МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
О.Г. Локтионова
«__» _____ 2020 г.

РАЗРАБОТКА ЭКСПЕРТНОЙ СИСТЕМЫ НА ЯЗЫКЕ CLIPS

Методические указания по выполнению лабораторной работы по дисциплине «Экспертные системы» для студентов направления подготовки 09.04.04 «Программная инженерия»

Курск 2020

УДК 004.65

Составители: В.Г. Белов, Т.М. Белова

Рецензент

Кандидат технических наук, доцент кафедры программной инженерии ЮЗГУ И.Н. Ефремова

Разработка экспертной системы на языке CLIPS: методические указания по выполнению лабораторной работы по дисциплине "Экспертные системы" для студентов направления подготовки 09.04.04 "Программная инженерия" / Юго-Зап. гос. ун-т; сост.: В.Г. Белов, Т.М. Белова, – Курск, 2017. – 19 с.

Изложены основные процедуры построения экспертной системой на языке CLIPS.

Материал предназначен для студентов направления подготовки 09.04.04 «Программная инженерия» (профиль "Разработка информационно-вычислительных систем").

Подписано в печать 15.12.17/ Формат 60x84 1/16.
Усл. печ. л. 1,0. Уч.-изд. л. 0,9 Тираж 100 экз. Заказ 4430. Бесплатно.
Юго-Западный государственный университет

305040, Курск, ул.50 лет Октября, 94.

Содержание

1 Цель работы

Цель работы: ознакомиться с особенностями языка CLIPS, получить практические навыки разработки экспертных систем, основанных на использовании продукционной модели представления знаний и сведений, предоставленных в [1-4].

2 Общие сведения системе CLIPS

CLIPS располагает тремя механизмами представления знаний: процедурным, эвристическим и объектно-ориентированным. Рассмотрим первые два механизма.

Процедурный механизм позволяет пользователю при помощи встроенных в язык функций разрабатывать или конструировать новые функции, выполняющие некоторые действия или возвращающие какие-либо значения. В этом смысле CLIPS напоминает такие известные языки программирования, как C, C++ или Pascal. Так, для создания пользовательских функций используется конструктор deffunction, имеющий следующий синтаксис:

```
(deffunction имя_функции
[необязательный комментарий]
(список формальных параметров)
(действие_1)
(действие_2)
 ......
(действие_N))
```

Например, определим функцию om(x,y), которая возвращает целую часть частного от деления переменной y на переменную x:

```
(deffunction om (
?x ?y)
(div ?y ?x))
```

Обратите внимание на то, что в CLIPS имя переменной начинается с символа “ ? “, и что для вызова функции (в данном случае – встроенной функции деления нацело div) используется префиксная нотация, а также на

то, что вся конструкция представляет собой список, состоящий из четырех полей. Этим CLIPS похож не только на C, но и на LISP.

Эвристический механизм представления знаний в CLIPS реализуется при помощи правил в форме

```
ЕСЛИ условие_1 и ... и условие_N выполняются,
ТО
ВЫПОЛНИТЬ действие_1 и ... и действие_N.
```

Список условий называется левой частью правила (Left-Hand Side или LHS). Список действий называется правой частью правила (Right-Hand Side или RHS). Возможность применить конкретное правило определяется тем, выполняются ли условия, которые сформулированы в его левой части. Выполнение или невыполнение условий определяется в момент их сопоставления с так называемыми фактами, которые образуют ни что иное, как базу данных. В CLIPS такая база данных может представлять некоторую предметную область, исходное или текущее состояние какой-либо проблемы, может моделировать в пространстве или во времени поведение какой-либо системы или любой сущности, которую можно описать посредством множества записей в виде списков.

Существует несколько способов создания базы данных, один из них – использование конструктора deffacts. Его синтаксис таков:

```
(deffacts имя_базы_данных
[необязательный комментарий]
(факт_1)
(факт_2)
 .....
(факт_N))
```

Каждый факт в базе данных представляет собой запись в виде списка. Список может содержать одно или несколько полей, принимающих символьные либо числовые значения. Список также может быть пустым.

