МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
НЕВИННОМЫССКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по дисциплине

«ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В СИСТЕМАХ УПРАВЛЕНИЯ»

Методические указания к выполнению лабораторных работ
для студентов направления подготовки
15.03.04 «Автоматизация технологических процессов и производств»

Невинномысск 2024

Методические указания предназначены для студентов направления подготовки 15.03.04 «Автоматизация технологических процессов и производств». Они содержат основы теории, порядок проведения лабораторных работ и обработки экспериментальных данных, перечень контрольных вопросов для самоподготовки и список рекомендуемой литературы. Работы подобраны и расположены в соответствии с методикой изучения дисциплины «Представление знаний в системах управления». Объем и последовательность выполнения работ определяются преподавателем в зависимости от количества часов, предусмотренных учебным планом дисциплины, как для очной, так и для заочной форм обучения.

Методические указания разработаны в соответствии с требованиями федерального государственного образовательного стандарта в части содержания и уровня подготовки выпускников по направлению подготовки 15.03.04 «Автоматизация технологических процессов и производств»

Составитель: канд. техн. наук А.А. Евдокимов

Ответственный редактор: канд. техн. наук Д.В. Болдырев

Содержание

Лабораторная работа № 1

Аппроксимация функции правилами нечеткого вывода

Цель работы: Изучение основных понятий теории нечетких множеств и их приложений. Знакомство с пакетом математических приложений MATLAB.

Краткие теоретические сведения

Значительный шаг в направлении развития теории нечетких множеств сделал профессор Калифорнийского университета Лотфи А. Заде (1965 г. – публикация работы “Fuzzy Sets”). Заде расширил понятие множества, допустил, что характеристическая функция (функция принадлежности элемента множеству) может принимать любые значения в интервале [0, 1]. Такие множества были названы им нечеткими (fuzzy).

Пусть E – универсальное множество, x – элемент E, P – некоторое свойство. Обычное (четкое) множество A универсального множества E, элементы которого удовлетворяют свойству P, определяются как множество упорядоченных пар

$$A = \{ \mu_A(x) / x \},$$

где $\mu_A(x)$ – характеристическая функция, принимающая значение 1, если *x* удовлетворяет свойству *P*, и 0 – в противном случае.

В теории нечетких множеств для элементов *x* из *E* нет однозначного ответа «да/нет» относительно свойства *P*. В связи с этим нечеткое множество *A* универсального множества *E* определяется как множество упорядоченных пар с функцией принадлежности $\mu_A(x)$, принимающей значение в некотором упорядоченном множестве *M* (например, *M*=[0, 1]).

Функция принадлежности указывает степень (или уровень) принадлежности элемента x подмножеству *A*. Множество *M* называют

множеством принадлежностей. Если M={0, 1},то нечеткое подмножество A может рассматриваться как обычное или четкое множество.

Примеры записи нечеткого множества.

1. Пусть E ={x_1, x_2, x_3, x_4, x_5}; A – нечеткое множество, для которого $\mu_A(x_1)$=0,3; $\mu_A(x_2)$=0; $\mu_A(x_3)$=1; $\mu_A(x_4)$=0,6; $\mu_A(x_5)$=0,9.

Тогда A можно представить в виде

A = {0,3/x_1; 0/x_2; 1/x_3; 0,6/x_4; 0,9/x_5}.

2. Пусть E = {1, 2, 3, … 100} и соответствует понятию «возраст», тогда нечеткое множество «молодой» может быть определено с помощью выражения

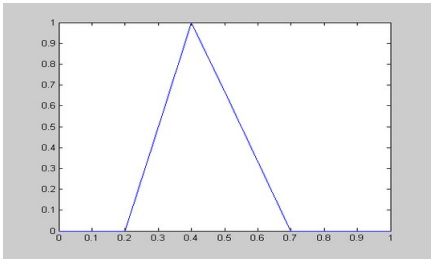
$$\mu_{\text{"молодой"}} = \begin{cases} 1, & x \in [1,25], \\ \dfrac{1}{1+\left(\dfrac{x-25}{5}\right)^2}, & x > 25. \end{cases}$$

В приведенных выше примерах использованы прямые методы определения функций принадлежности, когда эксперт задает значение $\mu_A(x)$ для каждого $x \subset E$. Такой способ используется для измеряемых понятий (скорость, температура и т.д.) или когда выделяют полярные значения. Разновидностью прямого способа задания функции принадлежности является групповой метод, когда группе экспертов предлагают сделать оценку того или иного явления, например, оценить: «этот человек лысый» или нет, – тогда количество утвердительных ответов, деленное на общее число экспертов, дает значение $\mu_{лысый}$ для данного лица.

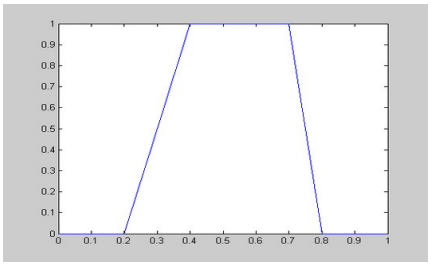
Кроме указанных способов задания функций принадлежности используют также типовые формы функций принадлежности (рисунок 1).

Аналитическая запись некоторых типовых функций принадлежности:

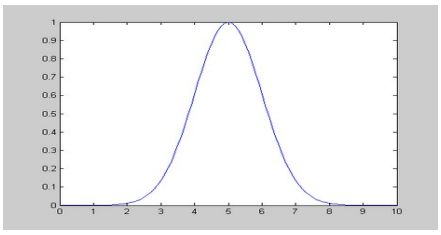
треугольная – $\mu(x) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right)$, a, b, c – определяют параметры функции; гауссова – $\mu(x) = e^{-\left(\frac{x-a}{b}\right)^2}$, a, b – параметры функции принадлежности; сигмоидная – $\mu(x) = \frac{1}{1+\exp(-a(x-b))}$, a, b – параметры функции принадлежности.



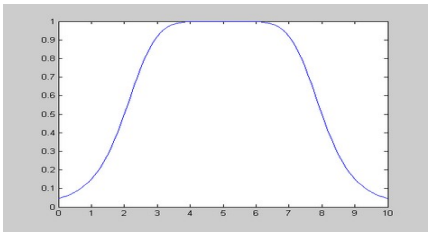
а) треугольная (trimf)



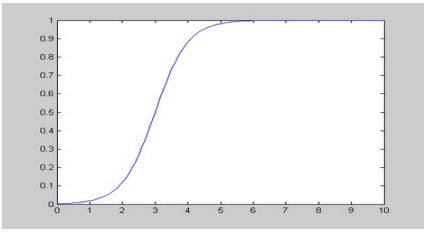
б) трапецеидальные (trapmf)



в) гауссова (gaussmf)



г) обобщенная колоколообразная (gbellmf)



д) сигмоидная (sigmf)

Рисунок 1 – Примеры типовых форм функций принадлежности (ниже в скобке приводится обозначение функции в среде MATLAB)

Основные операции над нечеткими множествами

Включение

Пусть А и В нечеткие множества на универсальном множестве Е. А содержится в В, если $\forall x \in E \ \mu_A(x) \le \mu_B(x)$. Обозначается $A \subset B$.

Равенство

A и B равны, если $\forall x \in E \ \mu_A(x) = \mu_B(x)$. Обозначение: $A=B$.

Дополнение

Пусть M=[0,1], А и В дополняют друг друга, если $\forall x \in E \ \mu_A(x) = 1 - \mu_B(x)$. Обозначается $B = \overline{A}$ или $A = \overline{B}$.

Пересечение

$A \cap B$ – наименьшее нечеткое подмножество, содержащееся одновременно в А и В: $\mu_{A \cap B}(x) = \min\left(\mu_A(x), \mu_B(x)\right) = \mu_A(x) \tilde{\wedge} \mu_B(x)$.

Или, по-другому, определение в классе треугольных норм (t – норма).

Типичными t – нормами являются [2, 3]:

- операция min как нечеткое логическое произведение с нечеткими переменными X1, X2: $X_1 \tilde{\wedge} X_2 = \min(X_1, X_2)$;

- алгебраическое произведение $X_1 \tilde{\wedge} X_2 = X_1 \cdot X_2$.

Объединение

$A \cup B$ – наибольшее нечеткое подмножество, включающее как A, так и B, с функцией принадлежности:

$$\mu_{A \cup B}(x) = \max\left(\mu_A(x), \mu_B(x)\right) = \mu_A(x) \tilde{\vee} \mu_B(x).$$

Или, по другому, определение в классе треугольных конорм (s – норма).

Типичными s – нормами являются:

- операция нечеткого логического сложения $X_1 \tilde{\vee} X_2 = \max(X_1, X_2)$;

- алгебраическая сумма $X_1 \tilde{\vee} X_2 = X_1 + X_2 - X_1 X_2$,

где «~» означает нечеткую логическую операцию.

Разность

A-B= $A \cap \overline{B}$ с функцией принадлежности: $\mu_{A-B}(x) = \min(\mu_A(x), 1 - \mu_B(x))$.

Нечеткие отношения

Нечеткое n-мерное отношение определяется как нечеткое подмножество R на E, принимающее свои значения в M. В случае n=2 и M=[0,1] нечетким отношением R между множествами X=E_1 и Y=E_2 будет называться функция R: (X, Y) → [0,1], которая ставит в соответствие каждой паре элементов (x, y) ∈ X x Y величину $\mu_R(x, y) \in [0,1]$.

Обозначение: $x \in X,\ y \in Y : xRy$.

Алгебраические операции над нечеткими отношениями аналогичны операциям с нечетким множествам.

Композиция (свертка) двух нечетких отношений

Пусть R_1 – нечеткое отношение R_1: (X x Y) →[0,1] между X и Y, и R_2 – нечеткое отношение R_2: (Y x Z) →[0,1] между Y и Z. Нечеткое отношение между X и Z обозначается $R_1 \bullet R_2$ и определяется как

$$\mu_{R_1 \bullet R_2}(x, y) = \bigvee_y \left[\mu_{R_1}(x, y) \wedge \mu_{R_2}(y, z) \right] = \max\left(\min\left(\mu_{R_1}(x, y), \mu_{R_2}(y, z) \right) \right),$$

где символ $\bigvee_y$ – обозначает операцию выбора наибольшего по y значения и называется (max-min)-сверткой отношений R_1 и R_2.

Пример. Пусть заданы отношения R_1 и R_2.

Таблица 1

R_1	y_1	y_2	y_3
x_1	0,1	0,7	0,4
x_2	1	0,5	0

Таблица 2

R_2	z_1	z_2	z_3	z_4

y_1	0,9	0	1	0,2
y_2	0,3	0,6	0	0,9
y_3	0,1	1	0	0,5

$$\mu_{R_1 \bullet R_2}(x_1, y_1) = \left[\mu_{R_1}(x_1, y_1) \wedge \mu_{R_2}(y_1, z_1)\right] \vee \left[\mu_{R_1}(x_1, y_2) \wedge \mu_{R_2}(y_2, z_1)\right] \vee$$
$$\vee \left[\mu_{R_1}(x_1, y_3) \wedge \mu_{R_2}(y_3, z_1)\right] =$$
$$= (0{,}1 \wedge 0{,}9) \vee (0{,}7 \wedge 0{,}3) \vee (0{,}4 \wedge 0{,}1) = 0{,}1 \vee 0{,}3 \vee 0{,}1 = 0{,}3.$$

Таблица 3

$R_1 \cdot R_2$	z_1	z_2	z_3	z_4
x_1	0,3	0,6	0,1	
x_2	0,9	0,5	1	

Нечеткие выводы

В экспертных и управляющих системах механизм нечетких выводов в своей основе имеет базу знаний, формируемую специалистами предметной области в виде совокупности нечетких предикатных правил вида:

$П_1$: если x есть A_1, то y есть B_1,

$П_2$: если x есть A_2, то y есть B_2,

…

$П_n$: если x есть A_n, то y есть B_n,

где x – входная переменная, y – переменная вывода, A и B – функции принадлежности, определенные на x и y соответственно.

Знания эксперта $A \rightarrow B$ отражает нечеткое причинное отношение предпосылки и заключения, поэтому его называют нечетким отношением:

$R = A \rightarrow B$,

где «$\rightarrow$» – нечеткая импликация.

Отношение R можно рассматривать как нечеткое подмножество прямого произведения $X \times Y$ полного множества предпосылок X и заключений Y. Таким образом, процесс получения (нечеткого) результата вывода B' с использованием данного наблюдения A' и значения $A \rightarrow B$ можно представить в виде

$B' = A' \bullet R = A' \bullet (A \rightarrow B)$.

Алгоритм нечеткого вывода

1 ***Нечеткость*** (фаззификация, fuzzification). Функции принадлежности, определенные для входных переменных, применяются к их фактическим значениям для определения степени истинности каждой предпосылки каждого правила).

2 ***Логический вывод***. Вычисленное значение истинности для предпосылок каждого правила применяется к заключениям каждого правила. Это приводит к одному нечеткому подмножеству, которое будет назначено переменной вывода для каждого правила. В качестве правил логического вывода используются только операции min (минимума) или prod (умножение).

3 ***Композиция***. Нечеткие подмножества, назначенные для каждой переменной вывода (во всех правилах), объединяются вместе, чтобы сформировать одно нечеткое подмножество для каждой переменной вывода. При подобном объединении обычно используются операции max (максимум) или sum (сумма).

4 ***Дефаззификация*** – приведение к четкости (defuzzification). Преобразование нечеткого набора выводов в число.

Алгоритмы нечеткого вывода Мамдани (Mamdani)

Пусть заданы два нечетких правила:

$П_1$: если x есть A_1 и y есть B_1, тогда z есть C_1,

П$_2$: если x есть A_2 и y есть B_2, тогда z есть C_2.

1. ***Нечеткость***. Находят степени принадлежности для предпосылок каждого правила: $A_1(x_0)$, $A_2(x_0)$, $B_1(y_0)$, $B_2(y_0)$.

2. ***Нечеткий вывод***. Определяют уровни «отсечения» для предпосылок каждого правила (операция min):

$\alpha_1 = A_1(x_0) \wedge B_1(y_0)$,

$\alpha_2 = A_2(x_0) \wedge B_2(y_0)$,

определяют усеченные функции принадлежности

$C'_1 = (\alpha_1 \wedge C_1(z))$,

$C'_2 = (\alpha_2 \wedge C_2(z))$.

3. ***Композиция***. Производится объединение найденных усеченных функций (операция max), получают нечеткое подмножество для переменной выхода с функцией принадлежности:

$$\mu_{\Sigma}(z) = C(z) = C'_1(z) \vee C'_2(z) = (\alpha_1 \wedge C_1(z)) \vee (\alpha_2 \wedge C_2(z)).$$

4. ***Дефаззификация***. Приведение к четкости (определение z_0), например, *центроидным* методом (как центр тяжести для кривой $\mu_{\Sigma}(z)$):

$$z_0 = \frac{\int\limits_{\Omega} z\mu_{\Sigma}(z)dz}{\int\limits_{\Omega} \mu_{\Sigma}(z)dz}.$$

Алгоритм иллюстрируется на рисунке 2.

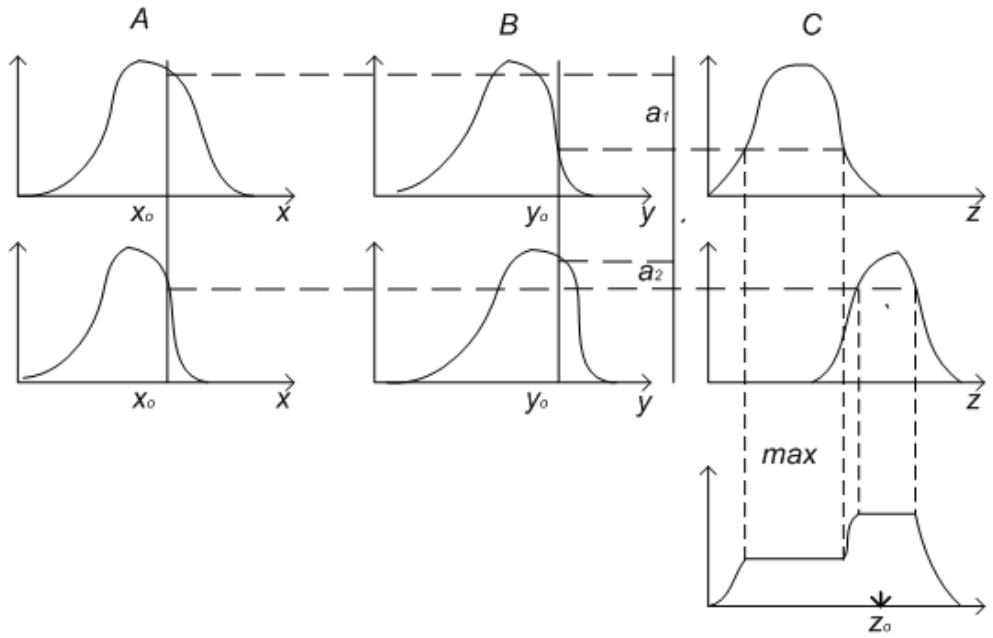


Рисунок 2 – Графическая интерпретация алгоритма Мамдани

Алгоритмы нечеткого вывода Сугено (Sugeno) и Такаги (Takagi)

Сугено (Sugeno) и Такаги (Takagi) использовали набор правил в следующей форме:

$П_1$: если x есть A_1 и y есть B_1, тогда $z=a_1x+b_1y$,

$П_2$: если x есть A_2 и y есть B_2, тогда $z=a_2x+b_2y$.

1 Первый этап – аналогично алгоритму Мамдани.

2 Определение $\alpha_1= A_1(x_0) \wedge B_1(y_0)$, $\alpha_2= A_2(x_0) \wedge B_2(y_0)$ и индивидуальные выходы правил:

$z^*_1=a_1x_0+b_1y_0$,

$z^*_2=a_2x_0+b_2y_0$.

3 Определяется значение переменной вывода:

$$z_0 = \frac{\alpha_1 z_1^* + \alpha_2 z_2^*}{\alpha_1 + \alpha_2}.$$

Представленная форма правил иллюстрирует алгоритм Сугено 1-го порядка. Если правила записаны в форме:

$П_1$: если x есть A_1 и y есть B_1, тогда $z=c_1$,

$П_2$: если x есть A_2 и y есть B_2, тогда $z=c_2$,

то задан алгоритм Сугено 0-го порядка.

Иллюстрация алгоритма Сугено 0-го порядка представлена на рисунке 3.

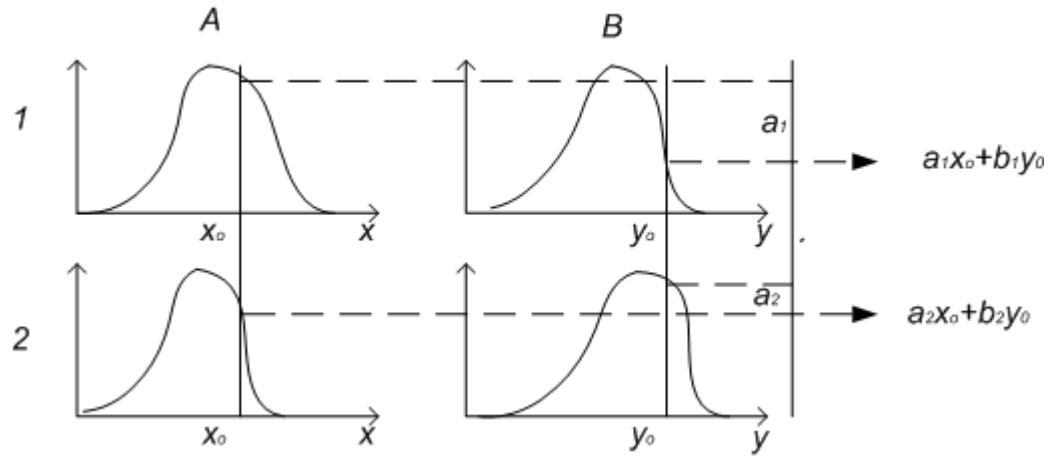


Рисунок 3 – Графическая интерпретация алгоритма Сугено

Перечислим наиболее известные методы дефаззификации.

Метод Максимума – выбирается тот элемент нечеткого множества, который имеет наивысшую степень принадлежности этому множеству.

Если этот элемент не является единственным, т.е. функция принадлежности $\mu_A(y_1)$ имеет несколько локальных максимумов, или если имеется максимальное «плато», то выбор среди элементов, имеющих наивысшую степень принадлежности множеству, осуществляется на основе некоторого критерия.

