткое описание предметной области и решаемой задачи (каков результат работы системы, что является входными данными)
3. Описание примененного подхода к извлечению знаний с необходимыми рисунками, диаграммами (деревья И/ИЛИ, деревья решений) и т.д.
4. Полный текст базы знаний и протокол ее работы для трех консультаций

Основные теоретические сведения

Экспертные системы – это сложные программные комплексы, аккомулирующие знания специалистов в конкретных предметных областях и тиражирующие эти знания для консультации менее квалифицированных специалистов.

CLIPS сочетает в себе 3 парадигмы программирования: логическую, процедурную и объектно-ориентированную. Также, в CLIPS предусмотрены 3 основных формата представления информации: факты, глобальные переменные и объекты.

Команды

Команда (run) запускает выполнение программы в среде (с машиной логического вывода), (reset) – обновляет рабочую память, а (clear) – полностью очищает среду CLIPS

(рабочую память). Опция главного меню Load позволяет загрузить файл с конструкциями, а Load Batch – файл с командами (программу).

Главный файл программы на CLIPS (например, main.bat) может иметь следующий вид:

```
(clear)
(load* "classes.clp") ; загрузка структуры классов
(load-instances "instances.clp") ; загрузка экземпляров классов
(load* "templates.clp") ; загрузка предопределённых фактов
(load* "functions.clp") ; загрузка функций
(load* "rules.clp") ; загрузка правил
(reset)
(run)
```

Вывод информации может осуществляться как на экран, так и в выходные файлы.

Например, нижеследующая функция может использоваться для запроса у пользователя значения, присвоения его переменной, а затем вывода его на экран:

```
(deffunction foo
        () ; функция не имеет аргументов
        (printout t crlf "Please input value:" crlf) ; t – ключ для вывода на экран
        (bind ?var (read)) ; bind – функция для присваивания значения
        (printout t crlf "Your value: " ?var crlf) ; crlf – символ конца строки
)
```

В свою очередь, команды (save-facts ?file+name) и (save-instances ?file+name) служат для сохранения всех имеющихся в CLIPS на текущий момент фактов и экземпляров классов соответственно. Значением переменной ?file+name может быть, например, "system_output.clp".

Обратите внимание, что в CLIPS в именах функций для обозначения пробела обычно используется знак «-», а в именах переменных и строковых значениях – знак «+». Имя переменной всегда начинается со знака «?».

Форматы представления данных в CLIPS

Факты являются одной из основных форм высокого уровня для представления информации в системе CLIPS. Факт (fact) – это список элементарных значений, на которые ссылаются либо позиционно (упорядоченные (ordered) факты), либо по имени (неупорядоченные (non-ordered) или шаблонные (template) факты).

Каждый факт представляет часть информации и помещается в текущий список фактов (fact-list). Факты могут быть добавлены в список фактов (используя команду

assert), удалены из него (используя команду retract), изменены (используя команду modify) или скопированы (используя команду duplicate) в результате явного воздействия пользователя или при исполнении программы CLIPS.

Глобальные переменные. Конструкция defglobal позволяет описывать переменные, которые являются глобальными в контексте окружения CLIPS. То есть глобальная переменная доступна в любом месте окружения CLIPS и сохраняет свое значение независимо от других конструкций. Напротив, некоторые конструкции (как, например, defrule или deffunction) могут иметь собственные локальные переменные, которые задаются в пределах описания конструкции. Обращение к таким переменным возможно только изнутри конструкции, где они описаны; за ее пределами они не имеют значения.

Функция bind используется, чтобы задать значения глобальным переменным. Значения глобальных переменных сбрасываются к начальным установочным значениям при выполнении команды окружения reset или если для глобальной переменной вызвана функция bind без соответствующего значения.

Объекты в CLIPS могут быть описаны как символьные, строковые, целые или вещественные числа, значения с множеством полей, внешние адреса или объекты определенного пользователем класса.

Для создания пользовательского класса используется конструкция defclass. Объект пользовательского класса создается посредством функции make-instance, и к созданному таким образом объекту можно обращаться по уникальному адресу. В пределах модульного контекста к объекту можно уникально обращаться и по имени.

```
; описание класса car (машина)
(defclass Car
        (is-a USER) ; пользовательский класс
        (single-slot model+name ; слот «название модели» (одно значение)
                (type STRING)) ; тип данных – строка
        (multislot producer ; слот «производитель» (0 или более значений)
                (type STRING)) ; тип данных – строка
)
; создание экземпляра класса Car
(make-instance Toyota+Corolla of Car
(model+name "Corolla")
(producer "Toyota" "Kanto Auto Works")
)
```

Взаимодействие с классами осуществляется путём отправки им «сообще-ний». Для этого сначала определяется «хэндлер» (по смыслу близок к функциям), через конструкцию defmessage-handler:

(defmessage-handler Guideline print-tag () ; выводит содержимое слота tag класса Guideline

(printout t ?self:tag crlf)

)

Затем в нужном месте программы классу отправляется «сообщение»:

?guideline <- (object (is-a Guideline))

=>

(send ?guideline print-tag ())

Основные конструкторы COOL

Среда CLIPS имеет встроенные средства для осуществления логического вывода, но позволяет программисту управлять очередностью выполнения правил (посредством параметра salience).