Если каждое условие в левой части правила находит себя среди фактов – происходит активизация правила и выполняются ВСЕ действия, записанные в его правой части. В противном случае правило не активизируется.

Работа правила очень напоминает условный оператор if-then, присутствующий во многих процедурных языках программирования. Принципиальная разница заключается в том, что оператор if-then выполнится в любом случае, когда до него дойдет очередь в программе. Что касается правила, то интерпретатор CLIPS еще “подумает”, выполнять его или нет. Так, при старте программы, содержащей множество фактов и правил, интерпретатор CLIPS запускает машину логического вывода, которая выясняет, какие из правил можно активизировать. Это выполняется циклически, причем каждый цикл состоит из трех шагов:

- сопоставление фактов и правил;
- выбор правила, подлежащего активизации;
- выполнение действий, предписанных правилом.

Таким образом, правила, взаимодействующие с базой данных в виде фактов, вносят в нее функциональность и образуют вместе с ней базу знаний. Для создания правила используется конструктор defrule, который имеет следующий синтаксис:

```
(defrule имя_правила
[необязательный комментарий]
[необязательное объявление]
(условие_1)
(условие_2)
 ......
(условие_M)
=>
(действие_1)
(действие_2)
 ......
(действие_N))
```

Обратите внимание: левая часть правила отделяется от правой комбинацией символов “=>”, а количество условий и действий в правиле в общем случае не совпадает. Для пояснения вышесказанного рассмотрим несколько примеров.

2. 1 Пример 1

Рассмотрим предметную область, которая представляет участников некоторой конференции, приехавших из разных городов. На подобных мероприятиях все участники обычно проходят регистрацию. Пусть эта процедура представляет собой ввод сведений об участниках в базу данных, в которой на каждого участника выделяется одна запись (факт), состоящая из списка с тремя полями. Пусть первое поле имеет символьное значение rep – сокращение от representative (представитель). В общем случае это значение может быть любым, а поле может отсутствовать. Во втором поле списка хранится фамилия участника, а в третьем – город, из которого участник прибыл. Содержимое фактов базы данных с именем rep может быть, например, таким:

```
(deffacts rep
(rep Alejnov Odessa)
(rep Ladak  Odessa)
(rep Slobodjanjuk Lvov)
(rep Klitka Lvov)
(rep Bojko Kiev)
(rep Pustovit Odessa)
(rep Spokojnij Odessa)
(rep Shamis Odessa)
(rep Lobovko Kiev)
(rep Zadorozhna Lvov)
(rep Javorskij Lvov))
```

Используя любой текстовый редактор, создадим и сохраним базу данных в виде текстового ASCII-файла с именем, повторяющим имя базы данных (то есть rep). Это позволяет легко редактировать данные, независимо от каких-либо других программных модулей, добавляя новых участников или удаляя выбывших.

После окончания конференции организаторы подводят итоги, определяя массу показателей. В частности, пусть требуется определить количество представителей от каждого города. Алгоритм решения такой задачи прост. Для каждого города задаем счетчик и последовательно просматриваем списки в записях файла rep. Если в записи третье поле списка

имеет значение Kiev, то содержимое соответствующего счетчика увеличиваем на единицу. Для других городов – аналогично. Программа на языке CLIPS, реализующая указанный алгоритм, может быть, например, такой:

```
(defglobal ?*odessa* = 0)
(defglobal ?*kiev* = 0)
(defglobal ?*lvov* = 0)

(defrule start
(initial-fact)
=>
(printout t crlf «REPRESENTATIVES» crlf)

(defrule odessa
(rep ? Odessa)
=>
(bind ?*odessa* (+ ?*odessa* 1)))

(defrule kiev
(rep ? Kiev)
=>
(bind ?*kiev* (+ ?*kiev* 1)))

(defrule lvov
(rep ? Lvov)
=>
(bind ?*lvov* (+ ?*lvov* 1)))

(defrule result
(declare (salience -1))
(initial-fact)
=>
(printout t «from Odessa: « ?*odessa* crlf)
(printout t «from Kiev: « ?*kiev* crlf)
(printout t «from Lvov: « ?*lvov* crlf))
```