Метод левого (правого) максимума – выбирается наименьшее (наибольшее) из чисел y_1, y_2, .. ,y_n, имеющих наивысшую степень принадлежности нечеткому множеству.

Метод среднего из максимумов – в качестве искомого четкого значения y_o принимается среднее арифметическое координат локальных максимумов $y_o = \frac{1}{n}\sum_{1}^{n} y_i$.

Возможные методы дефаззификации и их обозначения в MATLAB приведены на рисунке 4: наименьший максимум (smallest of max, som), наибольший максимум (largest of max, lom), средний максимум (mean of max, mom), бисекторный (bisector of area), рассмотренный выше центроидный (centroid).

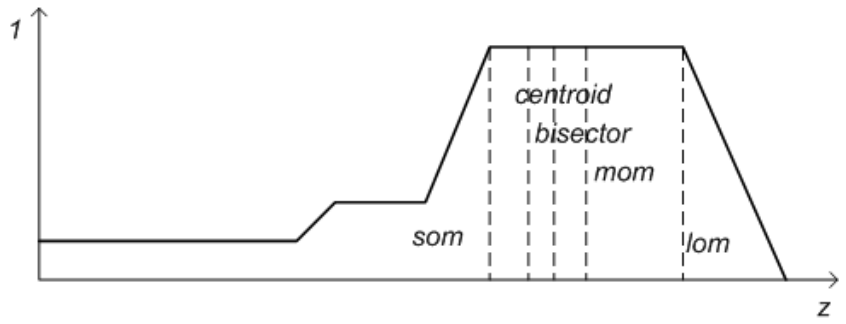


Рисунок 4 – Иллюстрация примеров фаззификации z

Пример проектирования системы нечеткого вывода в среде MATLAB

В качестве примера рассмотрим построение нечеткой аппроксимирующей функции $y=x^2$. Исходные данные для аппроксимации представлены в таблице 4.

Таблица 4 – Значения x и y

X	-1	-0,6	0	0,4	1
Y	1	0,36	0	0,16	1

1. Командой **fuzzy** из режима командной строки запускается основная интерфейсная программа пакета Fuzzy Logic – редактор нечеткой системы вывода.

2. В меню **File** выбрать команду New Sugeno FIS.

3. Выбрать входной элемент системы *input1* и ввести в поле *Name* обозначение входной переменной (*x)*.

4. Войти в режим редактирования функции принадлежности – *Membership Function Editor* (двойное нажатие левой клавиши мыши). Выбором **Edit/Add MFs** (добавить функцию принадлежности) задать тип функции принадлежности (gaussmf) и количество (5).

5. Установить диапазон (*Range*) изменения *x* от -1 до 1 (определяется заданным диапазоном *x*).

6. Совместить ординаты максимумов функций принадлежности с значениями аргумента *x*.

Шаг выполняется двумя способами: выделить редактируемую функцию принадлежности, перетащить мышью кривую функции принадлежности; более точную установку проводят заданием числовых значений параметров функции принадлежности в поле *Params* (первое значение определяет размах кривой, второе – положение центра).

В поле *Name* вводится имя функции принадлежности (1- bn, 2 – n, 3 (центральная) – z, 4 – p, 5 – bp).

Выйти из редактора функций принадлежности – *Close*.

7. Ввести в поле название выходной переменной *output1* (*y)*, войти в режим редактирования функций принадлежности.

8. Задать вид функции принадлежности для выходной переменной. Предлагается выбрать в качестве функции принадлежности линейные (*linear*) или постоянные (*constant*) – в зависимости от алгоритма Сугено (1-го или 0-го порядка). В поставленной задаче необходимо выбрать постоянные функции принадлежности с общим числом 4 (по числу различающихся значений *y*). Нажать Ok.

9. Установить диапазон (*Range*) – [0, 1]. Изменить значения (*Params*) и задать для выходных переменных имена (*Name*) соответственно 0; 0,16; 0,36; 1. Закрыть редактор.

10. Перейти в редактор привил (*Rule Editor*) (дважды щелкнуть на средний белый квадрат разрабатываемой структуры системы нечеткого вывода). При вводе каждого правила необходимо обозначить соответствие между каждой функцией принадлежности аргумента *x* и числовым значением *y*. Кривая, обозначенная bn, соответствует *y=1*, для чего выбирается в левом поле (с заголовком *x is*) вариант bn, а в правом – 1 и нажимается кнопка *Add rule*.

Введенное правило появится в окне правил в виде

If (x is bn) then (y is 1) (1).

Аналогично вводятся все правила (всего 5).

Закрыть окно редактора правил и вернуться в окно FIS-редактора.

Построение системы закончено.

Сохранить на диске (**File/Save to disk as**) созданную систему. Перейти в редактор функций принадлежности (**View/Edit Membership function**). Из окна редактора командой можно перейти в окно просмотра правил (**View rules**), просмотра поверхности (**View surface**).

В окне просмотра правил иллюстрируется процесс принятия решения (вычисления *y*). Красная вертикальная черта, пересекающая графики в правой части окна, которую можно перемещать с помощью мыши, позволяет изменять значения переменной входа (либо вводят значение с клавиатуры в поле *Input*), при этом соответственно изменяется значение выхода. Просмотр кривой *y(x)* осуществляется командой **View/View surface**. Большая погрешность аппроксимации заданной зависимости объясняется малым числом экспериментов. С помощью рассмотренных редакторов на любом этапе проектирования нечеткой модели можно внести необходимые коррективы, например задание пользовательской функции принадлежности.

В рассмотренном примере использованы следующие операции, устанавливаемые по умолчанию в алгоритме Сугено: логический вывод организуется с помощью операции логического умножения (prod); композиция организуется с помощью операции логической суммы (вероятностного ИЛИ, probor); приведение к четкости (дефаззификация) организуется дискретным вариантом центроидного метода (взвешенным средним, wtaver).

Задание на лабораторную работу

Создать систему нечеткого вывода для аппроксимации функции $y=k\cdot x^2$ (вариант см. в таблице 6). Для аппроксимации взять 50 значений функции.

Таблица 6 – Варианты задания для аппроксимации функции

Вариант	1	2	3	4	5	6	7	8
k	1.5	1.8	1.3	1.9	2	2.2	1.4	2.5
mf input	trimf	gaussmf	trimf	gaussmf	trimf	trimf	trimf	gaussmf2

9	10	11	12	13	14	15	16	17	18
2.3	2.4	3.5	3.8	3.3	3.9	3	3.2	3.4	4
trimf	gaussmf	trimf	gaussmf	trimf	gaussmf	trimf	trimf	trimf	gaussmf

Содержание отчета

1. Цель работы.

2. Исходные данные для задачи. Правила нечеткого вывода, форма функций принадлежности на всем диапазоне значений аргумента, результат вывода в графическом виде, относительная погрешность аппроксимации. Результат оформить в форме таблицы:

X				
Y				
$y^{аппрокс.}$				
σ, %				

3. Выводы по работе.

Контрольные вопросы

1. Что такое нечеткое множество?
2. Что характеризует функция принадлежности?
3. Какие методы дефазификаци вам известны?
4. В чем суть нечеткого вывода по Мамдани?
5. В чем суть нечеткого вывода по Сугено
6. На чем основывается нечеткий вывод?

Лабораторная работа № 2

Исследование системы автоматического регулирования с нечетким регулятором

Цель работы: Получить навык разработки нечеткого регулятора на базе продукционной модели представления знаний.

Краткие сведения из теории

Для создания одноконтурной системы управления воспользуемся программой Simulink пакета MATLAB. В Simulink необходимо собрать схему изображенную на рисунке 1.

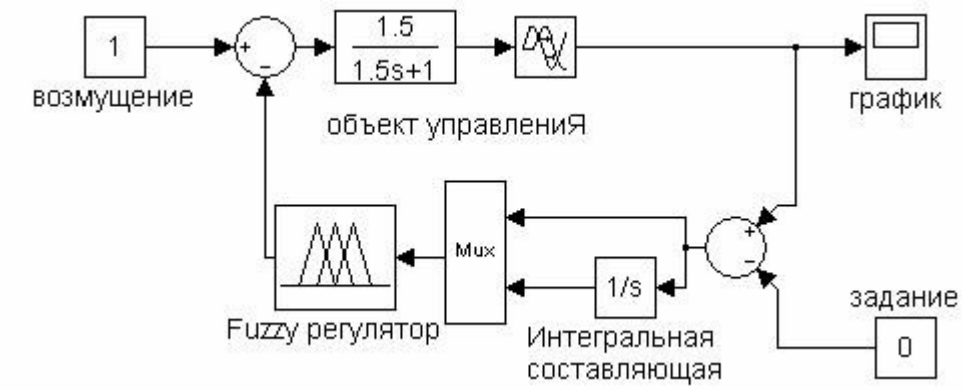


Рисунок 1 – Модель одноконтурной системы автоматического регулирования с ПИ-подобным fuzzy-регулятором

Теперь при помощи инструментов графического интерфейса пользователя (GUI) пакета "Fuzzy Logic Toolbox" создадим нечёткую систему, реализующую типовой аналоговый ПИ-регулятор. Заметим, что с помощью пакета "Fuzzy Logic Toolbox" можно строить нечеткие системы двух типов - Мамдани и Сугэно. Остановимся на системе типа Мамдани. Командой fuzzy в окне MATLAB вызываем окно Редактора фази-инференционной системы (Fuzzy Inference System Editor), выбираем тип системы - Мамдани, задаём два входа – для пропорциональной и интегральной составляющих и называем входные переменные, например, x1 и x2, а выходную – y.

Из данного окна вызываем окно Редактора функций принадлежности (Membership Function Editor) двойным щелчком мыши по изображению переменной x1 или при помощи меню Edit. Здесь для лингвистического описания каждой переменной выберем семь треугольных термов (NB,NM,NS,ZE,PS,PM,PB). Термы выходной переменной лучше выбирать непересекающимися. Это повысит чёткость регулирования. В этом же окне зададим диапазоны изменения переменных:

Для входных переменных регулятора рекомендуются симметричные диапазоны изменения, при этом

$$x1 \in \left[-\frac{1}{k_1}; \frac{1}{k_1} \right] \text{ и } x2 \in \left[-\frac{1}{k_0}; \frac{1}{k_0} \right],$$

где K_1, K_0 – оптимальные настройки пропорциональной и интегрирующей частей ПИ – регулятора в смысле какого-либо критерия.

Для выходной переменной регулятора диапазон изменения рекомендуется брать в виде $y \in [0; C]$, где верхняя граница C при единичном

ступенчатом воздействии варьируется от 1.1 до 2, чтобы выходной сигнал регулятора мог компенсировать это возмущение. По мере увеличения значения C уменьшается динамическая ошибка, но возрастают время регулирования и число колебаний переходного процесса. Поэтому рекомендуется C принимать равным 2, когда наблюдается оптимальное соотношение между величиной динамической ошибки, времени регулирования и количеством колебаний.

Теперь необходимо сформировать базу правил fuzzy-регулятора. В основу положен способ, предложенный в литературе. Линейный непрерывный ПИ-регулятор с дифференциальным уравнением

$$y(t) = k_D \cdot \varepsilon(t) + \frac{1}{T_E}\int_0^t \varepsilon(t) \cdot dt$$

можно заменить близким по стратегии и логике управления fuzzy-регулятором, если в качестве его выходной переменной рассматривать приращение управляющего воздействия Δy. Тогда ПИ закон регулирования можно представить в следующей дифференциальной форме:

$$\frac{dy(t)}{dt} = k_D \frac{d\varepsilon(t)}{dt} + \frac{1}{T_E}\varepsilon(t),$$

или в разностной форме:

$$\Delta y(k) = y(k) - y(k-1) = k_D \cdot \Delta\varepsilon(k) + \frac{\Delta t}{T_E} \cdot \varepsilon(k).$$

Таким образом, для входных переменных ε(k) и Δε(k) и выходной Δy(k) может быть синтезирован fuzzy-регулятор, реализующий нелинейный закон

$$\Delta y(k) = F\left[\Delta\varepsilon(k), \varepsilon(k)\right]$$

и эквивалентный в определённом смысле ПИ-регулятору.

Для нашего случая x1 соответствует сигналу рассогласования ε(k), x2 соответствует приращению сигнала рассогласования Δε(k), а y соответствует Δy(k). Лингвистические правила для такого ПИ-подобного fuzzy-регулятора приведены в таблице 1.

Вызываем окно Редактора правил (Rule Editor) в меню Edit или нажатием Ctrl+3 и заполняем список правилами из таблицы 1. Правила формируются по типу: ЕСЛИ … И …, ТО…. Можно посмотреть пространство управления, вызвав окно Просмотра пространства управления(Surface Viewer) из меню View или комбинацией клавиш Ctrl+6. Полученный файл сохраним под именем **fuzzy1.fis**.

В окне параметров блока Fuzzy Logic Controller укажем имя файла **fuzzy1**. В окне модели в меню **File** выберем пункт **Model Properties**. В открывшемся окне выберем вкладку **CallBacks** и в поле **Model pre-load function** напишем:

```
fuzzy1=readfis('fuzzy1').
```

Данная команда будет каждый раз при открытии файла модели помещать файл **fuzzy1.fis** в WorkSpace (рабочее пространство системы MATLAB). Это необходимо для нормального функционирования модели. Стоит заметить, что при внесении изменений в fis-файл нужно помещать его исправленную версию в WorkSpace либо при помощи пункта Export/ To Workspace меню File, либо комбинацией клавиш Ctrl+T, либо каждый раз закрытием и открытием файла модели.

В диалоговом окне **Simulation Parameters** меню **Simulation** во вкладке **Advanced** для опции **Boolean logic signals** необходимо установить значение **off**. При этом блоки логики будут допускать переменные в форме с плавающей точкой.

Запуск модели – **Simulation/Start** (Ctrl+T).

В папке Demo находится файл с правилами нечеткого вывода. Загрузка файла осуществляется из окна редактора правил Rule Viewer: во вкладке **File/Import From Disk** открыть файл fuzzy1.fsi. Затем нужно поместить правила в WorkSpace системы (описано выше).

Таблица 1 – Лингвистические правила для ПИ fuzzy-регулятора

ε Δε	NB	NM	NS	ZE	PS	PM	PB
NB	NB	NB	NB	NB	NM	NS	ZE
NM	NB	NB	NB	NM	NS	ZE	PS
NS	NB	NB	NM	NS	ZE	PS	PM
ZE	NB	NM	NS	ZE	PS	PM	PB
PS	NM	NS	ZE	PS	PM	PB	PB
PM	NS	ZE	PS	PM	PB	PB	PB
PB	ZE	PS	PM	PB	PB	PB	PB

Работу разработанного fuzzy-регулятора можно сравнить с аналоговой системой регулирования (рисунок 2).

Дифференциальное уравнение ПИД-регулятора имеет вид:

$$y(t) = k_D \cdot \varepsilon(t) + \frac{1}{T_E}\int_0^t \varepsilon(t) \cdot dt + T_Д \cdot \frac{d\varepsilon(t)}{dt}.$$

Для исследования ПИ-регулятора в блоке настроек регулятора необходимо в поле параметра $T_Д$ (настройка дифференцирующей части ПИД-регулятора) установить значение 0.

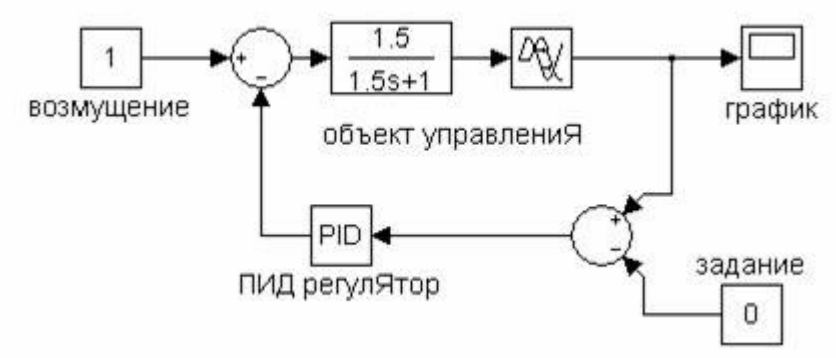


Рисунок 2 – Модель одноконтурной системы автоматического регулирования

Задание на лабораторную работу

Исследовать АСР с аналоговым ПИ-регулятором и fuzzy-регулятором. В качестве модели объекта принять апериодическое звено, параметры объекта принять из таблицы 2.

Таблица 2 – Параметры объекта к заданию 5.2

Вариант	1	2	3	4	5	6	7	8
K	1.5	1.8	1.3	1.9	2	2.2	1.4	2.5
T	1.8	1.8	1.5	1.5	1.3	1.5	1.5	1.6

9	10	11	12	13	14	15	16	17	18
2.3	2.4	3.5	3.8	3.3	3.9	3	3.2	3.4	4
1.4	1.7	1.8	1.5	1.5	1.5	1.8	1.2	1.4	1.9

Настройки ПИ-регулятора определить по формульному методу:

Регулятор	Апериодический процесс	Процесс с перерегулированием 20 %	Процесс с минимальным временем регули-

			рования
ПИ	$K_1 = \frac{0,6 \cdot T}{K \cdot \tau}$, $K_0 = \frac{1}{K \cdot \tau}$	$K_1 = \frac{0,7 \cdot T}{K \cdot \tau}$, $K_0 = \frac{1}{K \cdot \tau}$	$K_1 = \frac{T}{K \cdot \tau}$, $K_0 = \frac{1}{K \cdot \tau}$

Содержание отчета

1 Цель работы.

2 Исходные данные для задачи. Блок схема моделируемой системы, графики переходных процессов для аналогового и нечеткого регуляторов. Параметры элементов системы нечеткого вывода проектируемой системы (тип функций принадлежности, способ реализации логических выражений, способ дефаззификации). Анализ качества переходных процессов.

3 Выводы по работе.

Контрольные вопросы

1. Какие критерии качества переходного процесса свидетельствуют о преимуществе классического регулятора, а какие – о преимуществе нечеткого регулятора?

2. Как формируется набор правил для нечеткого регулятора?

3. Есть ли преимущества в настройке нечеткого регулятора?

4. Изобразите структуру нечеткого регулятора.

Лабораторная работа № 3

Изучение основных элементов и возможностей экспертной оболочки CLIPS

Цель работы: Изучить основные возможности экспертной оболочки CLIPS

Краткие сведения из теории

Первоначально аббревиатура CLIPS была названием языка — C Language Integrated Production System (язык С, интегрированный с продукционными системами), удобного для разработки баз знаний и макетов экспертных систем. Язык был разработан в Центре космических исследований NASA (NASA's Johnson Space Center) в середине 1980-х годов.

Его появление было обусловлено тем, что прежняя технология для построения систем использовала для реализации язык LIPS, имевшего ряд недостатков: высокая цена технического и программного обеспечения, предназначенного для работы с LISP, недостаточная адаптируемость к широкому кругу стандартных компьютеров и низкая способность интеграции систем, написанных на LISP, с системами, написанными на других языках (производство вложенных приложений). Отличительной чертой CLIPS стало использование в качестве языка реализации язык С, что решило значительную часть проблем.

Первые версии (с 1-й по 4-ю) CLIPS поддерживали только продукционную парадигму (основанную на правилах), суть которой заключалась в том, что для построения экспертной системы используется продукционная МПЗ (модель представления знаний). Знания в такой модели представляются в виде предложений типа "Если (условие), то (действие)". Под “условием” понимается некоторое предложение-образец, по которому осуществляется поиск в базе данных, а под “действием” – действия, выполняемые при успешном исходе поиска. Продукционная, модель часто применяется в промышленных экспертных системах (более 80% используют именно ее). Это объясняется тем, что эта модель привлекает разработчиков своей наглядностью, высокой модульностью, легкостью внесения дополнений и изменений и простотой механизма логического вывода.

Но в 1991 году вышла версия 5.0. В ней была реализована поддержка двух новых парадигм программирования: традиционное процедурное и объектно-ориентированное программирование.

Объектно-ориентированный язык программирования, предоставляемый системой CLIPS, называется CLIPS Object-Oriented Language (COOL).

В программе на языке CLIPS стало возможным вызывать функции, написанные на языке C. Исполняющая система CLIPS может выступать в качестве внедренного приложения, т.е. программа на CLIPS может быть скомпилирована и скомпонована с программой на языке C, которая будет вызывать CLIPS-фрагменты как подпрограммы. Это позволяет внедрять функции искусственного интеллекта в компоненты больших программных комплексов.

