МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по выполнению практических работ по дисциплине
«Интеллектуальные системы управления»

для направления подготовки 15.04.02 Технологические машины и оборудование
Направленность (профиль) Проектирование технологического оборудования

Невинномысск 2024

Содержание

Введение 4
1 Практическое занятие № 1. Классификация знаний. Исследование предметной области 6
2 Практическое занятие № 2. Выявление знаний в системах искусственного интеллекта. Нечеткая логика. Формирование функций принадлежности в программной среде Fuzzy Logic Toolbox 14
3 Практическое занятие № 3. Построение моделей в системах искусственного интеллекта (декларативный язык Prolog) 20
4 Практическое занятие № 4. Продукции в системах искусственного интеллекта 33
5 Практическое занятие № 5. Фреймовые модели представления знаний 39
6 Практическое занятие № 6. Нейронные сети в системах искусственного интеллекта. Аппроксимация функций нейронной сети 43
Список использованных источников 55

Условия и порядок выполнения работы:

1 Изучить методические рекомендации по выполнению практических занятий.

2 Ответить на перечень вопросов, необходимых для выполнения заданий.

3 Изучить содержание заданий и приступить к выполнению.

4 Консультацию по выполнению работы получить у преподавателя.

5 Работа оценивается в целом, по итогам выполнения работы выставляется оценка.

Защита проводится путем индивидуальной беседы или выполнения зачетного задания.

Процесс выполнения практических заданий направлен на формирование компетенции ПК-1 – Способен осуществлять проведение работ по обработке и анализу научно-технической информации и результатов исследований

1 Практическое занятие №1. Классификация знаний. Исследование предметной области

Цель занятия: изучить заданную предметную область и построить модель знаний в виде графа.

Краткие теоретические сведения

Основная цель представления знаний в интеллектуальной информационной системе состоит в организации ее в такой форме, чтобы был возможен легкий доступ к знаниям для эффективного принятия решений, распознавания объектов и ситуаций, вывода заключений и прочих когнитивных функций.

Семантическая сеть в самом общем случае представляет собой ориентированный граф, вершинами которого являются обобщенные понятия, события, свойства объектов или действия. Вершины графа соединяются дугой, если соответствующие информационные единицы находятся в каком-либо взаимодействии. Дуги, таким образом, характеризуют отношения, связи между объектами, явлениями.

В основе семантических моделей представления знаний лежит конструкция, названная семантической сетью. Семантический подход применяется в системах распознавания естественного языка, в различных предметно-ориентированных исследовательских системах, в вопросно-ответных системах.

Вообще, термин «семантическая» означает «смысловая», ведь семантика – наука, которая устанавливает отношения между объектами и обозначающими их символами, т.е. определяющая смысловое значение знаков [1].

В семантической сети дуги можно определить различными методами, выбор конкретного метода зависит от вида представляемых знаний. Часто дуги, используемые для представления иерархии, включают дуги типа «множество», «подмножество», «элемент». Если семантическая сеть применяется в качестве модели представления знаний при описании естественных языков (переводчики), используют дуги типа «объект», «агент», или «реципиент».

Понятиями в семантической сети могут выступать объекты (абстрактные или конкретные), а отношениями – связи наподобие «это» («is»), «имеет частью» («has part»), «принадлежит». Чтобы сеть являлась семантической, обязательно присутствие трех типов отношений: «свойство – значение», «класс – элемент класса», «пример элемента класса».

Основные характеристики семантических сетей как моделей представления знаний:

– объекты описаны на естественном языке;

– все знания накоплены в относительно однородной структуре памяти;

– на сетях определены унифицированные отношения между объектами, им соответствуют унифицированные методы вывода;

– методы вывода в соответствии с запросами определяют участки семантического знания, имеющего отношение к поставленной задаче, формулируя акт понимания запроса и некоторую цепь выводов, соответствующих решению задачи.

Имеются различные классификации семантических сетей.

По количеству типов отношений:

– однородные (в сети имеется единственный тип отношений);

– неоднородные (в сети имеются различные типы отношений).

По типам отношений:

– бинарные (в этом случае отношениями связаны два объекта);

– парные (есть специальные отношения, которые связывают более двух понятий).