Отличительной особенностью CLIPS являются конструкторы для создания баз знаний (БЗ):

defrule *определение правил;*
deffacts *определение фактов;*
deftemplate *определение шаблона факта;*
defglobal *определение глобальных переменных;*
deffunction *определение функций;*
defmodule *определение модулей (совокупности правил);*
defclass *определение классов;*
defintances *определение объектов по шаблону, заданному defclass ;*
defmessagehandler *определение сообщений для объектов;*
defgeneric *создание заголовка родовой функции;*
defmethod *определение метода родовой функции.*

Конструкторы не возвращают никаких значений, в отличии от функций, например:

(deftemplate person
(slot name)
(slot age)
(multislot friends))

Пример функции:

```
(deffunction factorial (?a)
        (if (or (not (integerp ? a)) (< ? a0)) then
                (printout t "Factorial Error!" crlf)
        else
                (if (= ? a0) then
                        1
                else
                        (*? a (factorial ($-$ ? a1))))))
```

Правила в CLIPS состоят из предпосылок и следствия. Предпосылки также называют ЕСЛИ-частью правила, левой частью правила или LHS правила (left-hand side of rule). Следствие называют ТО-частью правила, правой частью правила или RHS правила (right-hand side of rule).

Пример правила представлен ниже:

```
(deftemplate data (slot x) (slot y))
(defrule twice
(data (x ? x) (y =(*2 ? x)))
=>)
(assert (data (x2) (y4)); f-0
        (data (x3) (y9))); f-1
```

Здесь самая распространенная в CLIPS функция assert добавляет новые факты в список правил. В противоположность assert функция retract удаляет факты из списка фактов, например:

```
(defrule vis11
        ?doors < — (fit ? wdfit)
        (test (eq ? wdfit no))
 =>
        (assert (EVIDENCE OF MAJOR ACCIDENT))
        (retract ? doors))
```

В этом правиле проверяется наличие факта doors и в случае его отсутствия факт doors удаляется из списка фактов задачи.

Функция modify является также весьма распространенной. Она позволяет в определенном факте поменять значение слота, например,

(deftemplate age (slot value))

(assert (age (value young)))

(modify 0 (value old))

Следующий пример описывает представление данных в виде фактов, объектов и глобальных переменных. Примеры фактов:

(voltage is 220 volt)

(meeting (subject "AI") (chief "Kuzin") (Room "3240"))

В первой строке приведен упорядоченный факт, во второй - неупорядоченный, в котором порядок слотов не важен.

CLIPS поддерживает следующие типы данных: integer, float, string, symbol, external-address, fact-address, instance-name, instance-address.

Пример integer: 594 23 +51 -17

Пример float: 594e2 23.45 +51.0 -17.5e-5

String — это строка символов, заключенная в двойные кавычки.

Пример string: "expert", "Phil Blake", "состояние -0", "quote=\"

CLIPS поддерживает следующие процедурные функции, реализующие возможности ветвления, организации циклов в программах и т.,п.:

If оператор ветвления;

While цикл с предусловием;

loop-for-count итеративный цикл;

prong объединение действий в одной логической команде;

prong$ выполнение набора действий над каждым элементом поля;

return прерывание функции, цикла, правила и т.д.;

break то же, что и return, но без возвращения параметров;

switch оператор множественного ветвления;

bind создание и связывание переменных.

Среди логических функций (возвращающих значения true или false) следует выделить следующие группы:

- функции булевой логики: *and, or, not*
- функции сравнения чисел: = , >, <
- предикативные функции для проверки принадлежности проверяемому типу: *integerp, floatp, stringp, symbolp, pointerp (относится ли аргумент к xternal-address), numberp (относится ли аргумент к integer или float), lexemerp (относится ли аргумент к string или symbol), evenp (проверка целого на четность), oddp (проверка целого на нечетность), multifildp (является ли аргумент составным полем).*

– Функции сравнения по типу и по значению: *eq, neq*

Среди математических функций следует выделить следующие группы:

– Стандартные: *+, -, *, /, max, min, div (целочисленное деление), abs (абсолютное значение), float (преобразование в тип float), integer (преобразование в тип integer)*

– Расширенные: *sqrt (извлечение корня), round (округление числа), mod (вычисление остатка от деления)*