Существующая в настоящее время версия является многофункциональной средой программирования, современным инструментальным средством для создания экспертной системы(expert system tool). Она может эксплуатироваться на платформах UNIX, DOS, Windows и Macintosh. Она является хорошо документированным общедоступным программным продуктом и доступна по сети FTP с множества университетских сайтов. Исходный код программного пакета CLIPS распространяется совершенно свободно и его можно установить на любой платформе, поддерживающей стандартный компилятор языка C. Разработка CLIPS помогла усовершенствовать возможности технологии производства экспертных систем среди широкого диапазона приложений.

CLIPS имеет модульную архитектуру. Ее характерная особенность заключается в разбиении продукта на почти независимые универсальные элементы. CLIPS состоит из интерактивной среды – экспертной оболочки,

гибкого и мощного языка и нескольких особенностей, позволяющих говорить о CLIPS как о полноценном инструментарии: встроенный редактор и средство отладки. Понятие «оболочка» отвечает за ту часть CLIPS, которая осуществляет рассуждения и выводы. Она обеспечивает основные элементы экспертной системы:

- База данных (содержит в себе факты и переменные)
- Продукционная база знаний (содержит в себе правила)
- Машина логического вывода.

Машина логического вывода (интерпретатор правил) – это программа, имитирующая логический вывод эксперта, пользующегося данной продукционной базой знаний для интерпретации поступивших в систему данных.

Благодаря тому, что CLIPS является свободно распространяемым программным продуктом с доступными исходными кодами, в последнее время было выпущено множество программ и библиотек, усовершенствующих и дополняющих возможности CLIPS. Примеры некоторых из них:

DYNACLIPS (динамические утилиты CLIPS) – включает доску объявлений, механизм динамического обмена знаниями и инструментальные средства для CLIPS v5.1 и v6.0;

FuzzyCLIPS 6.02 – версия CLIPS, оболочка экспертной системы, основанная на правилах, используется для представления и управления нечеткими фактами и правилами. В дополнение к функциональным возможностям CLIPS, FuzzyCLIPS может иметь дело с точными, нечеткими (или неточными) знаниями, сложными рассуждениями, которые можно свободно смешивать в правилах и фактах экспертной системы;

WxCLIPS снабжает CLIPS v5.1, CLIPS v6.0 и CLIPS v6.0 с нечетким представлением знаний простым графическим внешним интерфей-

ком. Официальный сайт CLIPS располагается по адресу: http://www.ghg.net/clips/CLIPS.html. Этот сайт поможет вам получить как сам CLIPS, так и всевозможный материал для его изучения и освоения (документацию, примеры, советы специалистов, исходные коды и многое другое).

Отличия COOL (CLIPS Object-Oriented Language) от других объектно-ориентированных языков

В так называемых "чистых" объектно-ориентированных языках абсолютно все программные элементы являются объектами, и любые действия над ними выполняются посредством посылки сообщений. В CLIPS объектами являются только объекты классов, определенных пользователем, и объекты, представляющие данные примитивных типов CLIPS. С объектами, представляющими данные примитивных типов, можно манипулировать с помощью сообщений, а для объектов классов, определенных пользователем, это является единственно возможным способом работы с объектом.

Работа со всеми программными элементами CLIPS, не являющимися объектами, выполняется не в объектно-ориентированном стиле, а с помощью вызовов соответствующих функций.

Основные возможности ООП

Любая объектно-ориентированная система должна обладать следующими пятью характеристиками: абстрактностью, инкапсуляцией, наследованием, полиморфизмом и динамическим связыванием. Абстракция – это способ представления данных на определенном уровне некоторой конкретной проблемной области. Инкапсуляция – это процесс, позволяющий скрывать детали реализации объекта с помощью некоторого определенного для этого класса внешнего интерфейса. Наследование поз-

воляет определять классы, использующие определения других классов и обладающие всеми их и некоторыми своими свойствами, если это необходимо. Полиморфизм – свойство, благодаря которому различные объекты по-разному реагируют на одни и те же сообщения. Динамическое связывание является возможностью выбирать определенный обработчик сообщения объекта во время выполнения программы. Рассмотрим теперь, как CLIPS реализует все эти основные свойства системы ООП.

Создание нового класса реализует возможность абстрактного представления нового типа данных. Слоты и обработчики сообщений этого класса определяют свойства и поведение целой группы объектов, принадлежащих этому классу.

Инкапсуляция реализуется в CLIPS требованием обязательно использовать сообщения при работе с объектами определенных пользователем классов. Обработчики сообщений класса представляют собой доступный пользователю интерфейс, скрывающий реализацию класса.

COOL поддерживает множественное наследование. Это означает, что некоторый класс может обладать всеми свойствами указанного одного или более суперкласса. Для установления линейного порядка наследования свойств классов при множественном наследовании COOL использует список предшествования классов (class precedence list), построенный с использованием иерархии наследования. Объект, представляющий собой экземпляр нового класса, наследует все свойства (слоты) и поведение (обработчики сообщений) каждого класса из списка предшествования классов. Слово «предшествование» обозначает, что свойства и поведение класса, находящегося ближе к началу списка, переопределяют конфликтующие определения ранее встретившихся классов.

Различные COOL-объекты могут реагировать на одно и то же сообщение совершенно по-разному. Это реализует свойство полиморфизма. На практике это выполняется присоединением к разным классам обработчиков одного и того же сообщения, но с разными последовательностями выполняемых действий.

CLIPS также поддерживает возможность динамического связывания, реализуемую с помощью функции *send*, предназначенной для посылки сообщений объекту. Вызов этой функции осуществляется именно в процессе выполнения программы, таким образом, определение обработчика выполняющегося в тот или иной момент также происходит в процессе выполнения программы.

Запросы и наборы объектов

В дополнение к возможности использовать объекты в процессе сопоставления образцов правил, COOL поддерживает гибкую систему запросов, позволяющую использовать заданные пользователем критерии для выборки некоторого набора объектов и выполнения над ним определенных действий. Запросы позволяют объединять в наборы объекты самых разных классов. Запросы можно использовать, например, для проверки существования того или иного набора объектов, выполнения действий над набором или сохранения ссылки на набор для последующего использования.

Эвристические знания

Одним из основных методов представления знаний в CLIPS являются правила. Правила используются для представления эвристик или эмпирических правил, определяющих действия, которые необходимо выполнить в случае возникновения некоторой ситуации. Разработчик экспертной системы создает набор правил, которые, работая вместе, решают

поставленную задачу. Правила состоят из предпосылок и следствия. Предпосылки называются также ЕСЛИ-частью правила или LHS правила (left-hand side). Следствие называются ТО-частью правила или RHS правила (right-hand side).

Предпосылки правила представляют собой набор условий (или условных элементов), которые должны удовлетвориться, для того чтобы правило выполнилось. Предпосылки правил удовлетворяются в зависимости от наличия или отсутствия некоторых заданных фактов в списке фактов или некоторых созданных объектов, являющихся экземплярами классов, определенных пользователем. Один из наиболее распространенных типов условных выражений в CLIPS – образцы (patterns). Образцы состоят из набора ограничений, которые используются для определения того, удовлетворяет ли некоторый факт или объект условному элементу. Другими словами, образец задает некоторую маску для фактов или объектов. Процесс сопоставления образцов фактам или объектам называется сопоставлением образцов (pattern-matching). CLIPS предоставляет механизм, называемый механизмом логического вывода (inference engine), который автоматически сопоставляет образцы с текущим списком фактов и определенными объектами и ищет правила, которые применимы в настоящий момент.

Следствие правила представляется набором некоторых действий, которые нужно выполнить, в случае если правило применимо к текущей ситуации. Таким образом, действия, заданные вследствие правила, выполняются по команде механизма логического вывода, если все предпосылки правила удовлетворены. В случае если в данный момент применимо более одного правила, механизм логического вывода использует текущую стратегию разрешения конфликтов (conflict resolution strategy),

которая определяет, какое именно правило будет выполнено. После этого CLIPS выполняет действия, описанные вследствие выбранного правила (которые могут оказать влияние на список применимых правил), и приступает к выбору следующего правила. Этот процесс продолжается до тех пор, пока список применимых правил не опустеет.

В большинстве случаев правила CLIPS можно представить в виде операторов IF-THEN, используемых в процедурных языках программирования. Однако условные выражения IF-THEN в процедурных языках проверяются только тогда, когда поток управления программы непосредственно попадает на данное выражение путем последовательного перебора выражений и операторов, составляющих программу. В CLIPS, в отличие от этого, механизм логического вывода создает и постоянно модифицирует список правил, условия которых в данный момент удовлетворены. Эти правила запускаются на выполнение механизмом логического вывода

Процедурные знания

Помимо эвристической, CLIPS поддерживает и процедурную парадигму представления знаний, используемую в большинстве языков программирования. Конструкторы *deffunction* и *defgeneric* позволяют пользователю определять новые выполняемые конструкции непосредственно в среде CLIPS, возвращающие некоторые значения или выполняющие какие-то полезные действия. Вызов этих новых функций ничем не отличается от вызова встроенных функций CLIPS. Обработчики сообщений позволяют пользователю определять поведение объектов, с помощью задания той или иной реакции на сообщения. Функции, родовые функции и обработчики сообщений представляют собой отрезки кода, заданного пользователем и выполняемого, в случае необходимости,

интерпретатором CLIPS. Кроме того, механизм модулей позволяет разбивать базу знаний CLIPS на отдельные смысловые части.

Функции

Конструктор *deffunction* позволяет создавать новые функции непосредственно в CLIPS. Более ранние версии CLIPS позволяли использовать только внешние пользовательские функции, написанные на каком-нибудь языке программирования (чаще всего Си) и присоединенные к среде CLIPS.

Тело функции, определенной с помощью конструктора *deffunction*, представляет собой последовательность действий, подобную используемой в правой части правил. Заданные пользователем действия выполняются при вызове соответствующей функции. Значение, возвращаемое функцией, является результатом вычисления последнего действия.

Родовые функции

Родовые функции, так же как и обычные функции, могут быть созданы непосредственно в CLIPS. Способ вызова таких функций также ничем не отличается от способа вызова обычных функций. Однако родовые функции гораздо мощнее обычных, т. к. они способны перегружаться. Благодаря механизму перегрузки родовая функция может выполнять различные действия в зависимости от типа и числа аргументов. Обычно родовая функция состоит из нескольких компонентов, называемых методами. Каждый метод содержит различный набор аргументов родовой функции.

Обработчики сообщений

Поведение объекта обуславливается обработчиками сообщений, которые являются присоединенной к классу последовательностью действий с заданным именем. Любые манипуляции с объектом можно выполнить только с помощью сообщений. Назначение обработчиков сообщений в

принципе эквивалентно назначению любой функции – возвращение результата или выполнение неких полезных действий.

Модули

Модули позволяют разбивать базу знаний на отдельные смысловые части. Каждый вызываемый конструктор помещается в определенный модуль. Программист имеет возможность контролировать возможности доступа и видимость конструкторов в тех или иных модулях. Кроме того, для модулей можно устанавливать видимость определенных фактов или объектов. Модули можно использовать для контроля или изменения потока выполнения правил.

Для представления данных CLIPS использует три основных абстракции данных*:* факты, объекты и глобальные переменные.

Факты

Факты – одна из основных форм представления данных в CLIPS. Каждый факт представляет собой определенный набор данных, сохраняемый в текущем списке фактов – рабочей памяти системы. Объем списка фактов ограничен только памятью вашего компьютера. Список фактов хранится в оперативной памяти компьютера, но CLIPS предоставляет возможность сохранять текущий список в файл и загружать список из ранее сохраненного файла.

В системе CLIPS фактом является список неделимых (или атомарных) значений примитивных типов данных. CLIPS поддерживает два типа фактов – упорядоченные факты (ordered facts) и неупорядоченные факты или шаблоны (non-ordered facts или template facts). Ссылаться на данные, содержащиеся в факте, можно либо используя строго заданную позицию значения в списке данных для упорядоченных фактов, либо указывая имя значения для шаблонов.

Факты можно добавлять, удалять, изменять и дублировать, вводя соответствующие команды с клавиатуры, либо из программы.

После добавления факта в список фактов ему присваивается целый уникальный идентификатор, называемый индексом факта (fact-index). Индекс первого факта равен нулю, в дальнейшем индекс увеличивается на единицу при добавлении каждого нового факта. CLIPS предоставляет команды, очищающие текущий список фактов или всю базу знаний. Эти команды присваивают текущему значению индекса 0.

Некоторые команды, например изменения, удаления или дублирования фактов, требуют указания определенного факта. Факт можно задать либо индексом факта, либо его адресом. Адрес факта представляет собой переменную-указатель, хранящую индекс факта. Процесс создания адресов фактов будет описан ниже.

Объекты

Объект CLIPS состоит из двух основных частей – свойств объекта и его поведения. Класс представляет собой своеобразный шаблон, определяющий общие свойства и поведение объектов – экземпляров этого класса. Объекты могут принадлежать классам, определенным пользователем, или некоторым системным классам (например, классам, реализующим представление объектов в виде примитивных данных).

Объекты CLIPS разделяются на две важные категории: объекты, хранящие примитивные типы данных, и объекты классов, определенных пользователем. Объекты этих двух типов отличаются способом своего создания и удаления, наборами свойств и даже методом использования.

Объекты примитивных типов неявно создаются и удаляются системой CLIPS в местах, где это необходимо. По ссылке на такой объект можно получить хранящееся в нем значение соответствующего

типа. Объекты примитивных типов не имеют слотов и, как правило, не имеют имен. Классы, определяющие эти объекты, являются предопределенными классами CLIPS. Функциональность предопределенных классов, определяющих объекты примитивных типов, подобна функциональности классов, определенных пользователем, за исключением того, что к таким классам нельзя присоединить обработчики сообщений. Это делает не очень удобным использование таких классов в объектно-ориентированном программировании. Основным назначением объектов примитивных типов является использование их в определении родовых функций. Родовые функции применяют эти объекты в качестве своих аргументов и, по заданному набору, в момент вызова, определяют, какой именно метод родовой функции должен быть вызван.

Для ссылки на объект класса, определенного пользователем, необходимо использовать имя или адрес объекта. Такие объекты явно создаются или удаляются с помощью сообщений или специальных системных функций. Свойства объекта класса, определенного пользователем, определяются набором слотов, заданных при определении класса. Слот объекта имеет имя и может содержать простое или составное значение. Поведение объектов определяется наличием тех или иных обработчиков сообщений, присоединенных к соответствующему классу. Все объекты одного класса имеют одинаковые наборы слотов, но могут содержать в этих слотах различные значения. Однако если два объекта имеют одинаковые наборы слотов, это еще не означает, что они принадлежат одному классу, т. к. два абсолютно разных класса, теоретически, могут иметь одинаковые наборы слотов.

Основная разница между слотами объекта и неупорядоченного факта заключается в возможности наследования. Наследование позволяет ис-

пользовать в классе некоторые свойства и поведение, определенные в другом классе. COOL (объектно-ориентированная часть языка CLIPS) поддерживает множественное наследование, при котором класс может наследовать слоты и обработчики сообщений непосредственно от нескольких классов.

Глобальные переменные

Доступ к такой переменной можно получить из любого места среды CLIPS, а значения, которые они содержат, не зависят ни от каких других конструкций языка. Глобальные переменные CLIPS подобны глобальным переменным, встречающимся в таких традиционных процедурных языках. Однако, в отличие от них, глобальные переменные CLIPS являются слабо типизированными. Они способны хранить значение любого типа.

Задание на лабораторную работу

Задание 1. Ознакомиться с примером.

Данный пример наглядно продемонстрирует работу с фактами и правилами.

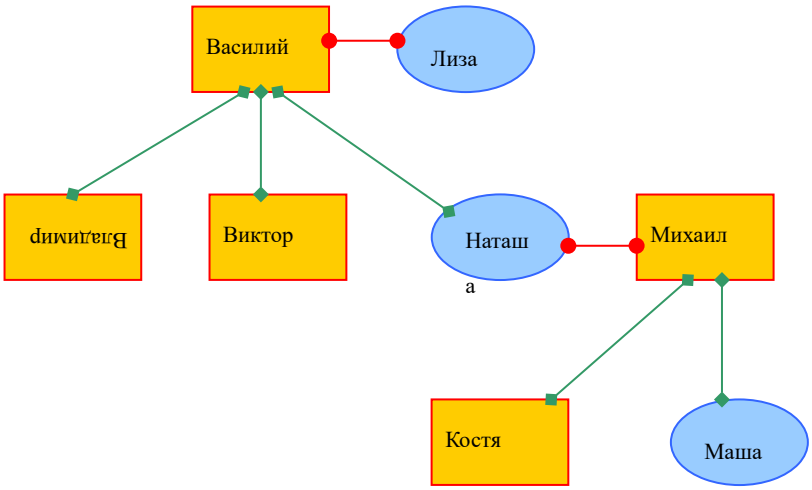


Рисунок 3 – Пример генеалогического дерева.

Описание структуры

Создадим шаблон для неупорядоченных фактов.

Для описания структуры генеалогического дерева достаточно четыре слота.

```
(deftemplate person
(slot name)
(slot gender)
(slot father)
(slot wife))
```

Для проверки добавления шаблона можно воспользоваться специальным инструментом Deftemplate Manager (Менеджер шаблонов), доступным в Windows-версии среды CLIPS. Для запуска менеджера шаблонов воспользуйтесь меню *Browse* и выберите пункт *Deftemplate Manager*.

Менеджер шаблонов позволяет в отдельном окне просматривать список всех шаблонов, доступных в текущей базе знаний, удалять выбранный шаблон и отображать все его свойства.

На основе шаблона person добавим список фактов, описывающих элементы структуры.

```
(deffacts  people
(person (name Vasya) (gender male) (wife Liza))
(person (name Liza) (gender female))
(person (name Vladimir) (gender male) (father Vasya))
(person (name Natasha) (gender female) (father Vasya))
(person (name Viktor) (gender male) (father Vasya))
(person (name Misha) (gender male) (wife Natasha))
(person (name Kostya) (gender male) (father Misha) (wife Liza))
(person (name Masha) (gender female) (father Misha)))
```

Для проверки добавления шаблона можно воспользоваться специальным инструментом Deffacts Manager (Менеджер предопределенных фактов). Для запуска менеджера шаблонов воспользуйтесь меню *Browse* и выберите пункт *Deffacts Manager*.

Определение отношения «Мать»

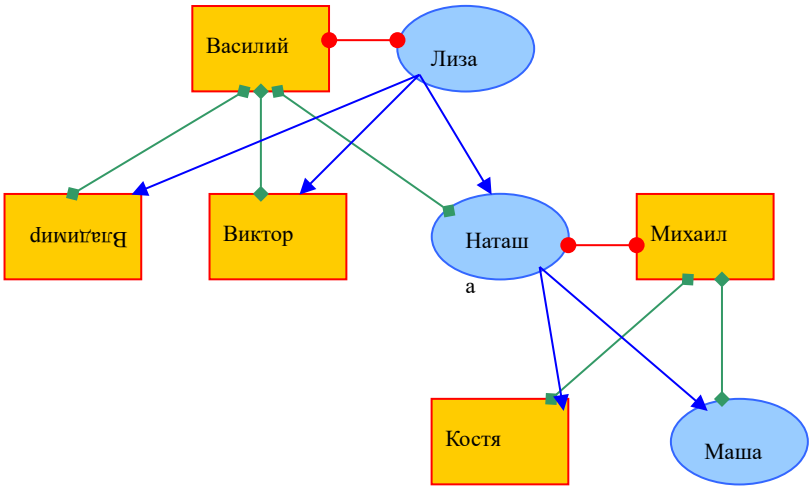


Рисунок 4 – Отображение отношения “Мать”

Создадим шаблон:

(deftemplate mother

(slot namel)

(slot name2))

Создадим правило, описывающее отношение:

(defrule Mother

(person (name ?x) (wife ?y))

(person (name ?z) (father ?x))

=>

(printout t ?y " is mother of " ?z crlf)

(assert (mother (namel ?y) (name2 ?z))))

Выполним команды:

(reset)

(run)

Результат:

Natasha mother of Masha

Natasha mother of Kostya

Liza mother of Viktor

Liza mother of Natasha

Liza mother of Vladimir

Определение отношения «Брат»

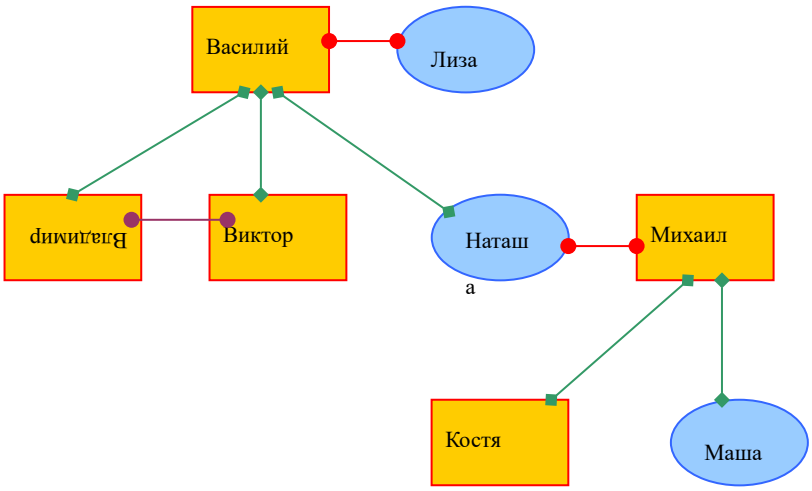


Рисунок 5 – Отображение отношения «Брат»

Создадим шаблон:

(deftemplate brother

(slot namel)

(slot name2))

Создадим правило, описывающее отношение:

(defrule Brother

(person (name ?x) (gender male) (father ?y&~nil))

(person (name ?z&~?x) (gender male) (father ?y&~nil))

(not (brother (namel ?x) (name2 ?z)))

(not (brother (namel ?z) (name2 ?x))) =>

(printout t ?x " brother of " ?z crlf)

(assert (brother (namel ?x) (name2 ?z))))

Ограничение ?z&~?x запрещает выводить бессмысленные пары одинаковых имен.