В семантических сетях чаще всего используются следующие отношения:

– связи «часть – целое» («элемент – множество», «класс – подкласс», и т.п.);

– количественные («меньше», «больше», «равно» и т.п.);

– пространственные («близко от», «далеко от», «над», «за», «под» и т.п.);

– функциональные связи (обычно определяемые глаголами «производит», «влияет» и т.п.);

– временные («раньше», «позже», «в течение» и т.п.);

– атрибутивные связи («иметь значение», «иметь свойство» и т.п.);

– логические связи («и», «или», «не») и др.

При решении задач в семантических сетях проблема поиска решения в базе знаний сводится к задаче нахождения фрагмента сети, соответствующего некоторой подсети, которая относится к поставленному вопросу [2].

Методические указания к работе

Для построения модели представления знаний в виде графа необходимо последовательно выполнить следующие шаги:

1 Определить целевые действия задачи (являющиеся решениями).

2 Определить промежуточные действия или цепочку действий, между начальным состоянием и конечным (между тем, что имеется, и целевым действием).

3 Опередить условия для каждого действия, при котором его целесообразно и возможно выполнить.

4 Определить порядок выполнения действий.

5 Добавить конкретные факты, исходя из сформулированной задачи.

6 Преобразовать полученный порядок действий и соответствующие им факты, условия и действия.

7 Для проверки правильности построения записать цепочки, явно проследив связи между ними. Подобный набор шагов предполагает движение при построении модели от результата к начальному состоянию, но возможно и движение от начального состояния к результату (шаги 1 и 2).

8 Присвоить обозначения фактам Ф, правилам П, действиям Д.

9 Построить граф предметной области.

Пример

Построить модель представления знаний в предметной области «Ресторан» (посещение ресторана).

Для построения модели представления знаний необходимо выполнить следующие шаги:

1 Определить целевые действия задачи (являющиеся решениями).

2 Определить промежуточные действия или цепочку действий, между начальным состоянием и конечным (между тем, что имеется, и целевым действием).

3 Опередить условия для каждого действия, при котором его целесообразно и возможно выполнить. Определить общий порядок выполнения действий.

4 Добавить конкретики при необходимости, исходя из поставленной задачи.

5 Преобразовать полученный порядок действий и соответствующие им условия в продукции.

6 Для проверки правильности построения продукций записать цепочки продукций, явно проследив связи между ними.

Этот набор шагов предполагает движение при построении продукционной модели от результата к начальному состоянию, но возможно и движение от начального состояния к результату (шаги 1 и 2).

Рассмотрим решение поставленной задачи по всем перечисленным пунктам.

1 Обязательное действие, выполняемое в ресторанах – поглощение пищи и ее оплата. Значит, есть уже два целевых действия «съесть пищу» и «оплатить», которые взаимосвязаны и следуют друг за другом.

2 Прежде чем что-либо съесть в ресторане, туда нужно прийти, дождаться официанта и сделать заказ. Кроме того, нужно выбрать, в какой именно ресторан пойти. Значит, цепочка промежуточных действий: «выбор ресторана и путь туда», «сделать заказ официанту».

3 Прежде чем идти в ресторан, необходимо убедиться, что есть необходимая сумма денег. Выбор ресторана может обуславливаться многими причинами, выберем территориальный признак – к какому ближе в тот и идем. В разных ресторанах работают разные люди, поэтому в зависимости от выбора ресторана, официанты будут разные.

Кроме того, разные рестораны специализируются на разных кухнях, поэтому заказанные блюда будут в разных ресторанах отличаться. Значит вначале идут действия, позволяющие выбрать ресторан, затем характеризующие рестораны, а уже после заказ, еда и оплата заказа.

4 Пусть в задаче будут рассматриваться два ресторана: «Вкусная еда» и «Вкуснятина». Первый – паб и заказы приносят быстрее, чем во втором, второй – пиццерия. В первом работает официант Сергей, а во втором официантка Марина. Петр – это клиент.

5 Выше описанное можно преобразовать в следующие предложения типа «Если, то»:

Если субъект хочет есть и у субъекта есть достаточная сумма денег, то субъект может пойти в ресторан.