– Тригонометрические*: sin, sinh, cos, cosh, tan, tanh, acos, acosh, acot, acoth, acsc, acsch, asec, asech, asin, asinh, atan, atanh, cot, coth, csc, csch, sec, sech, deg-grad (преобразование из градусов в секторы), deg-rad (преобразование из градусов в радианы), grad-deg (преобразование из секторов в градусы), rad-deg (преобразование из радиан в градусы)*

– Логарифмические: *log, log10, exp, pi*

Среди функций работы со строками следует назвать функции:

str-cat объединение строк,

sym-cat объединение строк в значение типа symbol,

sub-string выделение подстроки,

str-index поиск подстроки,

eval выполнение строки в качестве команды CLIPS,

build выполнение строки в качестве конструктора CLIPS,

upcase преобразование символов в символы верхнего регистра,

lowcase преобразование символов в символы нижнего регистра,

str-compare сравнение строк,

str-length определение длины строки,

check-syntax проверка синтаксиса строки,

string-to-field возвращение первого поля строки.

Функции работы с составными величинами являются одной из отличительных особенностей языка CLIPS. В их число входят:

create$ создание составной величины,

nth$ получение элемента составной величины,

members поиск элемента составной величины,

subset$ проверка одной величины на подмножество другой,

delete$ *удаление элемента составной величины,*

explode$ *создание составной величины из строки,*

implode$ *создание строки из составной величины,*

subseq$ *извлечение подпоследовательности из составной величины,*

replace$ *замена элемента составной величины,*

insert$ *добавление новых элементов в составную величину,*

first$ *получение первого элемента составной величины,*

rest$ *получение остатка составной величины,*

length$ *определение числа элементов составной величины,*

delete-member$ *удаление элементов составной величины,*

replace-member$ *замена элементов составной величины.*

Функции ввода-вывода используют следующие логические имена устройств:

stdin *устройство ввода,*

stdout *устройство вывода,*

wclips *устройство, используемое как справочное,*

wdialog *устройство для отправки пользователю сообщений,*

wdisplay *устройство для отображения правил, фактов и т.,п.,*

werror *устройство вывода сообщений об ошибках,*

wwarning *устройство для вывода предупреждений,*

wtrase устройство для вывода отладочной информации,

Собственно функции ввода-вывода следующие:

open *открытие файла (виды доступа r, w, r+, a, wb),*

close *закрытие файла,*

printout *вывод информации на заданное устройство,*

read *ввод данных с заданного устройства,*

readline *ввод строки с заданного устройства,*

format форматированный вывод на заданное устройство,

rename *переименование файла,*

remove *удаление файла.*

Среди двух десятков команд CLIPS следует назвать основные команды при работе со средой CLIPS:

load *загрузка конструкторов из текстового файла,*

load+ *загрузка конструкторов из текстового файла без отображения,*

reset *сброс рабочей памяти системы CLIPS,*

clear *очистка рабочей памяти системы,*

run *выполнение загруженных конструкторов,*

save *сохранение созданных конструкторов в текстовый файл,*

exit *выход из CLIPS.*

Более подробно о CLIPS читайте в книге: Частиков А.П., Гаврилова Т.А., Бело Д.Л. Разработка экспертных систем. Среда clips.

Варианты заданий

1. Разработка ПО (ИС)
2. Анализ требований к ПО
3. Проектирование архитектуры ПО
4. Программная реализация ПО
5. Тестирование и отладка ПО
6. Внедрение и поддержка (сопровождение) ПО
7. Проектирование компьютерных интерфейсов
8. CASE-средства
9. Контроль качества ПО
10. Моделирование бизнес-процессов
11. Свободный вариант, студент сам предлагает предметную область.

Лабораторная работа №2

Тема: нечеткие системы

Цель работы: получение практических навыков применения специализированных программ для нечетких систем.

Задание: используя программу SciLab (open source аналог MathLab), построить нечеткую базу знаний по варианту.

Работа заключается в построении:

- лингвистических переменных;
- нечетких продукций;
- поверхностей нечеткого вывода.

Общее количество лингвистических переменных должно быть не меньше 4, правил должно составлять не менее 3.

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание
2. Краткое описание предметной области и решаемой задачи (каков результат работы системы, что является входными данными)
3. Функции принадлежности, нечеткие продукции, поверхности нечеткого вывода.

В качестве методических пособий используйте:

- Андриевский А.Б., Андриевский Б.Р., Капитонов А.А., Фрадков А.Л. Решение инженерных задач в среде Scilab. Учебное пособие.— СПб.: НИУ ИТМО, 2013. — 97 с.
- Нечеткие интеллектуальные системы в среде SciLAB: методические указания к лабораторным работам / сост. Н. Г. Ярушкина, Н. Н. Ястребова, А. В. Чекина. – Ульяновск : УлГТУ, 2009. – 28 с.