В первых трех строках программы при помощи конструктора defglobal объявляются три глобальные переменные: ?*odessa*, ?*kiev* и ?*lvov*. Эти переменные являются счетчиками. В CLIPS переменная может быть и локальной – но тогда она связывается только с тем правилом, в котором объявляется.

Далее следует правило с именем start, левая часть которого представляет собой запись (initial-fact). Так обозначается системный начальный факт, который создается в рабочей памяти интерпретатора CLIPS

по команде (reset) до запуска программы на выполнение. Для чего он нужен? Дело в том, что в CLIPS-программах распространенными правилами являются такие, которые добавляют факты в базу данных, либо, наоборот, удаляют их. Типичной является ситуация, когда при старте программы в базе данных нет фактов, удовлетворяющих хотя бы одному правилу. В этом случае программа ничего не выполнит. Для того чтобы начать вычисления и используется системный начальный факт, который, независимо от фактов в базе данных, активизирует некоторое правило, добавляющее такие факты, которые, в свою очередь, активизируют правила, условия которых не выполнялись в начальный момент.

В данной программе (initial-fact) запускает правило start, которое активизируется независимо от фактов в файле rep и присутствует в программе только с одной целью – вывести заголовок. Для этого в его правой части вызывается встроенная функция printout с ключом t, выводящая на стандартное устройство вывода (монитор) заголовок, заключенный в кавычки. Комбинация символов crlf является аналогом endl в C++ и служит для перевода курсора на следующую строку.

Следующие три правила с именами odessa, kiev и lvov можно назвать ядром программы. В них производится подсчет количества участников – соответственно, из Одессы, Киева и Львова.

Рассмотрим правило lvov. Оно активизируется в том случае, когда в базе данных находится факт (rep ? Lvov). Не трудно догадаться, что символ “ ? “ во втором поле этого списка означает символ универсальной подстановки и заменяет собой любую фамилию. Отсюда следует, что правило lvov активизируется столько раз, сколько раз факт (rep ? Lvov) присутствует в базе данных. При этом столько же раз выполнятся действия, содержащиеся в правой части правила. Встроенная функция bind – аналог оператора присваивания. Следовательно, содержимое переменной ?*lvov* увеличивается на единицу и результат сохраняется в этой же переменной. Аналогично работают правила odessa и kiev.

Действия, которые выполняются в последнем правиле программы, отражены в его названии. Правая часть правила особых комментариев не требует, в то время как левая часть заслуживает подробного рассмотрения. В CLIPS существует несколько стратегий очередности выполнения правил, а сами правила могут иметь приоритет, который задается встроенной функцией declare с параметром salience (особенность). Этот параметр может принимать целочисленные значения от –10000 до +10000. По умолчанию для всех правил величина salience равна нулю. Если в правиле result не указать приоритет, оно будет конфликтовать с правилом start за очередность выполнения, так как у этих правил одинаковая левая часть. Для устранения конфликта в правиле result приоритет указан явно и со знаком минус, в связи с чем это правило выполнится последним.

Используя любой текстовый редактор, наберем и сохраним текст программы в ASCII-файле со стандартным для CLIPS-программ расширением .clp и с именем represent. Командой clips вызовем интерпретатор CLIPS, командой (load имя_файла) загрузим в интерпретатор файлы rep и represent.clp, командами (reset) и (run) запустим программу represent.clp на выполнение.

```
CLIPS (V6.21 06/15/03)
CLIPS> (load rep)
 .........
TRUE
CLIPS> (load represent.clp)
 .........
TRUE
CLIPS> (reset)
CLIPS> (run)

REPRESENTATIVES
from Odessa: 5
from Kiev: 2
from Lvov: 4

CLIPS>
```

Сообщение интерпретатора TRUE означает, что в файле нет синтаксических ошибок и команда загрузки выполнена корректно.