Если субъект ближе к ресторану «Вкусная еда», чем к ресторану «Вкуснятина», и субъект может пойти в ресторан, то субъект идет в ресторан «Вкусная еда».

Если субъект ближе к ресторану «Вкуснятина», чем к ресторану «Вкусная еда», и субъект может пойти в ресторан, то субъект идет в ресторан «Вкуснятина».

Если субъект идет в ресторан «Вкуснятина», и в ресторане «Вкуснятина» работает официант Марина, то у субъекта принимает заказ Марина.

Если субъект идет в ресторан «Вкусная еда», и в ресторане «Вкусная еда» работает официант Сергей, то у субъекта принимает заказ Сергей.

Если субъект выбрал блюда, и у субъекта принимает заказ Марина, то заказ принесут через 20 мин.

Если субъект выбрал блюда, и у субъекта принимает заказ Сергей, то заказ принесут через 10 мин.

Если заказ принесут через 20 мин. или заказ принесут через 10 мин., то субъект может есть.

Если субъект может есть, то после еды субъект обязан оплатить заказ.

Введем обозначения для фактов (Ф), действий (Д) и продукций (П), тогда:

Субъект = Петр;

Ф1= субъект хочет есть;

Ф2= у субъекта есть достаточная сумма денег;

Ф3= субъект ближе к ресторану «Вкусная еда», чем к «Вкуснятина»;

Ф4=в ресторане «Вкуснятина» работает официант Марина;

Ф5=в ресторане «Вкусная еда» работает официант Сергей;

Ф6= субъект выбрал блюда;

Д1= субъект может пойти в ресторан;

Д2=субъект идет в ресторан «Вкусная еда»;

Д3=субъект идет в ресторан «Вкуснятина»;

Д4= у субъекта принимает заказ Марина;

Д5=у субъекта принимает заказ Сергей;

Д6=заказ принесут через 20 мин.

Д7=заказ принесут через 10 мин.

Д8=после еды субъект должен оплатить заказ.

Для продукций установим приоритет (в скобках перед запятой, чем выше приоритет, чем раньше проверяется правило).

П1(4 , Ф1 и Ф2)= Д1;

П2(5 , Ф3 и Д1)= Д2;

П3(4 , не Ф3 и Д1)= Д3;

П4(3 , Д3 и Ф4)= Д4;

П5(3 , Д2 и Ф5)= Д5;

П6(2 , Д4)= Д6;

П7(2 , Д5)= Д7;

П8(1 , Д6 или Д7)= Д8;

6 Для отображения взаимосвязи продукций построим граф (рисунок 1).