Варианты заданий

1. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи закупок (соотношения цены, качества, объема закупок и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
2. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи распределения нагрузок спортсмена (соотношение нагрузок, физического состояния, потребляемых калорий и т.д.), проверить ее на

полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).

3. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи управления транспортным средством (регулировка скорости с учетом передачи, погодных условий, интенсивности потока и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
4. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи управления транспортным средством (управление рулем, газом, тормозом при въезде в гараж), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
5. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи регулирования теплоснабжения (соотношение среднесуточной температуры, ветра, размера здания и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
6. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи регулирования реверсного движения на волжском мосту (учитывать время, интенсивность потока, день недели и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
7. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи подбора специй для блюда (соотношение количества и остроты специй, рецептуры, предпочтений едока, объема пищи и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
8. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи подбора объема блюд (учитывать калорийность, вкусовые предпочтения, количество едоков и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
9. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи подачи электроэнергии в условиях экономии (учет времени суток, типа помещений, количества людей, типа оборудования и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
10. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи подбора интенсивности занятий (учитывать начальный уровень подготовки, объем учебного материала, количество человек в группе, необходимый уровень усвоения и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
11. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи расчета потребления бензина (учитывать тип совершаемых маневров, уровень подготовки водителя, состояние автомобиля, тип

автомобиля и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).

12. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи регулирования системы орошения (учитывать время года, количество выпадающих озадков, вид орошаемой культуры и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
13. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи настройки аудиосистемы (мощность колонок, их количество, размер помещения, назначение установки и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
14. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи выбора дозы снотворного (количество препарата, действие препарата, восприимчивость к выбранному препарату, цель и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
15. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи планирования объема производства продукции (с учетом возможной прибыли, необходимых ресурсов, платежеспособности населения, рынка сбыта и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
16. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи регулирования кондиционера (учитывать его мощность, объем помещения, температуру окружающей среды, необходимую температуру в помещении и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
17. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи распределения нагрузки между компьютерами при использовании их в кластерах (учитывать характеристики компьютеров, их количество, количество параллельного кода, характеристики сети и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
18. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи выбора складского помещения (учитывать площадь склада, количество и размеры продукции, удаленность от места производства и точек реализации, свойства продукции и характеристики помещений и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).
19. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи выбора комплектующих для компьютера (учитывать цену, потребности пользователя, совместимость, сроки использования и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).

20. Построить нечеткую базу знаний (использовать не менее 3 лингвистических переменных) для задачи определения количества линий в службе поддержки (учитывать количество обслуживаемых клиентов, среднюю частоту обращения в службу одного клиента, среднее время обслуживания одной заявки, квалификацию персонала и т.д.), проверить ее на полноту и произвести нечеткий вывод для конкретных значений (выбрать случайным образом).

Лабораторная работа №3

Тема: язык программирования Prolog

Задание 1. Построить базу фактов (не менее 50), используя сервис SWI-Prolog (https://swish.swi-prolog.org/) или аналогичный offline приложения согласно варианту. Сформулировать не менее 5 правил (хотя бы одно с использованием рекурсии). Сформулировать не менее 5 запросов с интерпретацией их на естественном языке.

Варианты:

1) Родословная Романовых.
2) Царства в биологии.
3) Географические объекты (страны, столицы, материки, границы государств).
4) Состав компьютера.
5) Компьютерные сети.
6) Организационная структура университета.
7) Субъекты РФ.
8) Устройство автомобиля.
9) Родственные отношения (сваты, зять, свекры и т.д.)
10) Свободный вариант (студент предлагает тему и согласует ее с преподавателем).

Задание 2. Решить логическую задачу на прологе согласно варианту.

Варианты:

1) Трое, назовем их А, Б и В оказались вместе: один из России, другой из Финляндии, третий из Англии. Один из них увлекается математикой, другой - астрономией, а третий - литературой.

a. А живет не в России, Б - не в Финляндии
b. Тот, кто из России равнодушен к математике, а финн любит астрономию.
c. Для Б не интересна литература.

Чем увлекается В и из какой он страны?

2) По древнему поверью, у каждого месяца есть свой камень-талисман. Так, июню, июлю и сентябрю соответствуют камни рубин, сапфир и жемчуг. Эти камни означают мудрость, здоровье и благополучие. У какого месяца какой камень-талисман и что он означает, если известно, что:

– жемчуг и рубин не соответствуют сентябрю;
– в июне и июле мудрости не наблюдается;
– здоровье не соответствует рубину;
– благополучие не относится к июню.

3) Три друга – Петр, Роман и Сергей учатся на математическом, физическом и химическом факультетах университета.