Ограничение ?y&~nill запрещает выводить пары, поля “отец” которых не определены (нулевое значение).

Условные элементы

(not (brother (namel ?x) (name2 ?z)))

(not (brother (namel ?z) (name2 ?x)))

проверяют наличие фактов типа brother и, тем самым отслеживают, была ли уже обработана данная пара или ее перестановка. Если эти факты отсутствуют, то это означает, что обработка еще не была выполнена. В этом случае правило активируется, и выполняются действия, описанные в правой части правила. А именно выводится на экран сообщение о найденной паре братьев и добавляется соответствующий факт brother, утверждающий, что данная пара уже была обработана.

Выполним команды:

(reset)

(run)

Результат:

Viktor brother of Vladimir

Задание 2. По аналогии с примером из Задания 1 описать свое генеалогическое дерево. При этом определить отношения:

- Мать;
- Брат;
- Сестра;

- Дедушка;
- Бабушка;
- Теща;
- Шурин (брат жены);
- Свояченица (сестра жены);
- Свояк (муж свояченицы);
- Свекор (отец мужа);
- Золовка (сестра мужа);
- Деверь (брат мужа);
- Сноха (жена сына для его матери);
- Невестка (жена сына для его отца).

Контрольные вопросы

1. В чем отличие COOL от других объектно-ориентированных языков?
2. Какие способ представления знаний поддерживает CLIPS?
3. Для чего нужны родовые функции и обработчики сообщений?
4. Что такое объект CLIPS и из чего он состоит?
5. Что является фактов в системе CLIPS?

Лабораторная работа № 4

Разработка прототипа диагностической экспертной системы

Цель работы: Исследовать предметную область, сформировать для нее поле знаний, список фактов, а также правил для работы с ними. Освоить принципы поиска решения в экспертных системах, основанных на правилах вида "ЕСЛИ-ТО", формирования последовательности активации правил при выводе результата.

Краткие сведения из теории

Экспертные системы, созданные с помощью CLIPS, могут быть запущены тремя основными способами: вводом соответствующих команд и конструкторов языка непосредственно в среду CLIPS; использованием интерактивного оконного интерфейса CLIPS (например для версий Windows или Macintosh); с помощью программ-оболочек, реализующих свой интерфейс общения с пользователем и использующих механизмы представления знаний и логического вывода CLIPS.

Windows-версия среды CLIPS полностью совместима с базовой спецификацией языка. Ввод команд осуществляется непосредственно в главное окно CLIPS. Однако по сравнению с базовой Windows-версия предоставляет множество дополнительных визуальных инструментов (например, менеджеры фактов или правил), значительно облегчающих жизнь разработчика экспертных систем.

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.

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=\".

Факты – одна из основных форм представления данных в CLIPS (существует также возможность представления данных в виде объектов и глобальных переменных, но об этом речь пойдет позже). Каждый факт представляет собой определенный набор данных, сохраняемый в текущем списке фактов – рабочей памяти системы. Список фактов представляет собой универсальное хранилище фактов и является частью базы знаний. Объем списка фактов ограничен только памятью вашего компьютера. Список фактов хранится в оперативной памяти компьютера, но CLIPS предоставляет возможность сохранять текущий список в файл и загружать список из ранее сохраненного файла.

В системе CLIPS фактом является список неделимых (или атомарных) значений примитивных типов данных. CLIPS поддерживает два типа фактов – упорядоченные факты (ordered facts) и неупорядоченные факты или шаблоны (non-ordered facts или template facts). Ссылаться на данные, содержащиеся в факте, можно либо используя строго заданную позицию значения в списке данных для упорядоченных фактов, либо указывая имя значения для шаблонов. Упорядоченные факты состоят из поля, обязательно являющегося данным типа symbol и следующей за ним, возможно пустой, последовательности полей, разделенных пробелами. Ограничением факта служат круглые скобки:

(поле_типа_symbol [поле]*)

Так как упорядоченный факт для представления информации использует строго заданные позиции данных, то для доступа к ней пользователь должен знать, не только какие данные сохранены в факте, но и какое поле содержит эти данные.

Неупорядоченные факты (или шаблоны) предоставляют пользователю возможность задавать абстрактную структуру факта путем назначения имени каждому полю. Для создания шаблонов, которые впоследствии будут применяться для доступа к полям факта по имени, используется конструктор deftemplate. Конструктор deftemplate аналогичен определениям записей или структур в таких языках программирования, как Pascal или C.

Конструктор deftemplate задает имя шаблона и определяет последовательность из нуля или более полей неупорядоченного факта, называемых также слотами. Слот состоит из имени, заданного значением типа symbol, и следующего за ним, возможно пустого, списка полей. Как и факт, слот с обеих сторон ограничивается круглыми скобками. В отличие от упорядоченных фактов слот неупорядоченного факта может жестко определять тип своих значений. Кроме того, слоту могут быть заданы значения по умолчанию. Синтаксис данного конструктора следующий:
(deftemplate <имя-шаблона> [<комментарии>] [<оперделение-слота>*])

Помимо фактов, CLIPS предоставляет еще один способ представления данных – глобальные переменные (globals). В отличие от переменных, связанных со своим значением в левой части правила, глобальная переменная доступна везде после своего создания (а не только в правиле, в котором она получила свое значение). Глобальные переменные CLIPS подобны глобальным переменным в процедурных языках программирования, таких как C или ADA. Однако, в отличие от переменных боль-

шинства процедурных языков программирования, глобальные переменные в CLIPS слабо типизированы. Фактически переменная может принимать значение любого примитивного типа CLIPS при каждом новом присваивании значения.

С помощью конструктора defglobal в среде CLIPS могут быть объявлены глобальные переменные и присвоены их начальные значения.

(defglobal [<имя-модуля>] <определение-переменной>*)

<определение-переменной> ::= <имя-переменной> = <выражение>

<имя-переменной> ::= ?*<значение-типа-symbol>*

CLIPS позволяет использовать произвольное количество конструкторов defglobal. Необязательный параметр <имя-модуля> указывает модуль, в котором должны быть определены конструируемые переменные. Если имя модуля не задано, то переменные будут помещены в текущий модуль.

Глобальные переменные применяются в любом месте, где могут быть использованы переменные, созданные в левой части правил с некоторыми исключениями. Во-первых, глобальные переменные не могут использоваться как параметры в конструкторах deffunction, defmethod или обработчиках сообщений. Во-вторых, глобальные переменные не могут использоваться для получения новых значений в левой части правил.

Неверно: (defrule example (fact ?*x*) =>).

Верно: (defrule example (fact ?y & :(> ?y ?*x*)) =>)

П р и м е р 1.

```
(defglobal
?*x* = 3
?*y* = ?*x*
?*z* = (+ ?*x* ?*y*)
```

?*q* = (create$ a b c))

После выполнения данного конструктора в CLIPS появятся 4 глобальные переменные: x, y, z и q. Переменной x присваивается целое значение 3. Переменной y – значение, сохраненное в глобальной переменной x (т.е. 3). Переменной z – сумма значений x и y (т.е. 6).

Обратите внимание, что переменная y не является указателем на переменную x, просто их значения в данный момент совпадают. Если изменить значение x, значения переменных y и z, несмотря ни на что, останутся равными 3 и 6 соответственно.

Добавьте еще один конструктор defglobal, объявляющий переменные вещественного и текстового типа, а также переменную со значением типа symbol.

(defglobal
?*d* = 7.8
?*e* = "string"
?*f* = symbol)

При выполнении команды reset все глобальные переменные получают начальные значения, определенные в конструкторе.

Команда ppdefglobal выводит в диалоговое окно системы определение заданной глобальной переменной.

Имя глобальной переменной должно быть задано без вопросительного знака и символов *, т.е. name для переменной ?*name*.

Команда list-defglobals предназначена для отображения в диалоговом окне списка имен всех определенных в системе глобальных переменных.

(list-defglobals [<имя-модуля>])

Если необязательный параметр <имя-модуля> не указан, то данная команда выводит имена глобальных переменных, определенных в текущем модуле. Если параметр содержит имя конкретного модуля, команда list-defglobal выводит список переменных, определенных в заданном модуле. Допускается использование символа *. В этом случае команда выведет в диалоговое окно имена всех глобальных переменные, определенных во всех модулях системы.

Команда show-defglobals, в отличие от команды list-defglobals, выводит в диалоговое окно CLIPS не только имена глобальных переменных, но и их значения. В остальном эти две команды практически идентичны.

(show-defglobals [<имя-модуля>])

Команда undefglobal предназначена для удаления определенных пользователем глобальных переменных.

(undefglobal <имя-глобальной-переменной>)

В качестве параметра <имя-глобальной-переменной> допускается использование символа *. В этом случае команда попытается удалить все определенные пользователем глобальные переменные. Если глобальная переменная указана, например, в определении функции, удаление этой переменной закончится неудачей.

Существуют похожие функции для стандартных операций с конструкторами различных типов (не только для defglobal).

(bind <имя-переменной> <выражение>*)

Параметр выражения является необязательным. Если он не задан, то переменной будет установлено начальное значение, заданное в конструкторе defglobal. В случае, если выражение было задано, то его значение будет вычислено и результат присвоен переменной. Если было задано несколько выражений, все они будут вычислены, из их результа-

тов будет составлено составное поле, которое будет присвоено глобальной переменной.

Функция bind возвращает значение false в случае, если переменной по какой-то причине не было присвоено никакого значения. В противном случае функция возвращает значение, присвоенное переменной.

Поскольку переменные в CLIPS слабо типизированы, типы значений, присваиваемые одной и той же переменной, в разные моменты времени могут не совпадать.

CLIPS поддерживает эвристическую и процедурную парадигму представления знаний. Для представления знаний в процедурной парадигме CLIPS предоставляет такие механизмы, как глобальные переменные, функции и родовые функции. Кроме того, существует такой способ представления знаний, как правила. Правила в CLIPS служат для представления эвристик или так называемых «эмпирических правил» действий при возникновении некоторой ситуации. Разработчик экспертной системы определяет набор правил, которые вместе работают над решением некоторой задачи. Правила состоят из предпосылок и следствия. Предпосылки называются также ЕСЛИ-частью правила, левой частью правила или LHS правила (left-hand side of rule). Следствие называется ТО-частью правила, правой частью правила или RHS правила (right-hand side of rule).

Предпосылки правила представляют собой набор условий (или условных элементов), которые должны удовлетвориться для того, чтобы правило выполнилось. Предпосылки правил удовлетворяются в зависимости от наличия или отсутствия некоторых заданных фактов в списке фактов или некоторых созданных объектов, являющихся экземплярами классов, определенных пользователем. Один из наиболее распространен-

ных типов условных выражений в CLIPS – образцы (patterns). Образцы состоят из набора ограничений, которые используются для определения того, удовлетворяет ли некоторый факт или объект условному элементу. Другими словами, образец задает некоторую маску для фактов или объектов. Процесс сопоставления образцов фактам или объектам называется процессом сопоставления образцов (pattern-matching). CLIPS предоставляет механизм, называемый механизмом логического вывода (inference engine), который автоматически сопоставляет образцы с текущим списком фактов и определенными объектами в поисках правил, которые применимы в данный момент.

Следствие правила представляется набором некоторых действий, которые необходимо выполнить в случае, если правило применимо к текущей ситуации. Таким образом, действия, заданные вследствие правила, выполняются по команде механизма логического вывода, если все предпосылки правила удовлетворены. В случае, если в данный момент применимо более одного правила, механизм логического вывода использует так называемую стратегию разрешения конфликтов (conflict resolution strategy), которая определяет, какое именно правило будет выполнено. После этого CLIPS выполняет действия, описанные вследствие выбранного правила (которые могут оказать влияние на список применимых правил), и приступает к выбору следующего правила. Этот процесс продолжается до тех пор, пока список применимых правил не опустеет.

Чтобы лучше понять сущность правил в CLIPS, их можно представить в виде оператора IF-THEN, используемого в процедурных языках программирования, например, таких как Ada или C. Однако условия выражения IF-THEN в процедурных языках вычисляются тогда, когда

поток управления программой непосредственно попадает на данное выражение путем последовательного перебора выражений и операторов, составляющих программу. В CLIPS, в отличии от этого, механизм логического вывода создает и постоянно модифицирует список правил, условия которых в данный момент удовлетворены.

Эти правила запускаются на выполнение механизмом логического вывода. С этой стороны правила похожи на обработчики сообщений, присутствующие в таких языках программирования, как, например, Ada или SmallTalk.

Для добавления новых правил в базу знаний CLIPS предоставляет специальный конструктор defrule. В общем виде синтаксис данного конструктора можно представить следующим образом:

(defrule
<имя-правила>
[<комментарии>]
[<определение-свойства-правила>]
<предпосылки> ; левая часть правила
=>
<следствие> ; правая часть правила
)

Имя правила должно быть значением типа symbol. В качестве имени правила нельзя использовать зарезервированные слова CLIPS, которые были перечислены ранее. Определение правила может содержать объявление свойств правила, которое следует непосредственно после имени правила и комментариев.

В справочной системе и документации по CLIPS для обозначения предпосылок правила чаще всего используется термин "LHS of rule", а

для обозначения следствия – "RHS of rule", поэтому в дальнейшем мы будем использовать аналогичную терминологию – левая и правая часть правила.

Левая часть правила задается набором условных элементов, который обычно состоит из условий, примененных к некоторым образцам. Заданный набор образцов используется системой для сопоставления с имеющимися фактами и объектами.

Все условия в левой части правила объединяются с помощью неявного логического оператора and. Правая часть правила содержит список действий, выполняемых при активизации правила механизмом логического вывода. Для разделения правой и левой части правил используется символ →. Правило не имеет ограничений на количество условных элементов или действий. Единственным ограничением является свободная память вашего компьютера. Действия правила выполняются последовательно, но тогда и только тогда, когда все условные элементы в левой части этого правила удовлетворены.

Если в левой части правила не указан ни один условный элемент, CLIPS автоматически подставляет условие образец initialfact или initial-object.

После того как в систему добавлены все необходимые правила и приготовлены начальные списки фактов и объектов, CLIPS готов выполнять правила. В традиционных языках программирования точка входа, точка остановки и последовательность вычислений явно определяются программистом. В CLIPS поток исполнения программы совершенно не требует ясного определения. Знания (правила) и данные (факты и объекты) разделены, и механизм логического вывода, предоставляемый CLIPS, применяет данные к знаниям, формируя список применимых правил, по-

сле чего последовательно выполняет их. Этот процесс называется основным циклом выполнения правил (basic cycle of rule execution). Рассмотрим последовательность действий (шагов), выполняемых системой CLIPS в этом цикле в момент выполнения нашей программы.

1. Если был достигнут предел выполнения правил или не был установлен текущий фокус, выполнение прерывается. В противном случае для выполнения выбирается первое правила модуля, на котором был установлен фокус. Если в текущем плане выполнения нет удовлетворенных правил, то фокус перемещается по стеку фокусов и устанавливается на следующий модуль в списке. Если стек фокусов пуст, выполнение прекращается. Иначе шаг 1 выполняется еще один раз.

2. Выполнение действий, описанных в правой части выбранного правила. Использование функции return может менять положение фокуса в стеке фокусов. Число запусков данного правила увеличивается на единицу для определения предела выполнения правила.

3. В результате выполнения шага 2 некоторые правила могут быть активированы или дезактивированы. Активированные правила (т.е. правила, условия которых удовлетворяются в данный момент) помещаются в план решения задачи модуля, в котором они определены. Размещение в плане определяется приоритетом правила (salience) и текущей стратегией разре-

шения конфликтов (эти понятия будут описаны ниже). Дезактивированные правила удаляются из текущего плана решения задачи. Если для правила установлен режим просмотра активаций, то пользователь получит соответствующее информационное сообщение при каждой активации или дезактивации правила (режим просмотра активаций можно

установить с помощью диалогового окна Watch options. Для этого выберите пункт Watch в меню Execution и установите флажок Activations).

4. Если установлен режим динамического приоритета (dynamic salience), то для всех правил из текущего плана решения задачи вычисляются новые значения приоритета. После этого цикл повторяется с шага 1.

Свойства правил позволяют задавать характеристики правил до описания левой части правила. Для задания свойства правила используется ключевое слово declare. Однако правило может иметь только одно определение свойства, заданное с помощью declare.

<определение-свойства-правила> :: = (declare <свойство-правила>)

<свойство-правила> :: = (salience <целочисленное выражение>) | (auto-focus TRUE | FALSE)

Свойство правила salience позволяет пользователю назначать приоритет для своих правил. Объявляемый приоритет должен быть выражением, имеющим целочисленное значение из диапазона от –10 000 до +10 000. Выражение, представляющее приоритет правила, может использовать глобальные переменные и функции. Однако старайтесь не указывать в этом выражении функций, имеющих побочное действие. В случае, если приоритет правила явно не задан, ему присваивается значение по умолчанию, т.е. 0.

Значение приоритета может быть вычислено в одном из трех случаев: при добавлении нового правила, при активации правила и на каждом шаге основного цикла выполнения правил. Два последних варианта называются динамическим приоритетом (dynamic salience). По умолчанию значение приоритета вычисляется только во время добавления пра-

вила. Для изменения этой установки можно использовать команду set-salience-evaluation.

Каждый метод вычисления приоритета содержит в себе предыдущий (т.е. если приоритет вычисляется на каждом шаге основного цикла выполнения правил, то он вычисляется и при активации правила, а также при его добавлении в систему).

Свойство auto-focus позволяет автоматически выполняться команде focus при каждой активации правила.

План решения задачи – это список всех правил, имеющих удовлетворенные условия при некотором, текущем состоянии списка фактов и объектов (которые еще не были выполнены). Каждый модуль имеет свой собственный план решения задачи.

Выполнение плана подобно стеку (верхнее правило плана всегда будет выполнено первым). Когда активируется новое правило, оно размещается в плане решения задачи, руководствуясь следующими факторами.

1. Только активированное правило помещается выше всех правил с меньшим приоритетом и ниже всех правил с большим приоритетом.

2. Среди правил с одинаковым приоритетом используется текущая стратегия разрешения конфликтов для определения размещения среди других правил с одинаковым приоритетом.

3. Если правило активировано вместе с несколькими другими правилами, добавлением или исключением некоторого факта и с помощью шагов 1 и 2 нельзя определить порядок правила в плане решения задачи, то правило произвольным образом упорядочивается вместе с другими правилами, которые были активированы. Заметьте, что в этом случае порядок, в котором правила были добавлены в систему, оказывает произ-

вольный эффект на разрешение конфликта (который в высшей степени зависит от текущей реализации правил).