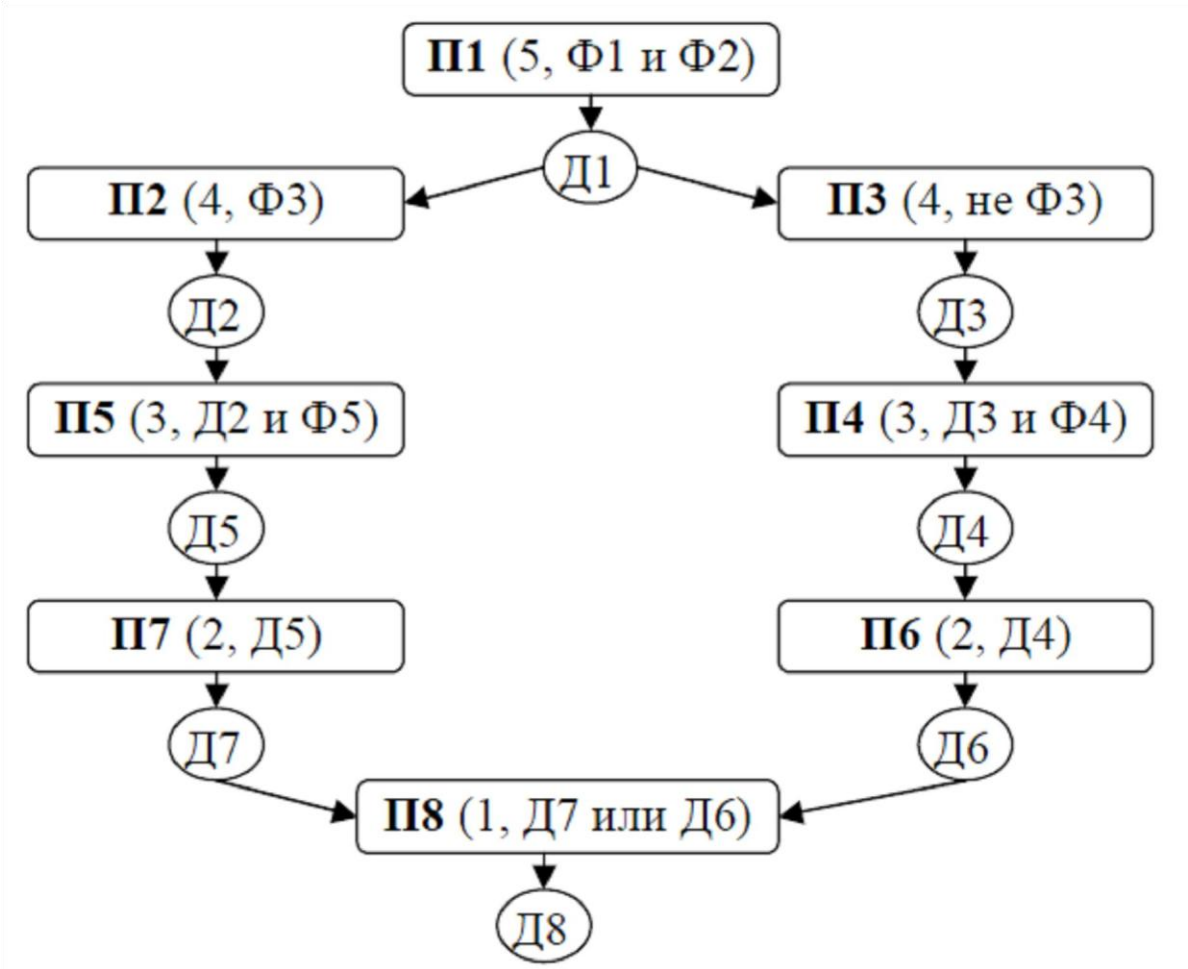

Рисунок 1 – Граф модели знаний

Задание

Выполнить задания согласно указанному варианту.

1 Вариант: построить модель представления знаний в предметной области «Железная дорога» (продажа билетов).

2 Вариант: построить модель представления знаний в предметной области «Торговый центр» (организация).

3 Вариант: построить модель представления знаний в предметной области «Автозаправка» (обслуживание клиентов).

4 Вариант: построить модель представления знаний в предметной области «Компьютерные сети» (организация).

5 Вариант: построить модель представления знаний в предметной области «Университет» (учебный процесс).

6 Вариант: построить модель представления знаний в предметной области «Компьютерная безопасность» (средства и способы ее обеспечения).

7 Вариант: построить модель представления знаний в предметной области «Компьютерная безопасность» (угрозы).

8 Вариант: построить модель представления знаний в предметной области «Интернеткафе» (организация и обслуживание).

9 Вариант: построить модель представления знаний в предметной области «Разработка информационных систем» (ведение информационного проекта).

10 Вариант: построить модель представления знаний в предметной области «Туристическое агентство» (работа с клиентами).

11 Вариант: построить модель представления знаний в предметной области «Кухня» (приготовление пищи).

12 Вариант: построить модель представления знаний в предметной области «Больница» (прием больных).

13 Вариант: построить модель представления знаний в предметной области «Кинопрокат» (ассортимент и работа с клиентами).

14 Вариант: построить модель представления знаний в предметной области «Прокат автомобилей» (ассортимент и работа с клиентами).

15 Вариант: построить модель представления знаний в предметной области «Операционные системы» (функционирование).

16 Вариант: построить модель представления знаний в предметной области «Информационные системы» (виды и функционирование).

17 Вариант: построить модель представления знаний в предметной области «Предприятие» (структура и функционирование).

Контрольные вопросы и задания

1 Что такое семантическая сеть?

2 Что может выступать понятием в семантической сети?