Если Петр математик, то Сергей не физик.
Если Роман не физик, то Петр – математик.
Если Сергей не математик, то Роман – химик.

Определите специальность Сергея.

4) На новогодний праздник три друга – Евгений, Николай, Алексей, выбрали себе костюмы трех богатырей: Ильи Муромца, Алеши Поповича, Добрыни Никитича. Известно, что:

- Евгений – самый высокий.
- Выбравший костюм Добрыни Никитича меньше ростом, чем выбравший костюм Ильи Муромца.
- Алексею не подошел костюм Добрыни Никитича.

- Ни у одного из друзей имя не совпадает с именем богатырей, выбранных костюмов.

Какой костюм выбрал каждый из друзей?

5) Три друга заняли первое, второе и третье места в соревнованиях универсиады. Друзья — разной национальности, зовут их по-разному и любят они разные виды спорта.

Майкл предпочитает баскетбол и играет лучше чем американец. Израильтянин Саймон играет лучше теннисиста. Игрок в крикет занял первое место.

Кто является австралийцем? Каким видом спорта занимается Ричард?

6) Один из пяти братьев разбил окно.

Андрей сказал: Это или Витя, или Коля .

Витя сказал: Это сделал не я и не Юра .

Дима сказал: Нет, один из них сказал правду, а другой неправду .

Юра сказал: Нет, Дима ты не прав .

Их отец, которому, конечно можно доверять, уверен, что не менее трех братьев сказали правду.

Кто разбил окно?

7) В автомобильных гонках три первых места заняли Алеша, Петя и Коля. Какое место занял каждый из них, если Петя занял не второе и не третье место, а Коля - не третье?

8) Витя, Юра и Миша сидели на скамейке. В каком порядке они сидели, если известно, что Миша сидел слева от Юры, а Витя слева от Миши.

9) Трое ребят вышли гулять с собакой, кошкой и хомячком. Известно, что Петя не любит кошек и живет в одном подъезде с хозяйкой хомячка. Лена дружит с Таней, гуляющей с кошкой. Определить, с каким животным гулял каждый из детей.

10) Свободный вариант (студент предлагает тему и согласует ее с преподавателем).

Отчет по лабораторной должен содержать описание программ, запросов и результатов (отчет можно сдавать в электронном виде).

Информация по языку Пролог: https://coollib.com/b/186558#nav.

Теоретические основы

Наиболее известным языком декларативного (логического) программирования, реализующим модифицированную логику предикатов первого порядка, является PROLOG (англ. PROgramming LOGig - логическое программирование).

В 1965 году в работе «A machine oriented logic based on the resolution principle»[3], опубликованной в 12 номере журнала «Journal of the ACM», Дж Робинсон представил метод автоматического поиска доказательства теорем в исчислении предикатов первого порядка, получивший название **«принцип резолюции»**. На самом деле, идея данного метода была предложена Эрбраном в 1931 году, когда еще не было компьютеров (Herbrand, «Une methode de demonstration», These, Paris, 1931). Робинсон модифицировал этот метод так, что он стал пригоден для автоматического (компьютерного) использования и разработал эффективный алгоритм унификации, составляющий базис его метода.

Идеи использования логики в качестве языка программирования зародилась в начале 1970-х годов. Первыми исследователями, которые занялись разработкой этой идеи, были Роберт Ковальски (Robert Kowalski) из Эдинбурга (теоретические основы, статьи 1971 и 1974 г.), Маартен ван Эмден (Maarten van Emden) из Эдинбурга (экспериментальная демонстрационная система) и Ален Колмероэ (Alain Colmerauer) из Марселя (реализация, 1973 г.). В 1973 году «группа искусственного интеллекта» во главе с Аленом Колмероэ создала в Марсельском университете программу, предназначенную для доказательства теорем. Эта программа использовалась при построении систем обработки текстов на

естественном языке. Программа доказательства теорем получила название Prolog (фр. PROgrammation en LOGique) и послужила прообразом Пролога. Ходят легенды, что автором этого названия была жена Алена Колмероэ. Программа была написана на Фортране и работала довольно медленно.

Популяризации Пролога во многом способствовали:

- эффективная реализация (интерпретатор/компилятор) этого языка для ЭВМ DEC-10 Дэвидом Д. Г. Уорреном (David D.H. Warren) из Эдинбурга в 1977 г. Послужила прототипом для многих последующих реализаций Пролога. Что интересно, компилятор был написан на самом Прологе. Эта реализация Пролога, известная как «эдинбургская версия», фактически стала первым и единственным стандартом языка;
- разработка Кларком и Маккейбом (Великобритания) в 1980 году версии для персональных ЭВМ;
- японский проект создания компьютеров V поколения. В конце 1978 г. Министерство внешней торговли и промышленности (МВТП) Японии поручило разработать проект интеллектуальных ЭВМ, специально созданному для этих целей Токийскому институту вычислительной техники (ICOT)[4]. Сердцем этих компьютеров должен был стать не арифметический процессор, а специально оптимизированный для работы с прологоподобными программами.