CLIPS поддерживает семь различных стратегий разрешения конфликтов: стратегия глубины (depth strategy), стратегия ширины (breadth strategy), стратегия упрощения (simplicity strategy), стратегия усложнения (complexity strategy), LEX (LEX strategy), MEA (MEA strategy) и случайная стратегия (random strategy). По умолчанию в CLIPS установлена стратегия глубины. Текущая стратегия может быть установлена командой set-strategy (которая переупорядочит текущий план решения задачи, базируясь на новой стратегии).

• Стратегия глубины. Только что активированное правило помещается выше всех правил с таким же приоритетом. Например, допустим, что факт – А активировал правила 1 и 2 и факт Б активировал правило 3 и правило 4, тогда, если факт А добавлен перед фактом Б, в плане решения задачи правила 3 и 4 будут располагаться выше, чем правила 1 и 2. Однако позиция правила 1 относительно правила 2 и правила 3 относительно правила 4 будет произвольной.

• Стратегия ширины. Только что активированное правило помещается ниже всех правил с таким же приоритетом. Например, допустим, что факт А активировал правила 1 и 2 и факт Б активировал правила 3 и 4, тогда, если факт А добавлен перед фактом В, в плане решения задачи правила 1 и 2 будут располагаться выше, чем правила 3 и 4. Однако позиция правила 1 относительно правила 2 и правила 3 относительно правила 4 будет произвольной.

• Стратегия упрощения. Между всеми правилами с одинаковым приоритетом только что активированные правила размещаются выше всех активированных правил с равной или большей определенностью

(specificity). Определенность правила вычисляется по числу сопоставлений, которые нужно сделать в левой части правила. Каждое сопоставление с константой или заранее связанной с фактом переменной добавляет к определенности единицу. Каждый вызов функции в левой части правила, являющийся частью условных элементов : , = или test, также добавляет к определенности единицу. Логические функции and, or и not не увеличивают определенность правила, но их аргументы могут это сделать. Вызовы функций, сделанные внутри функций, не увеличивают определенность правила. Например, следующее правило имеет определенность, равную 5.

```
(defrule example
(item ?x ?y ?x)
(test (and (numberp ?x) (> ?x (+ 10 ?y)) (< ?x 100)))
=> )
```

Сравнение заранее связанной переменной >x с константой и вызовы функций numberp, < и > добавляют единицу к определенности правила. В итоге получаем определенность, равную 5. Вызовы функций and и + не увеличивают определенность правила.

• Стратегия усложнения. Между правилами с одинаковым приоритетом только что активированные правила размещаются выше всех активированных правил с равной или меньшей определенностью.

• Стратегия LEX. Между правилами с одинаковым приоритетом только что активированные правила размещаются с применением одноименной стратегии, впервые использованной в системе OPSS. Для определения места активированного правила в плане решения задачи используется "новизна" образца, который активировал правило. CLIPS маркирует каждый факт или объект временным тегом для отображения относи-

тельной новизны каждого факта или объекта в системе. Образцы, ассоциированные с каждой активацией правила, сортируются по убыванию тегов для определения местоположения правила.

Активация правила, выполненная более новыми образцами, располагается перед активацией, осуществленной более поздними образцами. Для определения порядка размещения двух активаций правил, поодиночке сравниваются отсортированные временные теги для этих двух активаций, начиная с наибольшего временного тега. Сравнение продолжается до тех пор, пока не остается одна активация с наибольшим временным тегом. Эта активация размещается выше всех остальных в плане решения задачи.

Если активация некоторого правила выполнена большим числом образцов, чем активация другого правила, и все сравниваемые временные теги одинаковы, то активация другого с большим числом временных тегов помещается перед активацией с меньшим. Если две активации имеют одинаковое количество временных тегов и их значения равны, то правило с большей определенностью помещается перед активацией с меньшей. В отличие от системы OPSS, условный элемент not в CLIPS имеет псевдовременной тег, который также используется в данной стратегии разрешения конфликтов. Временной тег условного элемента not всегда меньше, чем временной тег образца.

В качестве примера рассмотрим следующие шесть активаций правил, приведенные в LEX-порядке (запятая в конце строки активации означает наличие логического элемента not). Учтите, что временные теги фактов не обязательно равны индексу, но если индекс факта больше, то больше и его временной тег. Для данного примера примем, что временные теги равны индексам.

rule-6: f-1, f-4

rule-5: f-1, f-2, f-3,

rule-1: f-1, f-2, f-3

rule-2: f-3, f-1

rule-4: f-1, f-2,

rule-3: f-2, f-1

Далее показаны те же активации с индексами фактов в том порядке, в котором они сравниваются стратегией LEX.

rule-6: f-4, f-1

rule-5: f-3, f-2, f-1,

rule-1: f-3, f-2, f-1

rule-2: f-3, f-1

rule-4: f-2, f-1,

rule-3: f-2, f-1

• Стратегия MEA. Между правилами с одинаковым приоритетом только что активированные правила размещаются с использованием одноименной стратегии, впервые использованной в системе OPSS. Основное отличие стратегии MEA от LEX в том, что в стратегии MEA не производится сортировка образцов, активировавших правило. Сравниваются только временные теги первых образцов двух активаций. Активация с большим тегом помещается в план решения задачи перед активацией с меньшим. Если обе активации имеют одинаковые временные теги, ассоциированные с первым образцом, то для определения размещения активации в плане решения задачи используется стратегия LEX. Как и в стратегии LEX, условный элемент not имеет псевдовременной тег.

В качестве примера рассмотрим следующие шесть активаций, приведенные в MEA-порядке (запятая на конце активации означает наличие логического элемента not).

rule-2: f-3, f-1

rule-3: f-2, f-1

rule-6: f-1, f-4

rule-5: f-1, f-2, f-3,

rule-1: f-1, f-2, f-3

rule-4: f-1, f-2,

• Случайная стратегия. Каждой активации назначается случайное число, которое используется для определения местоположения среди активаций с одинаковым приоритетом. Это случайное число сохраняется при смене стратегий, таким образом, тот же порядок воспроизводится при следующей установке случайной стратегии (среди активаций в плане решения задачи, когда стратегия заменена на исходную).

Для описания функций пользователя служит следующий конструктор.

(deffunction <имя-функции>

[<комментарии>] <обязательные-параметры>

[<групповой-параметр>] <действия>)

<обязательные-параметры> ::= <выражение-простое-поле>

<групповой-параметр> : := <выражение-составное-поле>

Синтаксис конструктора deffunction включает в себя 5 элементов:

− имя функции;

− необязательные комментарии;

− список из нуля или более параметров;

− необязательный символ групповых параметров для указания того, что функция может иметь переменное число аргументов;

− последовательность действий или выражений, которые будут выполнены (вычислены) по порядку в момент вызова функции.

В зависимости от того, задан ли групповой параметр, функция, созданная конструктором, может принимать точное число параметров или число параметров не меньшее, чем некоторое заданное. Обязательные параметры определяют минимальное число аргументов, которое должно быть передано функции при ее вызове. В действиях функции можно ссылаться на каждый из этих параметров как на обычные переменные, содержащие простые значения. Если был задан групповой параметр, то функция может принимать любое количество аргументов большее или равное минимальному числу. Если групповой параметр не задан, то функция может принимать число аргументов точно равное числу обязательных параметров. Все аргументы функции, которые не соответствуют обязательным параметрам, группируются в одно значение составного поля.

Ссылаться на это значение можно, используя символ группового параметра. Для работы с групповым параметром могут использоваться стандартные функции CLIPS, предназначенные для работы с составными полями, такие как length и nth. Определение функции может содержать только один групповой параметр.

```
CLIPS>
(deffunction print-args (?a ?b $?c)
(printout t ?a " " ?b " and " (length ?c) " extras: " ?c
crlf))
CLIPS> (print-args 1 2)
1 2 and 0 extras: ()
```

```
CLIPS> (print-args a b c d)
a b and 2 extras: (c d)
CLIPS> (print-args a)
[ARGACCESS4] Function print-args expected at least 2 argument(s)
CLIPS>
```

В данном примере с помощью конструктора deffunction определяется функция print-args, которая принимает два обязательных параметра: ?a и ?b, и имеет групповой параметр $?c. Функция выводит на экран свои обязательные параметры, а также число полей в составном параметре и его содержимое.

При вызове функции интерпретатор CLIPS последовательно выполняет действия в порядке, заданном конструктором. Функция возвращает значение, равное значению, которое вернуло последнее действие или вычисленное выражение. Если последнее действие не вернуло никакого результата, то выполняемая функция также не вернет результата (как в приведенном выше примере). Если функция не выполняет никаких действий, то возвращенное значение равно FALSE. В случае возникновения ошибки при выполнении очередного действия выполнение функции будет прервано и возвращенным значением также будет FALSE.

Функции могут быть само- и взаимно рекурсивными. Саморекурсивная функция просто вызывает сама себя из списка своих собственных действий. В качестве примера можно привести функцию, вычисляющую факториал.

```
(deffuction factorial (?a)
(if (or (not (integerp ?a)) (< ?a 0)) then
(printout t "Factorial error! " crlf)
else
```

```
(if (= ?a 0) then 1 else
(* ?a (factorial (- ?a 1))))))
```

Взаимная рекурсия между двумя функциями требует предварительного объявления одной из этих функций. Для предварительного объявлении функции в CLIPS используется конструктор deffunction с пустым списком действий. В следующем примере функция foo предварительно объявлена и таким образом может быть вызвана из функции bar. Окончательная реализация функции foo выполнена конструктором после объявления функции bar.

```
(deffunction foo ())
(deffunction bar () (foo))
(deffunction foo () (bar))
```

Команда ppdeffunction выводит определение заданной функции на экран.

```
(ppdeffunction <имя-функции>)
```

Команда list-deffunctions предназначена для отображения в диалоговом окне списка имен всех определенных в системе функций.

```
(list-deffunctions)
```

Для удаления функций, определенных пользователем с помощью конструкторов deffunction, предназначена команда undeffunction.

```
(undeffunction <имя-функции>)
```

В качестве параметра <имя-функции> возможно использование символа *. В этом случае команда попытается удалить все определенные пользователем функции. Удаление функции закончится неудачей, если выбранная функция в данный момент используется или выполняется (например, правилом).

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 такие 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:

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);

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;

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.

CLIPS предоставляет возможность разбиения базы данных и решения задачи на отдельные независимые модули. Для создания таких модулей служит конструктор defmodule. С помощью модулей можно группировать вместе отдельные элементы базы знаний и управлять процессом доступа к этим элементам во время решения некоторой задачи. Подобный процесс управления доступом к данным напоминает механизмы пространства имен, используемый в C++, и глобальных и локальных областей видимости в языках C и Ada. Однако, в отличие от механизмов в перечисленных выше языках, области видимости в CLIPS строго иерархичны и однонаправлены: если модуль A может видеть данные модуля B, это не означает, что модуль B может видеть данные модуля A. С помощью управления с ограничением доступа к данным, содержащимся в различных модулях, при решении сложных задач модули могут реализовывать концепцию доски объявлений (blackboard strategy – стратегия решения задач с использованием разнородных источников знаний, взаимодейству-

ющих через общее информационное поле). В этом случае отдельный модуль позволяет видеть правилам из других модулей строго определенный набор фактов и объектов. Кроме того, модули используются для управления потоком вычисления правил.

(defmodule <имя-молуля> [<комментарий>]
<спецификации-импорта-экспорта>*)
<спецификация-импорта-экспорта> : :=
(export <элемент-спецификация>) |
(import <имя-модуля> <элемент-спецификации>)
<элемент-спецификации> : := ?ALL | ?NONE |
<конструктор> ?ALL | <конструктор> ?NONE |
<конструктор> <имя-конструктора>
<конструкция>:: = deftemplate | defclass |
defglobal | deffunction | defgeneric

После своего создания модуль не может быть переопределен или удален (за исключением системного модуля MAIN, который пользователь может один раз переопределить). Единственный способ удалить существующий модуль – выполнить команду clear. Во время запуска системы и при вызове команды clear CLIPS автоматически создает предопределенный системный модуль: (defmodule MAIN).

Явное задание модуля выполняется с помощью имени модуля, разделенного с именем конструкции при помощи двойного двоеточия :: . Имя модуля и символ :: называются спецификатором модуля (module specifier). Например, запись MAIN::find-stuff ссылается на конструкцию find-stuff из модуля MAIN.

Неявное задание модуля выполняется с помощью установки текущего активного модуля. Текущий модуль меняется при каждом определении нового модуля или при вызове функции set-current-module.

В CLIPS факты и объекты принадлежат не тому модулю, в котором они были созданы, а тому, в котором был определен соответствующий конструктор deftemplate или defclass. Таким образом, факты и объекты становятся видимыми в тех модулях, которые импортируют соответствующие конструкции deftemplate или defclass. Это позволяет разбивать базу знаний таким образом, чтобы правила или другие конструкции могли видеть только необходимые им факты и объекты. Далее приведен пример создания и взаимосвязи двух модулей.

```
CLIPS> (clear)
CLIPS> (defmodule A (export deftemplate foo bar) )
CLIPS> (deftemplate A::foo (slot x) )
CLIPS> (deftemplate A::bar (slot y) )
CLIPS> (deffacts A::info (foo (x 3) ) (bar (y 4)))
CLIPS> (defmodule B (import A deftemplate foo) )
CLIPS> (reset)
CLIPS> (facts A)
f-1 (foo (x 3) )
f-2 (bar (y 4) )
For a total of 2 facts.
CLIPS> (facts B)
f-1 (foo (x 3) )
For a total of 1 fact.
CLIPS>
```

Таким образом, имя объекта можно указать тремя способами.

<имя-объекта> ::= [<имя>] |

[::<имя>] |

[<модуль> :: <имя>]

Скобки являются обязательным синтаксисом CLIPS.

Каждый модуль имеет свой собственный процесс сопоставления образцов для своих правил и свой план решения задачи. По команде run начинает выполняться план решения задачи модуля, на который в данный момент установлен фокус. Команды reset и clear автоматически устанавливают фокус на модуль MAIN. Выполнение правил продолжается до тех пор, пока в плане решения задачи не останется применимых правил, и другой модуль не получит фокус, либо правая часть одного из выполняемых правил не вызовет функцию return. После того как в плане решения задачи модуля, имеющего фокус, заканчиваются правила, текущей модуль удаляется из стека фокусов (focus stack) и находящийся в стеке следующий модуль получает фокус. Перед выполнением правила текущим становится модуль, в котором данное правило определено. Управлять стеком фокусов можно с помощью команды focus.

В завершение следует иметь в виду, что CLIPS может неудовлетворительно работать в реальном времени, когда потребуется время реакции менее 0,1 с. В этом случае надо исследовать на разработанном прототипе механизмы вывода для всего множества правил предметной области на различных по производительности компьютерах. Как правило, современные персональные компьютеры обеспечивают работу с продукционными системами объемом 1000 – 2000 правил в реальном времени. Web-ориентированные средства на базе JAVA (системы Exsys Corvid, JESS) являются более медленными, чем, например, CLIPS 6 или OPS-2000. Поэтому

CLIPS – лучший на сегодня выбор для работы в реальном времени среди распространяемых свободно оболочек ЭС, разработанных на C++.

Задание на лабораторную работу

1. Описать словесно факты и правила для разрабатываемого прототипа, представить возможную иерархию понятий.

2. Перевести факты и правила в синтаксис языка CLIPS.

3. Продемонстрировать работоспособность прототипа на конкретных примерах.

Конкретные задания студентам предлагается выбрать самостоятельно. Примерами могут быть системы диагностикипромышленной, бытовой или теле-, аудиоаппаратуры, компьютерной техники. В качестве основы для изучения предметной области можно воспользоваться руководствами к данным устройствам.

Контрольные вопросы

1. Какие стратегии поддерживает CLIPS?
2. Для чего нужны родовые функции и обработчики сообщений?
3. Что такое модуль CLIPS и из чего он состоит?
4. Для чего устанавливается фокус в системе CLIPS?

Лабораторная работа № 5

Использование объектно-ориентированного расширения clips при создании экспертных систем

Цель работы: Научиться решать типичные задачи искусственного интеллекта. Овладеть методами объектно-ориентированного расширения CLIPS.

Краткие сведения из теории

Рассмотрим пример создания диагностической экспертной системы, которая позволяет установить причину неисправности автомобиля и выдать соответствующую рекомендацию. Разработку любой экспертной системы следует начинать с выявления основных сущностей, имеющих значение при решении конкретной задачи и законов, скорее всего эмпирических, действующих над этими сущностями.

В результате работы с экспертом были установлены следующие эмпирические правила.

1. Двигатель обычно находится в одном из 3 состояний: он может работать нормально, работать неудовлетворительно или не заводиться.
2. Если двигатель работает нормально, то это означает, что он нормально вращается, система зажигания и аккумулятор находятся в норме и никакого ремонта не требуется.
3. Если двигатель запускается, но работает ненормально, то это говорит, по крайней мере, о том, что аккумулятор в порядке.
4. Если двигатель не запускается, то нужно узнать, пытается ли он вращаться. Если двигатель вращается, но при этом не заводится, то это может говорить о наличии плохой искры в системе зажигания. Если двигатель даже не пытается заводиться, то это говорит о том, что искры нет в принципе.
5. Если двигатель не заводится, но вращается, нужно проверить наличие топлива. Если топлива нет – то, скорей всего, для ремонта машины нужно просто заправиться.
6. Если двигатель не заводится, нужно также проверить, заряжен ли аккумулятор, если нет, то его следует зарядить.
7. Если двигатель не заводится и существует вероятность плохой искры в системе зажигания, то необходимо проверить контакты. Контакты могут

быть в одном из трех состояний – чистые, опаленные и грязные, в случае опаленных контактов их необходимо заменить, в случае если контакты грязные, их достаточно просто почистить.

8. Если двигатель не заводится, искры нет и аккумулятор заряжен, то нужно проверить катушку зажигания на электрическую проводимость. В случае, если ток не проходит через катушку, то ее необходимо заменить. Если катушка зажигания в порядке, значит необходимо заменить распределительные провода.

9. Если двигатель запускается, но при этом ведет себя инертно, не сразу реагирует на подачу топлива, то необходимо прочистить топливную систему.

10. Если двигатель запускается, но происходят перебои с зажиганием, то это говорит о наличии плохой искры в системе зажигания, для устранения данной неисправности необходимо отрегулировать зазоры между контактами.

11. Если двигатель запускается и стучит, то необходимо отрегулировать зажигание.

12. Если двигатель запускается, но не развивает нормальной мощности, то это может говорить об опаленных или загрязненных контактах (см. правило 7).

13. Возможны ситуации, когда состояние двигателя нельзя описать приведенными выше факторами и машине может потребоваться более детальный анализ состояния.

Из приведенных выше правил можно выделить следующие сущности, имеющие значение при решении задачи.

− Во-первых, для решения задачи экспертной системе необходимо знать, в каком состоянии находится машина, диагностика которой производит-

ся. Эксперт выделил три возможных состояния: нормальная работа двигателя, двигатель работает неудовлетворительно, не заводится (см. правило 1).

− Во-вторых, большинство приведенных правил, помимо состояния двигателя в целом, используют понятие состояния вращения двигателя. Согласно этим правилам двигатель может находиться в одном из двух состояний, которые определяются в зависимости от того, способен он вращаться (работать) или нет.

− В-третьих, в некоторых правилах (см. правила 4, 7, 8, 10) используется понятие состояния системы зажигания. Система зажигания может быть в одном из трех состояний: нормальное состояние, нерегулярная работа и нерабочее состояние.

− В-четвертых, в правилах 6 и 8 используется понятие состояния аккумулятора. Аккумулятор может быть в одном из двух состояний: заряженным и разряженным.

Таким образом, можно выделить следующие факты, описывающие состояние автомобиля и его узлов:

- Группа фактов, описывающая состояние машины:

working-state engine normal – нормальная работа;

working-state engine unsatisfactory – неудовлетворительная работа;

working-state engine does-not-start – не заводится.

- Группа фактов, описывающая состояние двигателя:

rotation-state engine rotates – двигатель вращается;

rotation-state engines does-not-rotate – двигатель не вращается.

- Группа фактов, описывающая состояние системы зажигания:

spark-state engine normal – зажигание в порядке;

spark-state engine irregular-spark – искра не регулярна;

spark-state engine does-not-spark – искры нет.

- Группа фактов, описывающая состояние системы питания:

charge-state battery charged – аккумулятор заряжен;

charge-state battery dead – аккумулятор разряжен.