Многоточием представлены другие сообщения интерпретатора, которые в данном случае опущены.

Как следует из описанных действий, в интерпретаторе CLIPS находятся два файла. Первый, с именем rep, является базой данных. Второй, с именем represent.clp, содержит сведения (правила) о том, как эти данные могут быть использованы. Таким образом, вместе файлы образуют базу знаний, которая содержит, по крайней мере, два знания. Первое – общий состав участников конференции. Его можно посмотреть, не выходя из интерпретатора по команде (facts). Второе знание – количество участников от каждого города.

В рассмотренном примере база знаний состоит из двух программных модулей. Однако ничто не мешает использовать одну программу, сохраненную в одном файле. В следующем примере показано, как это делается. В нем же эвристический механизм представления знаний используется вместе с процедурным.

2.2 Пример 2

Пусть требуется подобрать резистор для участка цепи схемы электрической принципиальной некоторого радиоэлектронного устройства. Резистор характеризуется сопротивлением, которое определяется по измеренным или рассчитанным значениям электрического тока, проходящего через резистор, и падению напряжения на нем. Программа с именем resistor.clp, решающая эту задачу, может быть, например, такой

```
(deffacts resistors; база данных резисторов
(resistor Ra 2)
(resistor Rb 5)
(resistor Rc 7))

(deffunction om; функция om(x,y)
( ?x ?y)
(div ?y ?x))

(defrule input; начальное правило
(initial-fact)
=>
(printout t crlf "Input current value: ")
(bind ?i (read))
```

```
(printout t "Input strait value: ")
(bind ?u (read))
(assert (numbers ?i ?u)))


(defrule take; подобрать резистор из БД
(numbers ?i ?u)
(resistor ?r =(om ?i ?u))
=>
(printout t crlf "You must take resistor « ?r»." crlf crlf)
(reset)
(halt))

(defrule nothing; если в БД нет подходящего резистора
(numbers ?i ?u)
(resistor ?r ~=(om ?i ?u))
=>
(printout t crlf "There is nothing for You in my database!" crlf
crlf)
(reset)
(halt))
```

Программа состоит из нескольких частей: базы данных с именем resistors, объявления пользовательской функции om и трех правил с именами input, take и nothing.

В базе данных содержатся сведения о резисторах. Они представлены в виде списков, состоящих из трех полей. Первое поле имеет значение resistor, которое отражает тип радиодетали. Во втором поле списка содержится тип резистора. Последнее поле хранит значение сопротивления.

О функции om подробно говорилось ранее. В данном случае она используется для представления процедурного знания – закона Ома. Правило input предназначено для ввода исходных данных. Оно активизируется системным начальным фактом и требует от пользователя ввести ток и напряжение. Встроенная функция read возвращает значение, введенное со стандартного устройства ввода (клавиатуры), которое сохраняется в переменных ?i и ?u.

В правой части правила выполняется еще одно действие. Команда assert добавляет в рабочую память интерпретатора CLIPS факт (numbers ?i ?u) для того, чтобы можно было обращаться к локальным переменным ?i и ?u, связанным с правилом input, из других правил программы.

В следующих двух правилах пользователю либо предлагается тип подходящего резистора (правило take), либо сообщается об отсутствии такового (правило nothing).