В 1995 году был опубликован официальный стандарт ISO[5]/IEC[6] языка Пролог (ISO/IEC 13211-1 «Information technology - Programming languages - Prolog - Part 1: General core» - «Информационные технологии. Языки программирования. Пролог. Часть 1. Общее ядро»).

На сегодня существует довольно много реализаций Пролога. Наиболее известные из них следующие: BinProlog, AMZI-Prolog, Arity Prolog, CProlog, Micro Prolog, МПролог, Prolog-2, Quintus Prolog, SICTUS Prolog, Silogic iis Workbench, Strawberry Prolog, SWI-Prolog, Turbo Prolog (PDC Prolog, Visual Prolog), UNSW Prolog и т. д. В нашей стране были разработаны такие версии Пролога как Пролог-Д (С. Григорьев), Акторный Пролог (А. Морозов), а также Флэнг (А. Манцивода, В. Петухин).

SWI-Prolog является свободно распространяемой (англ. Free Software[1]) реализацией (диалектом) языка программирования Пролог, которая практически полностью соответствует стандарту ISO[2]/IEC[3] 13211-1 «Information technology - Programming languages - Prolog - Part 1: General core» (рус. «Информационные технологии. Языки программирования. Пролог. Часть 1. Общее ядро»).

SWI-Prolog развивается с 1987 года. Его создателем и основным разработчиком является Ян Вьелемакер (Jan Wielemaker). Название SWI происходит от Sociaal-Wetenschappelijke Informatica (гол. социально-научная информатика), первоначального названия группы в Амстердамском университете, где работает Вьелемакер.

SWI-Prolog позволяет разрабатывать приложения любой направленности, включая Web-приложения и параллельные вычисления, но основным направлением использования является разработка экспертных систем, программ обработки естественного языка, обучающих программ, интеллектуальных игр и т.п.