Обратите внимание, что факты, входящие в одну группу (содержат одинаковое первое поле), являются взаимоисключающими, т.е. наличие в системе сразу двух фактов из одной группы лишено смысла.

Из постановки задачи следует, что наша экспертная система должна предоставлять пользователю рекомендации, позволяющие устранить найденную неисправность. Из приведенных выше правил можно выделить следующие рекомендации: добавить топливо (правило 5); зарядить аккумулятор (правило 6); заменить или почистить контакты (правило 7 или 12); заменить катушку зажигания или распределительные провода (правило 8); прочистить топливную систему (правило 9); отрегулировать зазоры между контактами (правило 10); отрегулировать зажигание (правило 11). Необходимо помнить также о двух крайних случаях: ремонт не требуется в принципе; экспертная система не смогла поставить диагноз.

Таким образом, получатся следующие рекомендации:

− repair "Add gas";

− repair "Charge the battery";

− repair "Replace the points";

− repair "Clean the points";

− repair "Replace the ignition coil";

− repair "Repair the distributor lead wire";

− repair "Clean the fuel line";

− repair "Point gap adjustment";

− repair "Timing adjustment";

− repair "No repair needed";

− repair "Take your car to a mechanic".

Обратите также внимание, что одни и те же рекомендации могут выводиться как правилом 7, так и правилом 12. Однако состояние машины при этой поломке отличается. Для того, чтобы иметь возможность обрабатывать эту ситуацию с помощью одного правила CLIPS, введем еще два дополнительных факта:

symptom engine low-output – низкая мощность;

symptom engine not-low-output – нормальная мощность.

Как упоминалось выше, для работы нашей системы можно заставить пользователя вручную вводить факты, описывающие проявление возникшей неисправности. Однако такой метод имеет ряд серьезных недостатков: пользователь может забыть о каких-нибудь существенных деталях или, наоборот, указать слишком много информации, что может помешать нормальной работе системы. Кроме того, факты, описывающие проявление неисправности, должны иметь строго определенный формат, и система не сможет их обработать в случае ошибки со стороны пользователя.

В данной экспертной системе мы реализуем правила диагностики, которые в зависимости от той или иной ситуации будут задавать пользователю необходимые вопросы и получать ответ в строго заданной форме. Дальнейшая диагностика будет производиться с учетом предыдущих ответов на вопросы, заданные пользователю. Эти ответы будут формировать описание текущей ситуации с помощью фактов, приведенных выше. Для реализации подобной архитектуры будет необходимо реализовать функцию, задающую пользователю произвольный вопрос и получающую

ответ из заданного набора корректных ответов. Далее приведена одна из возможных реализаций такой функции.

```
(deffunction ask-question (?question $?allowed-values)
(printout t ?question)
(bind ?answer (read))
(if (lexemep ?answer)
then (bind ?answer (lowcase ?answer)))
(while (not (member ?answer ?allowed-values)) do
(printout t ?question)
(bind ?answer (read))
(if (lexemep ?answer)
then (bind ?answer (lowcase ?answer))))
?answer)
```

Функция принимает два аргумента: простую переменную question, которая содержит текст вопроса, и составную переменную allowed-values с набором допустимых ответов. Сразу после своего вызова функция выводит на экран соответствующий вопрос и читает ответ пользователя в переменную answer. Если переменная answer содержит текст, то она будет принудительно приведена к прописному алфавиту. После этого функция проверяет, является ли полученный ответ одним из заданных корректных ответов. Если нет, то процесс повторится до получения корректного ответа, иначе функция вернет ответ, введенный пользователем.

Будет также очень полезно определить функцию, задающую пользователю вопрос и допускающую ответ в виде да/нет, так как это один из самых распространенных типов вопросов.

```
(deffunction yes-or-no-p (?question)
(bind ?response (ask-question ?question yes no y n))
```

```
(if (or (eq ?response yes) (eq ?response y))
then TRUE
else FALSE))
```

Функция yes-or-no-p вызывает функцию ask-question с постоянным набором допустимых ответов: yes, no, y и n. В случае, если пользователь ввел ответ yes или y, функция возвращает значение TRUE, иначе FALSE. Обратите внимание, что поскольку функция yes-or-no-p использует функцию ask-question, то она должна быть определена после нее.

Для упрощения реализации нашей экспертной системы введем следующее ограничение: за один запуск система может предоставить пользователю только одну рекомендацию по исправлению неисправности. В случае, если в машине несколько неисправностей, то систему нужно будет последовательно вызывать несколько раз, удаляя обнаруженную на каждом новом шаге неисправность. Таким образом, одним из образцов всех диагностических правил будет (not (repair ?)), гарантирующий, что диагноз еще не поставлен.

Первым реализуем правило, определяющее общее состояние двигателя (см. правило 1).

```
(defrule determine-engine-state ""
(not (working-state engine ?))
(not (repair ?))
=>
(if (yes-or-no-p "Does the engine start (yes/no)? ")
then
(if (yes-or-no-p "Does the engine run normally (yes/no)? ")
then (assert (working-state engine normal))
else (assert (working-state engine unsatisfactory)))
```

```
else
(assert (working-state engine does-not-start))))
```

Условный элемент (not (working-state engine ?)) гарантирует, что общее состояние двигателя еще не определено. Если это так, то пользователю задаются соответствующие вопросы и в систему добавляется факт, описывающий текущее общее состояние двигателя.

Теперь реализуем правило, определяющее, пытается ли двигатель вращаться в случае, если он не заводится.

```
(defrule determine-rotation-state ""
(working-state engine does-not-start)
(not (rotation-state engine ?))
(not (repair ?))
=>
(if (yes-or-no-p "Does the engine rotate (yes/no)? ")
then
(assert (rotation-state engine rotates))
(assert (spark-state engine irregular-spark))
else
(assert (rotation-state engine does-not-rotate))
(assert (spark-state engine does-not-spark))))
```

Это правило выполняется в случае, если общее состояние двигателя определено и известно, что он не заводится. Кроме того, условный элемент (not (rotation-state engine ?)) гарантирует, что это правило еще не вызывалось. В зависимости от того или иного ответа пользователя правило добавляет соответствующий набор фактов (см. правило 4).

Далее реализуем довольно простые правила 5 и 6. Выполняемые ими действия вы поймете без дополнительных комментариев.

```
(defrule determine-gas-level ""
(working-state engine does-not-start)
(rotation-state engine rotates)
(not (repair ?))
=>
(if (not (yes-or-no-p "Does the tank have any gas in it (yes/no)? "))
then
(assert (repair "Add gas.")))))
(defrule determine-battery-state ""
(rotation-state engine does-not-rotate)
(not (charge-state battery ?))
(not (repair ?))
=>
(if (yes-or-no-p "Is the battery charged (yes/no)? ")
then
(assert (charge-state battery charged))
else
(assert (repair "Charge the battery."))
(assert (charge-state battery dead))))
```

Обратите внимание, что правило determine-battery-state, помимо определения возможной неисправности, также применяется для добавления в систему факта, описывающего текущее состояние аккумулятора, который может быть использован другими правилами.

При реализации правила 7 необходимо обратить внимание на то, что рекомендации, предоставляемые этим правилом, подходят для двух в корне отличающихся ситуаций. Во-первых, в случае, если двигатель не заводится и существует вероятность плохой искры в системе зажигания

(правило 7). Во-вторых, в случае, если двигатель запускается, но не развивает нормальной мощности (правило 12). Поэтому выполним реализацию этих правил так, как представлено ниже.

```
(defrule determine-low-output ""
(working-state engine unsatisfactory)
(not (symptom engine low-output | not-low-output))
(not (repair ?))
=>
(if (yes-or-no-p "Is the output of the engine low (yes/no)? ")
then
(assert (symptom engine low-output))
else
(assert (symptom engine not-low-output))))
(defrule determine-point-surface-state ""
(or (and (working-state engine does-not-start)
(spark-state engine irregular-spark))
(symptom engine low-output))
(not (repair ?))
=>
(bind ?response
(ask-question "What is the surface state of the points (normal/burned/contaminated)? "
normal burned contaminated))
(if (eq ?response burned)
then
(assert (repair "Replace the points."))
else (if (eq ?response contaminated)
```

then (assert (repair "Clean the points.")))))

Правило determine-low-output определяет, имеет ли место низкая мощность двигателя или нет. Правило determine-pointsurface-state адекватно реагирует на условия, заданные в правилах 7 и 12. Обратите внимание на использование условных элементов or и end, которые обеспечивают одинаковое поведение правила в двух абсолютно разных ситуациях. Кроме того, правило determine-point-surface-state отличается от приведенных ранее тем, что непосредственно использует функцию askquestion вместо yes-or-no-p, так как в данный момент пользователю задается вопрос, подразумевающий три варианта ответа.

Реализация оставшихся диагностических правил (8 – 11) также не должна вызвать у вас затруднений.

```
(defrule determine-conductivity-test ""
(working-state engine does-not-start)
(spark-state engine does-not-spark)
(charge-state battery charged)
(not (repair ?))
=>
(if (yes-or-no-p "Is the conductivity test for the ignition coil positive (yes/no)?
")
then
(assert (repair "Repair the distributor lead wire."))
else
(assert (repair "Replace the ignition coil.")))))
(defrule determine-sluggishness ""
(working-state engine unsatisfactory)
(not (repair ?))
```

```
=>
(if (yes-or-no-p "Is the engine sluggish (yes/no)? ")
then (assert (repair "Clean the fuel line."))))
(defrule determine-misfiring ""
(working-state engine unsatisfactory)
(not (repair ?))
=>
(if (yes-or-no-p "Does the engine misfire (yes/no)? ")
then
(assert (repair "Point gap adjustment."))
(assert (spark-state engine irregular-spark))))
(defrule determine-knocking ""
(working-state engine unsatisfactory)
(not (repair ?))
=>
(if (yes-or-no-p "Does the engine knock (yes/no)? ")
then
(assert (repair "Timing adjustment."))))
```

Внимательно взглянув на список правил, мы увидим, что некоторые правила (2, 3 и 13) остались до сих пор нереализованными. Реализация правила 13 будет следующая.

```
(defrule no-repairs ""
(declare (salience –10))
(not (repair ?))
=>
(assert (repair "Take your car to a mechanic.")))
```

Обратите внимание на использование приоритета при определении этого правила. Все правила, приведенные в предыдущем разделе, определялись с приоритетом, по умолчанию равным нулю. Использование для правила no-repairs приоритета, равного –10, гарантирует, что правило не будет выполнено, пока в плане решения задачи находится, по крайней мере, одно из диагностических правил. Если все активированные диагностические правила опрошены и ни одно из них не смогло подобрать подходящую рекомендацию по устранению неисправности, то система запустит данное правило, рекомендующее пользователю обратиться к механику.

Реализация правил 2 и 3 приведена ниже.

```
(defrule normal-engine-state-conclusions ""
(declare (salience 10))
(working-state engine normal)
=>
(assert (repair "No repair needed."))
(assert (spark-state engine normal))
(assert (charge-state battery charged))
(assert (rotation-state engine rotates)))
(defrule unsatisfactory-engine-state-conclusions ""
(declare (salience 10))
(working-state engine unsatisfactory)
=>
(assert (charge-state battery charged))
(assert (rotation-state engine rotates)))
```

В этих правилах, наоборот, используется более высокий приоритет, что гарантирует их выполнение до запуска любого диагностирующего

правила (в случае выполнения соответствующих условий). Это избавит нашу систему от лишних проверок, а пользователя от лишних вопросов.

Экспертная система фактически готова к работе. Единственное, чего ей не хватает – это метода вывода итоговой информации и правила, сообщающего пользователю о начале работы системы. Ниже приведена реализация этих правил.

```
(defrule system-banner ""
(declare (salience 10))
=>
(printout t crlf crlf)
(printout t "The Engine Diagnosis Expert System")
(printout t crlf crlf))
(defrule print-repair ""
(declare (salience 10))
(repair ?item)
=>
(printout t crlf crlf)
(printout t "Suggested Repair:")
(printout t crlf crlf)
(format t " %s%n%n%n" ?item))
```

Теперь для того, чтобы запустить экспертную систему, достаточно выполнить команду reset, которая добавит факт initial-fact, необходимый для правила system-banner, и команду run. После этого вы сразу увидите сообщение "The Engine Diagnosis Expert System", которое означает, что система начала работать, и получите серию вопросов, ответы на которые помогут экспертной системе оценить состояние вашей машины и подобрать соответствующую рекомендацию по ремонту.

Задание на лабораторную работу

1. Для выбранной задачи искусственного интеллекта описать разрабатываемые классы и их иерархию.

2. Разработать и отладить методы данных классов.

3. Продемонстрировать работоспособность экспертной системы при поиске конечного решения из различных начальных состояний фактов.

В качестве задания можно использовать классические задачи искусственного интеллекта типа: задача фермера, обезьяны и бананы, каннибалы и миссионеры. Также могут быть выбраны в качестве заданий логические задачи математики по определенному упорядочиванию и др. Задание согласовывается с преподавателем.

Контрольные вопросы

1. В чем суть объектно-ориентированного подхода CLIPS?
2. Что такое объект CLIPS и из чего он состоит?
3. Что является фактов в системе CLIPS?

Литература

1. Загорулько, Ю. А. Инженерия знаний : учебное пособие / Ю. А. Загорулько, Г. Б. Загорулько. — Новосибирск : Новосибирский государственный университет, 2016. — 93 с. — ISBN 978-5-4437-0452-4. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/93454.html (дата обращения: 24.07.2023). — Режим доступа: для авторизир. пользователей.

2. Исаев, С. В. Интеллектуальные системы : учебное пособие / С. В. Исаев, О. С. Исаева. — Красноярск : Сибирский федеральный университет, 2017. — 120 с. — ISBN 978-5-7638-3781-0. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/84365.html (дата обращения: 16.01.2024). — Режим доступа: для авторизир. пользователей.

3. Представление знаний в информационных системах : учебное пособие / Ю. Ю. Громов, О. Г. Иванова, М. Ю. Серегин [и др.]. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2012. — 169 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/64163.html (дата обращения: 16.01.2024). — Режим доступа: для авторизир. пользователей.

4. Цильковский, И. А. Методы анализа знаний и данных : конспект лекций / И. А. Цильковский, В. М. Волкова. — Новосибирск : Новосибирский государственный технический университет, 2010. — 68 с. — ISBN 978-57782-1377-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/45385.html (дата обращения: 16.01.2024). — Режим доступа: для авторизир. пользователей.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
НЕВИННОМЫССКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)

ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В СИСТЕМАХ УПРАВЛЕНИЯ

Методические указания к практическим занятиям
для студентов направления подготовки
15.03.04 «Автоматизация технологических процессов и производств»

Невинномысск 2024

Методические указания предназначены для студентов направления подготовки 15.03.04 «Автоматизация технологических процессов и производств». Они содержат основы теории, порядок проведения практических занятий, перечень контрольных вопросов для самоподготовки и список рекомендуемой литературы.

Методические указания разработаны в соответствии с требованиями федерального государственного образовательного стандарта в части содержания и уровня подготовки выпускников по направлению подготовки 15.03.04 «Автоматизация технологических процессов и производств»

Составитель: канд. техн. наук А.А. Евдокимов

Ответственный редактор: канд. техн. наук Д.В. Болдырев

1 Семантические сети

Цель работы: получение знаний, умений и навыков по созданию базы знаний, представляющей собой семантическую сеть.

Последовательность выполнения работы

1. Изучите раздел «Семантические сети» теоретического материала.
2. Решить задачи.
3. Ответьте на контрольные вопросы.
    a. Что такое семантическая сеть?
    b. Какие отношения обычно используются в семантических сетях?
    c. Что такое бинарная семантическая сеть?
    d. Какими достоинствами и недостатками обладает семантическая сеть?
4. Напишите отчет к работе.

Задачи

Задача 1

Компания, в которой Вы работаете, получила задание на разработку справочной системы по журналам издательства «Издательство Мечты». Данная компания выпускает различные по целевой аудитории, ценовой категории и объему страниц журналы.

Вам необходимо построить модуль на основе семантической сети, позволяющий определить целевую аудиторию для различных журналов, а также для кого предназначено издание и его стоимость. Ваша задача построить семантическую сеть на основе информации, представленной в таблице 1.

Таблица 1

Название	**Основ-**	**Стои-**	**Объем стра-**	**Какая**	**Возмож-**

журнала	ная целевая аудитория	мость одного номера, руб.	ниц журнала	информация представлена в журнале	на ли подписка на журнал
Тюниг автомобилей	Мужчины	140	170	Современные технологии тюнинга автомобилей	нет
Мода	Женщины	90	90	Новейшие тенденции моды	да
Компьютерные и видео игры	Мужчины и женщины	65	60	Все о компьютерных и видео играх	нет
Рукоделие	Женщины	45	50	Эксклюзивные вещи своими руками	да
Фотография	Мужчины и женщины	100	95	Основы и секреты фотографии	да
Кино и музыка	Мужчины и женщины	30	30	Только актуальная информация и кино и музыке	нет

В построенной семантической сети определить:

1. Какой журнал предоставляет информацию о современных технологии тюнинга автомобилей?
2. Какие журналы предназначены для мужчин?
3. Какие журналы стоят 100 рублей?
4. На какие журналы можно оформить подписку?

Задача 2

Ваша задача состоит в создании экспертной системы АСУ предприятия, автоматизирующей контроль за выполнением задач коллективом предприятия. АСУ следует построить в виде семантической сети.

Система должна описывать структуру предприятия, в том числе руководство и структуру отделов.

Система так же должна описывать выполняемые предприятием задания, в том числе:

1. Наименование задания.
2. Сроки его выполнения.
3. Этапы выполнения задания и их очередность.

Для каждого этапа описывается:

1. Отдел, выполняющий этап.
2. Ответственное лицо, обычно – руководитель отдела или подразделения.
3. Сроки начала и окончания этапа.

Предприятие, для которого строится система – ООО «Созвездие»:

Директор: Иванов И.И.

Отдел разработки, нач. отдела – Перов П.П.

В составе отдела разработки:

Бюро постановки задач, нач. бюро – Сидоров С.С.

Бюро программирования, нач. бюро – Брайан Керниган

Бюро сопровождения, нач. бюро – Билл Гейтс

Отдел маркетинга, нач. отдела – Тошико Ямада

Задания в работе:

1. Разработка текстового редактора «Созвездие», этапы – постановка задачи, программирование, продвижение на рынок, поддержка.
2. Разработка Интернет - браузера «Созвездие», этапы – постановка задачи, программирование, продвижение на рынок, поддержка.

В построенной семантической сети определить:

1. Кто является начальником отдела маркетинга?

2. Какие задания выполняет ООО «Созвездие»?
3. Чем занимается Иванов И.И.?
4. Какие сроки выполнения заданы для разработки Интернет - браузера?

2 Продукционная модель представления знаний

Цель работы: получение знаний, умений и навыков по созданию базы знаний, представляющей собой продукционную модель представления знаний.

Последовательность выполнения работы

1. Изучите раздел «Продукционная модель представления знаний» теоретического материала.
2. Решить задачи.
3. Ответьте на контрольные вопросы.
    a. Как представлены знания в продукционной модели представления знаний?
    b. Что такое консеквент?
    c. Какие части имеет продукционная система?
    d. Для чего нужна рабочая память?
    e. Какими достоинствами и недостатками обладает продукционная модель представления знаний?
4. Напишите отчет к работе.

Задачи

Задача 1

На рынке фотоаппаратов существует огромное множество различных фотокамер, способных удовлетворить почти любого потребителя. Выбор камеры - непростое дело, и не всякий покупатель способен сам выбрать

себе подходящий фотоаппарат. В фирменных салонах по продаже фотокамер не хватает консультантов, которые могли бы квалифицированно помочь клиенту. Ваша задача - разработать продукционную систему, помогающую покупателю в его непростом выборе.

Таким образом, задание на лабораторную работу состоит в наполнении пустой оболочки экспертной системы знаниями из таблицы 2.

Большинство покупателей склонны пере- или недооценивать свои запросы (и способности), поэтому вовсе необязательно прямо спрашивать у человека, любитель он или профессионал. Такие вещи лучше выяснять, спросив у человека, например, собирается ли он покупать аппарат впервые (новичок), снимает для семейного альбома (любитель), печатается ли он в каких-нибудь изданиях (профессионал).

Новичок редко может сказать, нужна ли ему сменная оптика в камере. Необходимо узнать, собирается ли он снимать портреты и пейзажи – в этом случае она ему, возможно, понадобится, или только фотографии вечеринок и.т.д. (в этом случае сменная оптика не нужна).