Рассмотрим правило take. Его левая часть состоит из двух условий, поэтому правило активизируется, если оба условия будут выполнены. Первое условие выполняется, так как соответствующий факт уже создан правилом input. Второе условие выполнится, если будет точно соответствовать какому-либо факту (списку) в базе данных. Первое поле условия вопросов не вызывает. Во втором поле условия находится переменная ?r, которая может принять значение Ra, либо Rb, либо Rc – в зависимости от содержимого третьего поля условия. В этом поле осуществляется вызов функции om и сохраняется возвращаемое функцией значение. Так, если возвращаемое значение будет равно 7, то условие выполнится, переменная ?r примет значение Rc, правило активизируется и выведет на экран монитора предложение выбрать резистор Rc. Если возвращаемое функцией om значение равно 5, то пользователю будет предложен резистор Rb и т.д.

В левой части правила nothing вроде бы полная аналогия – за исключением одной маленькой модификации. В третьем поле второго условия перед вызовом функции om стоит символ “ ~ “, означающий логическое отрицание. Таким образом, условие выполнится и правило активизируется, если возвращаемое функцией om значение будет не 2, не 5 и не 7.

Находясь в интерпретаторе CLIPS, командой (clear) очистим его от данных предыдущего примера, загрузим файл resistor.clp и запустим программу на выполнение:

```
CLIPS> (clear)
CLIPS> (load resistor.clp)
 ............
TRUE
CLIPS> (reset)
CLIPS> (run)

Input current value: 3
Input strait value: 15
```

```
You must take

resistor Rb. CLIPS>

(run)

Input current
value: 3.5 Input
strait value: 5.44

There is nothing for You in my

database! CLIPS>
```

Таким образом, файл resistor.clp также представляет собой базу знаний,поскольку содержит и базу данных, и сведения (правила) о том, как данные могут быть использованы. Эта база располагает, по крайней мере, тремя знаниями. Первое – общий список резисторов с указанием типа и сопротивления. Второе – закон Ома. Третье знание – предлагаемый тип резистора.

Интеллект базы знаний можно существенно повысить добавлением новых данных и правил. Так, вместо закона Ома можно использовать более серьезные методики определения сопротивления резистора, например схемотехническую САПР PSpice. Результаты ее работы можно сохранить в текстовом файле, а затем вызвать из третьего поля второго условия правила take.

Другой путь – добавление новых типов резисторов в базу данных. Например, интересный результат получается при внесении в базу данных записи (resistor Rd 2). При некоторой доработке правил take и nothing, если возвращаемое функцией om значение равно 2, правило take отработает два раза и предложит резисторы Ra и Rd. Затем можно пойти дальше: добавить правило (правила), которое выберет из резисторов Ra и Rd предпочтительный для некоторых конкретных условий и т.д.

3 Порядок выполнения работы

1. Изучить общие сведения.
2. Запустить *«Пример 2»* и разобраться, как он работает.Разработать на основе примера (файл *auto.clp*) экспертную систему для требуемой предметной области согласно заданию.
3. Получить распечатки примеров работы программы.
4. Оформить отчет.

4 Содержание отчета

1. Вариант задания, включающий описание предметной области.
2. Исходный текст разработанной программы.
3. Графическое представление алгоритма работы программы.
4. Распечатки экрана с результатами работы программы.
5. Выводы по работе.

5 Контрольные вопросы

1. Основные компоненты экспертных систем.
2. Какая форма записи используется в CLIPS для выражений?
3. Как организована база знаний в CLIPS?
4. Какой механизм используется для вывода новых знаний?

6 Список использованных источников

1. Попов Э. В. Экспертные системы: Решение неформализованных задач в диалоге с ЭВМ. – М.: Наука, 1987.
2. Уотермен Д. Руководство по экспертным системам. Пер. с англ. – М.: Мир, 1989.
3. Частиков А. П., Гаврилова Т. А., Белов Д. Л. Разработка экспертных систем. Среда CLIPS. – СПБ: БХВ-Петербург, 2003.
4. Системы искусственного интеллекта [Электронный ресурс]: Методические указания к выполнению лабораторных работ / Сост. Гудков П.А. - Пенза: Пензенский гос. ун-т, 2007. - 53 с. – Режим доступа: http://window.edu.ru/resource/709/59709