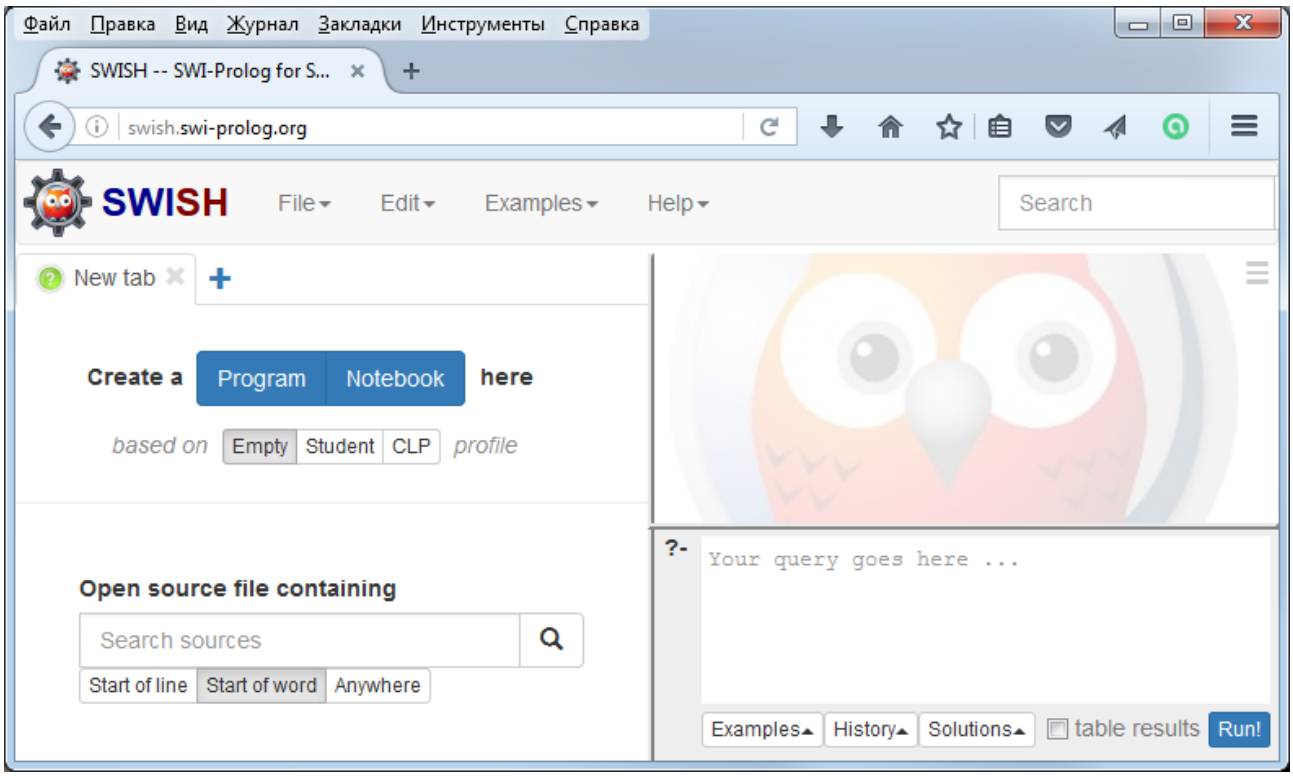

Рис.1. Стандартная online-среда программирования SWI-Prolog

Для перехода в режим редактирования и исполнения программ необходимо нажать на кнопку «Program» (см. рис.1).

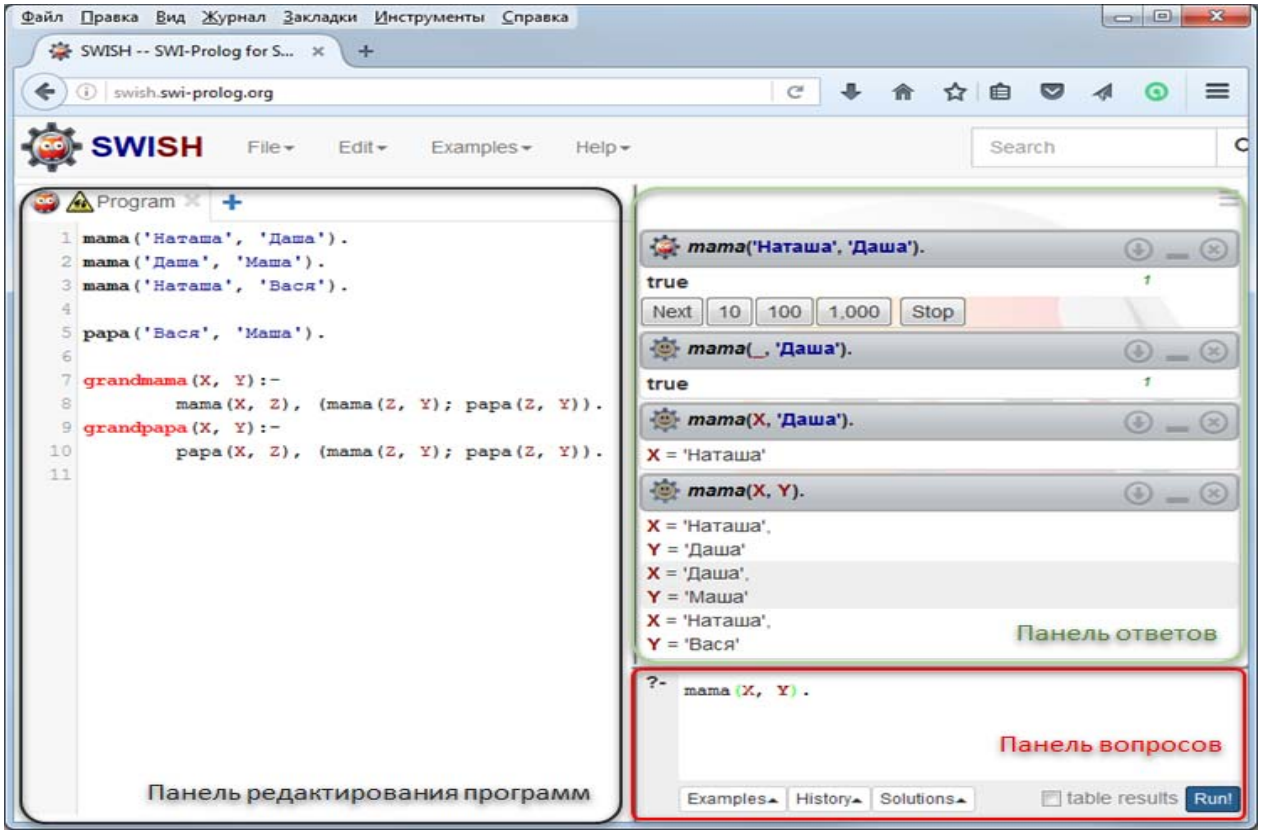

Рис.2. Режим редактирования и исполнения программ

В левой панели осуществляется редактирование программы, содержащей факты и правила.

В правой нижней панели выполняется набор вопросов и запуск их на исполнение с помощью кнопки «Run!».

В правой верхней панели интерпретатор SWI-Prolog выдает ответы на вопросы. В случае если на вопрос может быть получено более одного ответа, с помощью кнопок «Next», «10», «100» и «1,000» можно вывести на панель дополнительные ответы.