Мало кто так же способен внятно сказать, нужны ли ему художественные режимы (если он не занимается фотографией профессионально). Это лучше всего выяснить, спросив, например, что такое глубина резкости (предложите клиенту несколько вариантов на выбор, правильный ответ – диапазон расстояний, в котором все объекты выходят резкими).

Цена определяется исходя из максимальной суммы, которую готов потратить клиент.

Новые правила следует внести в часть программного кода, отвечающего за правила, то есть в часть, которая начинается со служебных слов:

topic camera.

set_number_of_values (camera,1).

Пример правила со служебным синтаксисом приведен ниже. В данном правиле класс пользователя - новичок, сменная оптика не нужна, художественные режимы отсутствуют, и низкая цена, этим характеристикам отвечает камера Riva Zoom 100. Этому правилу соответствует первая строка таблицы 2.

if ?class is new and ?opt is no and ?hud_res is no and ?price is low

then camera is 'Riva Zoom 100'.

Таблица 2

Класс	Сменная оптика	Художественные режимы	Цена	Камера
Новичок	нет	нет	Небольшая	Riva Zoom100
Новичок	нет	нет	Средняя	Riva Zoom500
Новичок	нет	нет	Высокая	Riva Zoom550
Новичок	есть	нет	Небольшая	Dynax 404si
Новичок	есть	нет	Средняя	Dynax 406si
Новичок	есть	нет	Высокая	Dynax 408si
Новичок	нет	есть	Небольшая	Nikon 1200
Новичок	нет	есть	Средняя	Kodak 835AF
Новичок	нет	есть	Высокая	Nikon 1201
Новичок	есть	есть	Небольшая	Rekam DH300
Новичок	есть	есть	Средняя	Rekam Max 3
Новичок	есть	есть	Высокая	Rekam Mega 110
Любитель	нет	нет	Небольшая	Konica POP MINI
Любитель	нет	нет	Средняя	Konica POP EFP-8
Любитель	нет	нет	Высокая	Konica Centuria-10
Любитель	нет	есть	Небольшая	Konica POP BF-8
Любитель	нет	есть	Средняя	Konica Centuria-20
Любитель	нет	есть	Высокая	Konica POP ST
Любитель	есть	нет	Небольшая	Konica POP ST 80
Любитель	есть	нет	Средняя	**Samsung FINO 15**

				SE
Любитель	есть	нет	Высокая	Samsung FINO 20 SE
Любитель	есть	есть	Неболь-шая	Dynax 505si
Любитель	есть	есть	Средняя	Samsung VEGA 170
Любитель	есть	есть	Высокая	**Pentax ESPIO 200**
Профессионал	нет	нет	Неболь-шая	нет
Профессионал	нет	нет	Средняя	нет
Профессионал	нет	нет	Высокая	нет
Профессионал	нет	есть	Неболь-шая	нет
Профессионал	нет	есть	Средняя	нет
Профессионал	нет	есть	Высокая	нет
Профессионал	есть	нет	Неболь-шая	Konica Centuria-60Z
Профессионал	есть	нет	Средняя	Konica Centuria-70Z
Профессионал	есть	нет	Высокая	Olympus ZOOM 80
Профессионал	есть	есть	Неболь-шая	Olympus is-300
Профессионал	есть	есть	Средняя	Dynax 7
Профессионал	есть	есть	Высокая	Dynax 9

Задача 2

Вы работаете в крупной компании производящей спецодежду. Ваша компания производит более 10 000 наименований различной спецодежды. Отдел программирования, в котором Вы работаете, получил задание на разработку системы, помогающей покупателю определиться с выбором. Покупатель может выбрать защищающую одежду для головы, рук или корпуса. Материал, из которого изготовлена одежда, может быть как натуральным, так и искусственным. Также в наличии имеются товары различной ценовой категории. Ваша задача: наполнить пустую оболочку экспертной системы знаниями из таблицы 3. Цена определяется исходя из максимальной суммы, которую готов потратить клиент.

Таблица 3

Защи-та	Материал	Серия	Цена	наименование

голова	натуральный	«Зима»	Неболь-шая	Каска «Труд»
голова	искусственный	«Омон»	Средняя	Каска «Байкал»
голова	искусственный	«Зима»	Высокая	Каска «СОМ 3-53 Люкс
голова	искусственный	«Рабочий»	Неболь-шая	Каска «СуперБосс»
голова	натуральный	«Комфорт»	Средняя	Каска «Эйрвинг»
голова	искусственный	«Зима»	Высокая	Каска «Вигард»
голова	натуральный	«Зима»	Неболь-шая	Каска «Металлург»
голова	искусственный	«Омон»	Средняя	Каска «Сварщик»
голова	натуральный	«Омон»	Высокая	Каска «Шахтер»
голова	натуральный	«Комфорт»	Неболь-шая	Каска «Термолюкс»
голова	искусственный	«Рабочий»	Средняя	Каска «Супер Вигард»
голова	искусственный	«Рабочий»	Высокая	Каска «Люкс»
руки	натуральный	«Зима»	Неболь-шая	Рукавицы утеплен-ные гладкокрашеные
руки	искусственный	«Зима»	Средняя	Рукавицы нагольные
руки	натуральный	«Зима»	Высокая	Рукавицы меховые
руки	натуральный	«Омон»	Неболь-шая	Рукавицы крытые
руки	натуральный	«Омон»	Средняя	Рукавицы морозо-стойкие
руки	искусственный	«Рабочий»	Высокая	Рукавицы с двойным налодонником
руки	искусственный	«Рабочий»	Неболь-шая	Рукавицы джинсовые
руки	искусственный	«Зима»	Средняя	Рукавицы брезенто-вые
руки	натуральный	«Комфорт»	Высокая	Рукавицы крагиспил-ковые
руки	искусственный	«Рабочий»	Неболь-шая	Рукавицы суконные
руки	искусственный	«Комфорт»	Средняя	Вачеги для метал-лурга
руки	искусственный	«Комфорт»	Высокая	Вачеги цельноспил-ковые
корпус	искусственный	«Омон»	Неболь-шая	Костюм мужской «Вектор»
корпус	искусственный	«Омон»	Средняя	Костюм мужской «Вектор+»

корпус	искусственный	«Омон»	Высокая	Костюм мужской «Амулет»
корпус	искусственный	«Зима»	Небольшая	Костюм мужской «Диксон»
корпус	искусственный	«Зима»	Средняя	Костюм мужской «Зимник»
корпус	искусственный	«Рабочий»	Высокая	Костюм мужской «Модуль»
корпус	натуральный	«Комфорт»	Небольшая	Костюм мужской «Стим»
корпус	натуральный	«Комфорт»	Средняя	Костюм мужской «Рейсер»
корпус	натуральный	«Зима»	Высокая	Костюм мужской «Тайшет»
корпус	натуральный	«Зима»	Небольшая	Костюм мужской «Буран КМФ»
корпус	натуральный	«Комфорт»	Средняя	Костюм мужской «Аляска»
корпус	натуральный	«Рабочий»	Высокая	Костюм мужской «Легенда»

3 Фреймовая модель представления знаний

Цель работы: получение знаний, умений и навыков по созданию базы знаний, представляющей собой фреймовую модель представления знаний.

Последовательность выполнения работы

1. Изучите раздел «Фреймовая модель представления знаний» теоретического материала.
2. Решите задачи.
3. Ответьте на контрольные вопросы.
    a. Что такое фрейм, слот?
    b. Что такое механизм наследования? Для чего он нужен?
    c. Какие указатели наследования Вы знаете?
    d. Какие свойства фреймов Вы знаете?
    e. Что такое демон?

f. Какими достоинствами и недостатками обладают фреймы?

4. Напишите отчет к работе.

Задачи

Задача 1

Вы – офицер штаба армии. Вам поступило задание разработать экспертную систему, предназначенную для выбора десантного средства для тактических операций.

В наличии имеются следующие боевые транспортные аппараты:

Таблица 4

Наименование	Способ передвижения	Вместимость, десант	Вооружение	Защита
БМП-3	Гусеницы	7 чел	пушка 100 мм	броня
Ми-24	Вертолет	6 чел	пушка 30 мм	сталь
БТР-50Б	Колеса	8 чел	пулемет 14.5 мм	броня
Ми-8	Вертолет	12 чел	нет	нет

Необходимо создать систему, занимающую наименьший объем памяти, позволяющую выбирать боевые средства тактической доставки десанта на основе заданных требований к ним.

Таким образом, задачей выполнения лабораторной работы является создание системы фреймов из знаний, представленных в таблице 4.

В созданной системе найти десантные средства, удовлетворяющие следующим требованиям:

1. вертолет, способный перевозить не менее 6 человек десанта над полем боя без поддержки наземных средств;
2. наземное средство доставки не менее 6 десантников в тыл противника при активном противодействии бронетанковых войск противника;

3. средство доставки не менее 6 человек десанта в тыл противника при наличии противодействия противника.

Задача 2

В задаче №1 «Семантические сети», вариант 1 представлена задача разработки семантической сети для небольшого издательства. Ваша задача представить знания, описанные в таблице 1 в виде фреймов.

В построенной системе фреймов определить:

1. Какой журнал предоставляет информацию о современных технологиях тюнинга автомобилей?
2. Какие журналы предназначены для мужчин?
3. Какие журналы стоят 100 рублей?
4. На какие журналы можно оформить подписку?
5. Найти все журналы издательства.
6. Найти все журналы, которые стоят не более 150 рублей.

4 Модель, основанная на нечеткой логике

Цель работы: получение знаний, умений и навыков по созданию базы нечетких знаний.

Последовательность выполнения работы

1. Изучите раздел «Модель, основанная на нечетких знаниях» теоретического материала.
2. Решить задачи.
3. Ответьте на контрольные вопросы.
    a. Что такое лингвистическая переменная, нечеткое множество?
    b. Какое нечеткое число называется триангулярным, трапецеидальным, в чем их различие?

c. Какие операции над нечеткими множествами Вы знаете?

d. Как работает нечеткий логический контроллер?

4. Напишите отчет к работе.

Задачи

Задача 1

Компания, в которой Вы работаете, получила задание на разработку нечеткого управляющего контроллера для бытового кондиционера. Контроллер должен управлять работой главного вентилятора.

На вход контроллера поступают сигналы от датчиков влажности и температуры. На основе показаний этих датчиков Вам необходимо построить набор правил и нечеткие множества, определяющие лексические термы для этих правил.

Вам необходимо так же выбрать метод импликации и дефаззификации полученного в результате работы системы нечеткого множества.

Задача 2

Вы сотрудник бюро автоматизации коммерческой поликлиники. Вам необходимо разработать нечеткий логический контроллер для выбора подходящего рациона питания. В качестве входных характеристик выступают рост человека и его вес, в качестве выходных параметров – калорийность диеты.

Вам необходимо так же выбрать метод импликации и дефаззификации полученного в результате работы системы нечеткого множества.

Методические указания к практическим занятиям

На практических занятиях № 1, 3, 5 и 7 предусмотрены работы по разделам дисциплины «Интеллектуализация систем управления химико-

технологическими процессами и производствами». Практические работы выполняются студентами на занятиях в письменном виде, сдаются преподавателю. За практическую работу выставляется оценка, которая в дальнейшем влияет на экзаменационную оценку.

Практическое занятие №1. «*Модели представления знаний*». Для выполнения работы №1 необходимо изучить теоретический материал.

Практическое занятие №2. «*Методы технологии инженерии знаний»*.

На данном занятии проводится семинар по методам извлечения знаний. Методы извлечения знаний делятся на текстологические и коммуникативные. Семинар посвящен коммуникативным методам извлечения знаний. Студенты делятся на группы по 4-5 человек. Группа студентов выбирает любой из коммуникативных методов извлечения знаний. Далее происходит распределение ролей в группе. Например, если выбран метод «Круглого стола», то четверо из пяти студентов становятся экспертами, один инженером по знаниям. По легенде эксперты обладают знаниями о предметной области, а задача инженера по знаниям эти знания получить и запротоколировать. Задача занятия состоит в детальном и практическом изучении методов инженерии знаний. Примеры систем, для которых предстоит наполнять базу знаний, приведены ниже. При подготовке к семинарским занятиям студентам потребуется дополнительная подготовка, такая как продумывание плана занятия, подготовка табличек (например, «доктор» «инженер по знаниям» «пациент»), легенды семинара.

Практическое занятие №3. *«Экспертные системы. Технологии инженерии знаний»*. Для выполнения работы №2 необходимо изучить теоретический материал.

Практическое занятие №4. *Чтение докладов по теме «Нейронные сети»*.

Практическое занятие №5. «*Эволюционные алгоритмы*». Для выполнения работы №3 необходимо изучить теоретический материал.

Практическое занятие №6. *«Специальные языки представления знаний»*. Тема обсуждения: логико-лингвистические ЯПЗ и функциональные семантические сети.

Практическое занятие №7. «*Специальные языки представления знаний*». Для выполнения работы №4 необходимо изучить теоретический материал.

Практическое занятие №8. «*Выводы на знаниях»*. Тема обсуждения: выводы в СПЗ, реализованных в виде семантических сетей и сетей фреймов.

Практическое занятие №9. «*Дедуктивные и абдуктивные выводы».* Тема обсуждения: отличительные черты дедуктивных абдуктивных рассуждений.

Практическое занятие №10. «*Языки логического программирования».* Темы обсуждения: отличительные черты традиционного и логического программирования.

Практическое занятие №11. «*Введение в язык Пролог. Примеры Пролог – программ».* Темы обсуждения: основные разделы Пролог - программы. Предикаты, правила-продукции, факты и данные в Пролог - программе.

Структура отчета

1. Титульный лист.

2. Задание. Задание оформляется в соответствии с выбранным заданием.

3. Содержание. Содержание включает наименование всех разделов, подразделов и пунктов, если они имеют наименование, а также список литературы и приложений, с указанием номера страниц, с которых они начинаются.

4. Введение. Введение должно содержать вводную информацию. То есть цель работы, краткое содержание теоретического материала по данной теме.

5. Основная часть работы должна содержать этапы выполнения работы и результаты выполнения контрольных тестов (результаты ответа системы на вопросы, приведенные в заданиях к работам).

6. Заключение. Заключение должно содержать основные выводы, сделанные студентом, по выполнению работы.

7. Список используемой литературы. В список литературы должны быть включены все источники, которые были использованы при выполнении работы.

8. Приложения. Приложения содержат вспомогательный материал, такой как базы знаний в виде одной из моделей представления знаний.

Список литературы

1. Загорулько, Ю. А. Инженерия знаний : учебное пособие / Ю. А. Загорулько, Г. Б. Загорулько. — Новосибирск : Новосибирский государственный университет, 2016. — 93 с. — ISBN 978-5-4437-0452-4. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/93454.html (дата обращения: 24.07.2023). — Режим доступа: для авторизир. пользователей.

2. Исаев, С. В. Интеллектуальные системы : учебное пособие / С. В. Исаев, О. С. Исаева. — Красноярск : Сибирский федеральный университет, 2017. — 120 с. — ISBN 978-5-7638-3781-0. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/84365.html (дата обращения: 16.01.2024). — Режим доступа: для авторизир. пользователей.

3. Представление знаний в информационных системах : учебное пособие / Ю. Ю. Громов, О. Г. Иванова, М. Ю. Серегин [и др.]. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2012. — 169 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/64163.html (дата обращения: 16.01.2024). — Режим доступа: для авторизир. пользователей.

4. Цильковский, И. А. Методы анализа знаний и данных : конспект лекций / И. А. Цильковский, В. М. Волкова. — Новосибирск : Новосибирский государственный технический университет, 2010. — 68 с. — ISBN 978-57782-1377-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/45385.html (дата обращения: 16.01.2024). — Режим доступа: для авторизир. пользователей.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
НЕВИННОМЫССКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ
(ФИЛИАЛ)

Методические указания к самостоятельной работе
для студентов направления
15.03.04 «Автоматизация технологических процессов и производств»
по дисциплине

«ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В СИСТЕМАХ УПРАВЛЕНИЯ»

Невинномысск, 2023

Методические указания разработаны в соответствии с требованиями ФГОС ВО в части содержания и уровня подготовки выпускников направления подготовки 15.03.04 «Автоматизация технологических процессов и производств».

Методические указания содержат рекомендации по организации самостоятельной работы студента при изучении дисциплины «Представление знаний в системах управления».

Составитель доцент кафедры ИСЭА Д.В. Болдырев

Ответственный редактор доцент кафедры ИСЭА А.А. Евдокимов

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дисциплина «Представление знаний в системах управления» ставит своей целью формирование следующих компетенций будущего бакалавра по направлению подготовки 15.03.04 — Автоматизация технологических процессов и производств.

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
УК-2. Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений.	ИД-3$_{УК-2}$ Обеспечивает выполнение проекта в соответствии с установленными целями, сроками и затратами, исходя из действующих правовых норм, имеющихся ресурсов и ограничений, в том числе с использованием цифровых инструментов.	Определяет параметры средств и систем автоматизации, обеспечивающие оптимальность проектных решений
ПК-2. Способен участвовать в работах по расчету и проектированию средств и систем автоматизации с использованием современных информационных технологий, методов и средств проектирования.	ИД-1$_{ПК-2}$ Рассчитывает и проектирует средства и системы автоматизации в соответствии с техническим заданием.	Рассчитывает и проектирует средства и системы автоматизации с использованием технологии искусственного интеллекта.
	ИД-3$_{ПК-2}$ Выполняет сбор и анализ исходных данных для расчета и проектирования средств и систем управления с использованием современных информационных технологий.	Выполняет сбор и анализ исходных данных для расчета и проектирования средств и систем управления с использованием технологии искусственного интеллекта.

Главными задачами дисциплины являются: усвоение студентами основных положений теории искусственного интеллекта, способов представления знаний и их инженерии, правил анализа и синтеза экспертных систем.

В результате освоения дисциплины студент должен:

- знать основные положения теории искусственного интеллекта; способы организации интеллектуальных систем; способы построения баз данных, баз знаний и экспертных систем; модели и методы формализации и представления знаний в интеллектуальных системах;
- уметь использовать интеллектуальные информационные системы, инструментальные средства управления базами данных и знаний;
- владеть навыками решения задач с помощью систем искусственно интеллекта и экспертных систем.

Методические указания предназначены для выполнения самостоятельной работы по дисциплине «Представление знаний в системах управления» с учетом требований ФГОС ВО для направления подготовки 15.03.04 — Автоматизация технологических процессов и производств. Они способствуют лучшему усвоению студентами теоретических положений и обеспечивает приобретение практических навыков по исследованию элементов и систем автоматического регулирования и управления.

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ ПРИ ИЗУЧЕНИИ ДИСЦИПЛИНЫ «ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В СИСТЕМАХ УПРАВЛЕНИЯ»

Самостоятельная работа студентов (далее — СРС) является неотъемлемой составляющей образовательного процесса в Университете и является обязательной для каждого студента. Основная цель СРС — освоение в полном объеме образовательной программы и последовательное формирование компетенций эффективной самостоятельной профессиональной (практической и научно-теоретической) деятельности. Самостоятельная работа кон-

кретна по своей предметной направленности и сопровождается непрерывным контролем и оценкой ее результатов.

Количество часов, отводимое на самостоятельную работу, определяется учебным планом направления подготовки 15.03.04.

Содержательно самостоятельная работа студентов определяется ФГОС ВО направления подготовки 15.03.04, программой и учебно-методическим комплексом дисциплины «Представление знаний в системах управления».

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей дисциплины «Представление знаний в системах управления», объема часов на ее изучение, вида заданий для СРС, индивидуальных возможностей студентов и условий учебной деятельности.

Формы самостоятельной работы студентов определяются содержанием дисциплины «Представление знаний в системах управления», степенью подготовленности студентов. Они могут быть тесно связаны с теоретическим курсом и иметь учебный или учебно-исследовательский характер. Форму самостоятельной работы студентов определяют кафедра ИСЭА при разработке программы дисциплины «Представление знаний в системах управления».

Самостоятельная работа может осуществляться индивидуально или группами студентов в зависимости от цели, объема, конкретной тематики самостоятельной работы, уровня сложности, уровня умений студентов.

СРС, не предусмотренная образовательной программой, учебным планом и учебно-методическими материалами, раскрывающими и конкретизирующими их содержание, осуществляется студентами инициативно, с целью реализации собственных учебных и научных интересов.

В учебном процессе выделяют аудиторную и внеаудиторную самостоятельную работу.

Аудиторная самостоятельная работа по дисциплине «Представление знаний в системах управления» выполняется на учебных занятиях (лекциях,

практических, лабораторных занятиях и консультациях) под руководством преподавателя и по его заданию.

Внеаудиторная самостоятельная работа студентов выполняется во внеаудиторное время по заданию и при методическом руководстве и контроле преподавателя, но без его непосредственного участия. СРС включает в себя:

- подготовку к аудиторным занятиям (лекционным и практическим) и выполнение соответствующих заданий;
- работу над отдельными темами учебных дисциплин (модулей) в соответствии с учебно-тематическими планами;
- выполнение контрольных работ;
- подготовку ко всем видам промежуточных и итоговых контрольных испытаний.

1.1. Подготовка к лекциям

Главное в период подготовки к лекционным занятиям — научиться методам самостоятельного умственного труда, сознательно развивать свои творческие способности и овладевать навыками творческой работы. Для этого необходимо строго соблюдать дисциплину учебы и поведения. Четкое планирование своего рабочего времени и отдыха является необходимым условием для успешной самостоятельной работы. В основу его нужно положить рабочие программы изучаемых в семестре дисциплин.

Каждому студенту следует составлять еженедельный и семестровый планы работы, а также план на каждый рабочий день. С вечера всегда надо распределять работу на завтрашний день. В конце каждого дня целесообразно подводить итог работы: тщательно проверить, все ли выполнено по намеченному плану, не было ли каких-либо отступлений, а если были, по какой причине это произошло. Нужно осуществлять самоконтроль, который является необходимым условием успешной учебы. Если что-то осталось невы-

полненным, необходимо изыскать время для завершения этой части работы, не уменьшая объема недельного плана.

Слушание и запись лекций — сложный вид вузовской аудиторной работы. Внимательное слушание и конспектирование лекций предполагает интенсивную умственную деятельность студента. Краткие записи лекций, их конспектирование помогает усвоить учебный материал. Конспект является полезным тогда, когда записано самое существенное, основное и сделано это самим студентом. Не надо стремиться записать дословно всю лекцию. Такое «конспектирование» приносит больше вреда, чем пользы. Запись лекций рекомендуется вести по возможности собственными формулировками. Желательно запись осуществлять на одной странице, а следующую оставлять для проработки учебного материала самостоятельно в домашних условиях.

Конспект лекций лучше подразделять на пункты, параграфы, соблюдая красную строку. Этому в большой степени будут способствовать пункты плана лекции, предложенные преподавателям. Принципиальные места, определения, формулы и другое следует сопровождать замечаниями «важно», «особо важно», «хорошо запомнить» и т.п. Можно делать это и с помощью разноцветных маркеров или ручек. Лучше если они будут собственными, чтобы не приходилось просить их у однокурсников и тем самым не отвлекать их во время лекции. Целесообразно разработать собственную «маркографию» (значки, символы), сокращения слов. Не лишним будет и изучение основ стенографии. Работая над конспектом лекций, всегда необходимо использовать не только учебник, но и ту литературу, которую дополнительно рекомендовал лектор. Именно такая серьезная, кропотливая работа с лекционным материалом позволит глубоко овладеть знаниями.

1.2. Подготовка к лабораторным занятиям

Для того чтобы лабораторные занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по

рассмотренному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться студентом на лабораторных занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

1.3. Подготовка к практическим занятиям

Подготовку к каждому практическому занятию студент должен начать с ознакомления с методическими указаниями, которые включают содержание

работы. Тщательное продумывание и изучение вопросов основывается на проработке текущего материала лекции, а затем изучения обязательной и дополнительной литературы, рекомендованную к данной теме. На основе индивидуальных предпочтений студенту необходимо самостоятельно выбрать тему доклада по проблеме и по возможности подготовить по нему презентацию.

Если программой дисциплины предусмотрено выполнение практического задания, то его необходимо выполнить с учетом предложенной инструкции (устно или письменно). Все новые понятия по изучаемой теме необходимо выучить наизусть и внести в глоссарий, который целесообразно вести с самого начала изучения курса. Результат такой работы должен проявиться в способности студента свободно ответить на теоретические вопросы семинара, его выступлении и участии в коллективном обсуждении вопросов изучаемой темы, правильном выполнении практических заданий и контрольных работ.

В зависимости от содержания и количества отведенного времени на изучение каждой темы практическое занятие может состоять из четырех-пяти частей:

1. Обсуждение теоретических вопросов, определенных программой дисциплины.

2. Доклад и/или выступление с презентациями по выбранной проблеме.

3. Обсуждение выступлений по теме — дискуссия.

4. Выполнение практического задания с последующим разбором полученных результатов или обсуждение практического задания.

5. Подведение итогов занятия.

Первая часть — обсуждение теоретических вопросов — проводится в виде фронтальной беседы со всей группой и включает выборочную проверку преподавателем теоретических знаний студентов. Примерная продолжительность — до 15 минут. Вторая часть — выступление студентов с докладами, которые должны сопровождаться презентациями с целью усиления наглядно-

сти восприятия, по одному из вопросов практического занятия. Обязательный элемент доклада — представление и анализ статистических данных, обоснование социальных последствий любого экономического факта, явления или процесса. Примерная продолжительность — 20-25 минут. После докладов следует их обсуждение — дискуссия. В ходе этого этапа практического занятия могут быть заданы уточняющие вопросы к докладчикам. Примерная продолжительность — до 15-20 минут. Если программой предусмотрено выполнение практического задания в рамках конкретной темы, то преподавателями определяется его содержание и дается время на его выполнение, а замет идет обсуждение результатов. Подведением итогов заканчивается практическое занятие.

В процессе подготовки к практическим занятиям, студентам необходимо обратить особое внимание на самостоятельное изучение рекомендованной учебно-методической (а также научной и популярной) литературы. Самостоятельная работа с учебниками, учебными пособиями, научной, справочной и популярной литературой, материалами периодических изданий и Интернета, статистическими данными является наиболее эффективным методом получения знаний, позволяет значительно активизировать процесс овладения информацией, способствует более глубокому усвоению изучаемого материала, формирует у студентов свое отношение к конкретной проблеме. Более глубокому раскрытию вопросов способствует знакомство с дополнительной литературой, рекомендованной преподавателем по каждой теме семинарского или практического занятия, что позволяет студентам проявить свою индивидуальность в рамках выступления на данных занятиях, выявить широкий спектр мнений по изучаемой проблеме.

1.4. Самостоятельное изучение материала тем

Конспект — наиболее совершенная и наиболее сложная форма записи. Слово «конспект» происходит от латинского «conspectus», что означает «об-

зор, изложение». В правильно составленном конспекте обычно выделено самое основное в изучаемом тексте, сосредоточено внимание на наиболее существенном, в кратких и четких формулировках обобщены важные теоретические положения.

Конспект представляет собой относительно подробное, последовательное изложение содержания прочитанного. На первых порах целесообразно в записях ближе держаться тексту, прибегая зачастую к прямому цитированию автора. В дальнейшем, по мере выработки навыков конспектирования, записи будут носить более свободный и сжатый характер.

Конспект книги обычно ведется в тетради. В самом начале конспекта указывается фамилия автора, полное название произведения, издательство, год и место издания. При цитировании обязательная ссылка на страницу книги. Если цитата взята из собрания сочинений, то необходимо указать соответствующий том. Следует помнить, что четкая ссылка на источник — непременное правило конспектирования. Если конспектируется статья, то указывается, где и когда она была напечатана.

Конспект подразделяется на части в соответствии с заранее продуманным планом. Пункты плана записываются в тексте или на полях конспекта. Писать его рекомендуется четко и разборчиво, так как небрежная запись с течением времени становиться малопонятной для ее автора. Существует правило: конспект, составленный для себя, должен быть по возможности написан так, чтобы его легко прочитал и кто-либо другой.

Формы конспекта могут быть разными и зависят от его целевого назначения (изучение материала в целом или под определенным углом зрения, подготовка к докладу, выступлению на занятии и т.д.), а также от характера произведения (монография, статья, документ и т.п.). Если речь идет просто об изложении содержания работы, текст конспекта может быть сплошным, с выделением особо важных положений подчеркиванием или различными значками.

В случае, когда не ограничиваются переложением содержания, а фиксируют в конспекте и свои собственные суждения по данному вопросу или дополняют конспект соответствующими материалами их других источников, следует отводить место для такого рода записей. Рекомендуется разделить страницы тетради пополам по вертикали и в левой части вести конспект произведения, а в правой свои дополнительные записи, совмещая их по содержанию.

Конспектирование в большей мере, чем другие виды записей, помогает вырабатывать навыки правильного изложения в письменной форме важные теоретических и практических вопросов, умение четко их формулировать и ясно излагать своими словами.

Таким образом, составление конспекта требует вдумчивой работы, затраты времени и труда. Зато во время конспектирования приобретаются знания, создается фонд записей.

Конспект может быть текстуальным или тематическим. В текстуальном конспекте сохраняется логика и структура изучаемого произведения, а запись ведется в соответствии с расположением материала в книге. За основу тематического конспекта берется не план произведения, а содержание какой-либо темы или проблемы.

Текстуальный конспект желательно начинать после того, как вся книга прочитана и продумана, но это, к сожалению, не всегда возможно. В первую очередь необходимо составить план произведения письменно или мысленно, поскольку в соответствии с этим планом строится дальнейшая работа. Конспект включает в себя тезисы, которые составляют его основу. Но, в отличие от тезисов, конспект содержит краткую запись не только выводов, но и доказательств, вплоть до фактического материала. Иначе говоря, конспект — это расширенные тезисы, дополненные рассуждениями и доказательствами, мыслями и соображениями составителя записи.

Как правило, конспект включает в себя и выписки, но в него могут войти отдельные места, цитируемые дословно, а также факты, примеры, цифры,

таблицы и схемы, взятые из книги. Следует помнить, что работа над конспектом только тогда будет творческой, когда она не ограничена текстом изучаемого произведения. Нужно дополнять конспект данными из других источников.

В конспекте необходимо выделять отдельные места текста в зависимости от их значимости. Можно пользоваться различными способами: подчеркиваниями, вопросительными и восклицательными знаками, репликами, краткими оценками, писать на полях своих конспектов слова: «важно», «очень важно», «верно», «характерно».

В конспект могут помещаться диаграммы, схемы, таблицы, которые придадут ему наглядность.

Составлению тематического конспекта предшествует тщательное изучение всей литературы, подобранной для раскрытия данной темы. Бывает, что какая-либо тема рассматривается в нескольких главах или в разных местах книги. А в конспекте весь материал, относящийся к теме, будет сосредоточен в одном месте. В плане конспекта рекомендуется делать пометки, к каким источникам (вплоть до страницы) придется обратиться для раскрытия вопросов. Тематический конспект составляется обычно для того, чтобы глубже изучить определенный вопрос, подготовиться к докладу, лекции или выступлению на семинарском занятии. Такой конспект по содержанию приближается к реферату, докладу по избранной теме, особенно если включает и собственный вклад в изучение проблемы.

2. СРЕДСТВА ОЦЕНИВАНИЯ УРОВНЯ СФОРМИРОВАННОСТИ КОМПЕТЕНЦИЙ ОБУЧАЮЩИХСЯ ПРИ ИЗУЧЕНИИ ДИСЦИПЛИНЫ «ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В СИСТЕМАХ УПРАВЛЕНИЯ»

Вопросы для собеседования

5 семестр

1. Дайте определение искусственному интеллекту.
2. Какие три направления сложились при моделировании искусственного интеллекта?
3. Дайте определение фактуальному знанию.
4. Дайте определение операционному знанию?
5. Опишите структуру системы, основанной на обработке баз данных.
6. Приведите структуру системы, основанной на обработке знаний.
7. Перечислите признаки ИИС.
8. Перечислите классы ИИС
9. Чем отличаются интеллектуальные базы знаний от традиционных?
10. Для решения каких задач используется естественно-языковой интерфейс?
11. Что является основным элементом экспертной системы?
12. Какие существуют требования к адаптивности ИС?
13. Что такое морфологический анализ?
14. Что такое синтаксический анализ?
15. Что такое семантический анализ?
16. Для чего используется естественно-языковой интерфейс?
17. Для чего предназначены гипертекстовые системы?
18. Что такое системы когнитивной графики?
19. Что такое данные и знания, в чем их отличие?

20. Перечислите этапы трансформации данных.
21. Перечислите этапы трансформации знаний.
22. Дайте определение интенсионала и экстенсионала понятия.
23. Что такое поверхностные знания, глубинные знания?
24. Перечислите основные модели представления знаний.
25. Что такое продукционная модель?
26. Что такое семантическая сеть?
27. Какие три основных типа отношений используются в семантических сетях?
28. Какие бывают семантические сети?
29. Какие отношения могут использоваться в семантических сетях?
30. Дайте определение фрейма.
31. Перечислите виды фреймов.
32. Как можно представить фрейм через список свойств?
33. Как можно представить фрейм в виде таблицы?
34. Дайте понятие формальной логической теории.
35. Перечислите основные компоненты системы продукций.
36. Что такое рабочая память?
37. Что такое продукционное правило?
38. Какова функция системы управления?
39. Опишите работу основного алгоритма системы продукций.
40. Что такое конфликтное множество?
41. В чем состоит проблема представления для систем продукций?
42. Опишите работу прямой и обратной системы продукций.
43. Дайте определение поля знаний.
44. На какой стадии разработки ЭС формируется поле знаний?
45. Что включает в себя семиотика.
46. Приведите синтаксическую структуру поля знаний.
47. В чем состоит процесс формирования поля знаний?
48. В чем состоит процесс извлечения знаний?

49. В чем состоит процесс приобретения знаний?
50. В чем состоит процесс формирования знаний (machine learning)?
51. Каковы три основных аспекта извлечения знаний?
52. Какой из этих аспектов является ведущим?
53. В чем состоит проблема «общего кода»?
54. В чем состоит процесс формирования понятийной структуры?
55. Какой вид имеет гносеологическая цепочка?
56. В чем суть коммуникативных методов извлечения знаний?
57. В чем суть текстологических методов извлечения знаний?
58. Как можно классифицировать предметные области по степени документированности?
59. Как можно классифицировать предметные области по степени структурированности?
60. Что означает, что предметная область хорошо структурирована?
61. Опишите простейший метод структурирования.
62. Что такое феномен восприятия?
63. Приведите постановку задачи классификации.
64. Как можно охарактеризовать способность к обучению?
65. Приведите математическую постановку задачи обучения.
66. При решении каких задач применяются методы классификации?
67. Какие методы используются для решения задачи классификации?
68. Как строятся дискриминантные функции?
69. Перечислите области применений деревьев решений.
70. Какими достоинствами и недостатками обладают деревья решений?
71. При решении каких практических задач могут применяться методы деревьев решений?
72. Опишите структуру деревьев решений.

6 семестр

1. Дайте определение экспертных систем.
2. Что является основным компонентом экспертной системы?
3. В каких областях применяются экспертные системы?
4. Приведите обобщенную структуру экспертной системы.
5. Опишите работу экспертной системы.
6. Для чего предназначена подсистема объяснений?
7. Для чего предназначен интеллектуальный редактор базы знаний?
8. Что такое база знаний?
9. Приведите классификацию экспертных систем по задаче. Охарактеризуйте каждую из задач.
10. Приведите классификацию экспертных систем по связям с реальным временем.
11. Приведите классификацию экспертных систем по типу ЭВМ.
12. Приведите классификацию экспертных систем по степени интеграции.
13. Перечислите участников коллектива разработчиков экспертной системы.
14. Какие требования предъявляются к пользователю?
15. Каковы функции эксперта в процессе разработки ЭС?
16. Какими качествами должен обладать инженер по знаниям?
17. С какими формами знаний имеет дело инженер по знаниям в процессе разработки ЭС?
18. Перечислите этапы процесса разработки ЭС.
19. Какие задачи решают разработчики на этапе выбора подходящей проблемы?
20. Что такое прототипная система?
21. Перечислите стадии разработки прототипа и охарактеризуйте каждый из них.
22. Опишите процесс перехода от прототипной системы к промышленной ЭС.

23. На основании каких критериев можно проводить оценку системы?
24. В чем суть иерархического подхода при проектировании сложных систем?
25. Как происходит движение между уровнями иерархии в нисходящей концепции?
26. Как происходит движение между уровнями иерархии в восходящей концепции?
27. Перечислите классы подходов к проектированию сложных систем.
28. Опишите объектно-структурный подход к формированию поля знаний.
29. Дайте понятие обучающей выборки.
30. Дайте понятие обучающей выборки «с учителем».
31. Дайте понятие обучающей выборки «без учителя».
32. Как осуществляется процесс классификации примеров обучающей выборки в индуктивных системах?
33. Перечислите основные проблемы, решаемые нейронными сетями.
34. Что такое полносвязная нейронная сеть?
35. Что такое слабосвязная нейронная сеть?
36. Что такое многослойная нейронная сеть?
37. Что такое нейронная сеть прямого распространения?
38. Приведите математическую модель нейрона.
39. Перечислите виды функций активации нейрона.
40. Приведите архитектуру многослойной сети прямого распространения.
41. Запишите формулы, в соответствии с которыми происходит функционирование многослойной сети прямого распространения.
42. Нарисуйте график пороговой функции активации.
43. Нарисуйте график экспоненциальной сигмоиды.
44. Что такое обучающий пример?
45. Приведите примеры обучающих примеров для различных практиче-

ских задач.

46. Что значит обучить многослойную сеть прямого распространения?
47. В чем суть алгоритма обратного распространения ошибки?
48. Чем отличаются обучение на всем множестве примеров, одиночное предъявление примеров и постраничное обучение?
49. Приведите формулировку теоремы о полноте.

Критерии оценивания компетенций

Оценка «**зачтено**» выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, правильно применяет теоретические положения при решении практических вопросов и задач, владеет необходимыми навыками и приемами их выполнения. Допускаются некоторые неточности, недостаточно правильные формулировки в изложении программного материала, затруднения при выполнении практических работ.

Оценка «**не зачтено**» выставляется студенту, если он не знает значительной части программного материала, допускает существенные ошибки, неуверенно, с большими затруднениями выполняет практические задания.

3. ОТЧЕТНОСТЬ ПО ДИСЦИПЛИНЕ

В рамках рейтинговой системы успеваемость студентов по дисциплине оценивается в ходе текущего контроля успеваемости и промежуточной аттестации.

Максимально возможный балл за весь текущий контроль устанавливается равным 55. Текущее контрольное мероприятие считается сданным, если студент получил за него не менее 60% от установленного для этого контроля максимального балла. Рейтинговый балл, выставляемый студенту за текущее контрольное мероприятие, сданное студентом в установленные графиком контрольных мероприятий сроки, определяется следующим образом:

Уровень выполнения контрольного задания	Рейтинговый балл (в % от максимального балла за контрольное задание)
Отличный	100
Хороший	80
Удовлетворительный	60
Неудвлетворительный	0

Промежуточная аттестация

Процедура зачета как отдельное контрольное мероприятие не проводится, оценивание знаний обучающегося происходит по результатам текущего контроля.

Для студентов заочной формы обучения рейтинговая оценка знаний не предусмотрена

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Перечень основной литературы:

1. Исаев, С. В. Интеллектуальные системы : учебное пособие / С. В. Исаев, О. С. Исаева. — Красноярск : Сибирский федеральный университет, 2017. — 120 с. — ISBN 978-5-7638-3781-0. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/84365.html (дата обращения: 16.01.2024). — Режим доступа: для авторизир. пользователей.

2. Представление знаний в информационных системах : учебное пособие / Ю. Ю. Громов, О. Г. Иванова, М. Ю. Серегин [и др.]. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2012. — 169 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/64163.html (дата обращения: 16.01.2024). — Режим доступа: для авторизир. пользователей.

Перечень дополнительной литературы:

1. Загорулько, Ю. А. Инженерия знаний : учебное пособие / Ю. А. Загорулько, Г. Б. Загорулько. — Новосибирск : Новосибирский государственный университет, 2016. — 93 с. — ISBN 978-5-4437-0452-4. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/93454.html (дата обращения: 24.07.2023). — Режим доступа: для авторизир. пользователей.

2. Цильковский, И. А. Методы анализа знаний и данных : конспект лекций / И. А. Цильковский, В. М. Волкова. — Новосибирск : Новосибирский государственный технический университет, 2010. — 68 с. — ISBN 978-57782-1377-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/45385.html (дата обращения: 16.01.2024). — Режим доступа: для авторизир. пользователей.