Некоторые операции и предикаты SWI-Prolog

Операция / Предикат	Назначение
true	Истина
fail, false	Ложь
=	Для переменной, стоящей слева от операции: - свободной - присваивание без преобразования (вычисления) выражения справа от операции; - связанной - сравнение без преобразования (вычисления) выражения справа от операции.
<, =<, >=, >	Арифметические (только для чисел) операции сравнения
=:=	Арифметическое равенство
=\=	Арифметическое неравенство
is	Для переменной, стоящей слева от операции: - свободной - присваивание с преобразованием (вычислением) выражения справа от операции; - связанной - сравнение с преобразованием (вычислением) выражения справа от операции.
@<, @=<, @>=, @>	Операции сравнения для констант и переменных любого типа (чисел, строк, списков и т.д.)
==	Равенство констант и переменных любого типа
\==	Неравенство констант и переменных любого типа
not(A)	Отрицание логического выражения A
read(A)	Чтение значения с клавиатуры и присваивание его переменной A
write(A)	Печать A на экран с установкой курсора после последнего напечатанного символа
writeln(A)	Печать A на экран с переводом курсора в начало следующей строки
nl	Перевод курсора в начало следующей строки
repeat	Предикат, выдающий новое истинное значение при возврате. Передоказываемый предикат
!	Предикат (cut, сократить), запрещающий возврат далее той точки, где он стоит
assert(A), assertz(A)	Динамическое добавление факта (правила) в конец списка подобных фактов (правил) базы знаний (программы)

asserta(A)	Динамическое добавление факта (правила) в начало списка подобных фактов (правил) базы знаний
retract(A)	Удаление первого факта (правила) базы знаний
retractall(A)	Удаление всех фактов (правил) базы знаний с именем A

Пример решения логическойзадачи

Задача:
Как то раз случай свёл в купе астронома, поэта , прозаика и драматурга. Это были Алексеев, Борисов, Константинов и Дмитриев. Оказалось, что каждый из них взял с собой книгу написанную одним из пассажиров этого купе. Алексеев и Борисов углубились в чтение предварительно обменявшись книгами. Поэт читал пьесу, прозаик — очень молодой человек, выпустивший свою книгу, говорил что он никогда и ни чего не читал по астрономии. Борисов купил одно из произведений Дмитриева. Никто из пассажиров не читал свои книги. Что читал каждый из них, кто кем был?

Модификация кода, описанного по ссылке (см. https://pro-prof.com/archives/1299)

```
man(alekseev).
man(borisov).
man(konstantinov).
man(dmitriev).

writebook(astronomy).
writebook(poetry).
writebook(prose).
writebook(piece).

passenger(_Name, _Read, _Buy, _Write).

% ДОБАВЛЕНО!!!
unique([]).
unique([X|Xs]) :-
  maplist(dif(X),Xs),
  unique(Xs).
% ------!!!

check([]):-!.
check([passenger(_, XRead, XBuy, XWrite)|T]):-
 !, not(XRead = XWrite), not(XBuy = XWrite), check(T).

solve(Solve):-
 Solve = [passenger(X, XRead, XBuy, XWrite), passenger(Y, YRead, YBuy, YWrite),
       passenger(Z, ZRead, ZBuy, ZWrite), passenger(W, WRead, WBuy, WWrite)],

 % 4 разных пасажира
 man(X), man(Y), man(Z), man(W), unique([X, Y, Z, W]),

 % каждый написал книгу
 writebook(XWrite), writebook(YWrite),
 writebook(ZWrite), writebook(WWrite),
 unique([XWrite, YWrite, ZWrite, WWrite]),
```

```
% каждый купил книгу
writebook(XBuy), writebook(YBuy),
writebook(ZBuy), writebook(WBuy),
unique([XBuy, YBuy, ZBuy, WBuy]),

% каждый читает книгу
writebook(XRead), writebook(YRead),
writebook(ZRead), writebook(WRead),
unique([XRead, YRead, ZRead, WRead]),
  % поэт читает пьесу
member(passenger(_, piece, _, poetry), Solve),

% прозаик читает не астрономию
not(member(passenger(_, astronomy, _, prose), Solve)),
% прозаик не покупал астрономию
not(member(passenger(_, _, astronomy, prose), Solve)),

% никто не читает и не покупал свою книгу
check(Solve),

% алексеев и борисов обменялись книгами
member(passenger(alekseev, AlekseevRead, AlekseevBuy, _), Solve),
member(passenger(borisov, AlekseevBuy, AlekseevRead, _), Solve),

% Борисов купил произведение Дмитриева
member(passenger(dmitriev, _, _, DmitrievWrite), Solve),
member(passenger(borisov, DmitrievWrite, _, _), Solve).
```