3 Перечислите основные характеристики семантических сетей.

4 Перечислите этапы построения семантической сети.

2 Практическое занятие №2. Выявление знаний в системах искусственного интеллекта. Нечеткая логика. Формирование функций принадлежности в программной среде Fuzzy Logic Toolbox

Цель занятия: ознакомится со способами и средствами описания нечётких множеств и продукций в системе нечёткого вывода в интерактивном режиме использования графических средств пакета Fuzzy Logic Toolbox (в составе MATLAB R2008b).

Краткие теоретические сведения

Для рассмотрения результатов разработки и функционирования систем нечёткой логики будем использовать графические средства пакета Fuzzy Logic Toolbox. Эти же средства используются и при разработке систем нечёткого вывода как графический объектно-ориентированный язык автоматического программирования.

В состав этих средств входят:

– редактор систем нечёткого вывода FIS Editor (FIS) – редактор функций принадлежности систем нечёткого вывода Membership Function Editor (MFE);

– редактор правил систем нечёткого вывода Rule Editor;

– программа просмотра правил системы нечёткого вывода Rule Viewer;

– программа просмотра поверхности нечёткого вывода Sur-face Viewer.

Для описания нечётких высказываний используются нечёткие лингвистические переменные (ЛП).

ЛП – это именованная переменная, которая принимает свои значения из множества лингвистических термов, т.е. символьных величин. Для нечёткой ЛП терм-множество задаётся как нечёткое множество. Этот процесс называется фаззификацией.

Фаззификация является одной из проблемных задач описания нечёткого вывода и отражает индивидуальные эмпирические знания автора. Нечёткие вы-

сказывания в условной части нечёткой продукции могут быть составными, соединёнными связками «И» и/или «ИЛИ». Эти связки при исчислении высказываний реализуются логическими или арифметическими операциями пересечения или объединения, соответственно.

При получении результата по каждому правилу необходимо дать оценку степени его истинности. Эта оценка зависит от степени истинности высказываний условной части правила, степени истинности отношения, положенного в основу правила, между исходными утверждениями (посылкой) и заключением, т.е. степени истинности импликации, и степени истинности высказывания относительно значения из терм-множества возможных результатов, приведенного в правиле. Получение оценки степени истинности заключения, полученного по правилу, называют активизацией.

В случае необходимости получения чёткого количественного значения результата оно может быть получено на основании функции принадлежности терм-множества результата различными способами по алгоритмам, названным по именам их авторов (Мамдани, Сугено, Цукамото и т.д.), что определяет тип системы нечёткого вывода. Эта операция называется дефаззификацией.

Редактор функций принадлежности в графическом режиме обеспечивает задание и изменение функции принадлежности любых термов ЛП систем нечеткого вывода (СНВ) [3].

Методические указания к работе

Для фаззификации лингвистической переменной СНВ следует выделить ее изображение – именованный прямоугольник в левой верхней части окна редактора.

В окне редактора выводятся графики функций принадлежности для всех значений выделенной ЛП (по умолчанию для трёх значений).

Для описания функции принадлежности каждого значения ЛП используются три поля: Name, Type и Params. Описываемая функция выделяется щелчком по её графику. В поле Name устанавливается значение ЛП. В поле Type, выбором элемента меню, устанавливается имя нужной функции принадлежно-

сти (одной из 11-ти встроенных). В поле ввода Params указываются необходимые параметры функции принадлежности, которые определяют положение ее модальных значений на числовой шкале, диапазон изменения которой указывается в полях ввода Range и Display range.

Эти операции выполняются над всеми значениями из терм-множеств лингвистических переменных СНВ.

Добавление нового значения ЛП со встроенной функцией принадлежности производится по команде основного меню Edit > Add MF.

Удаление ненужного значения ЛП производится нажатием клавиши Delete, после выделения графика функции принадлежности этого значения.

Пример

Командой (функцией) Fuzzy из режима командной строки запускается основная интерфейсная программа пакета Fuzzy Logic – редактор нечеткой системы вывода (Fuzzy Inference System Editor, FIS Editor, FIS-редактор).

Вид открывающегося при этом окна приведен на рисунке 2.

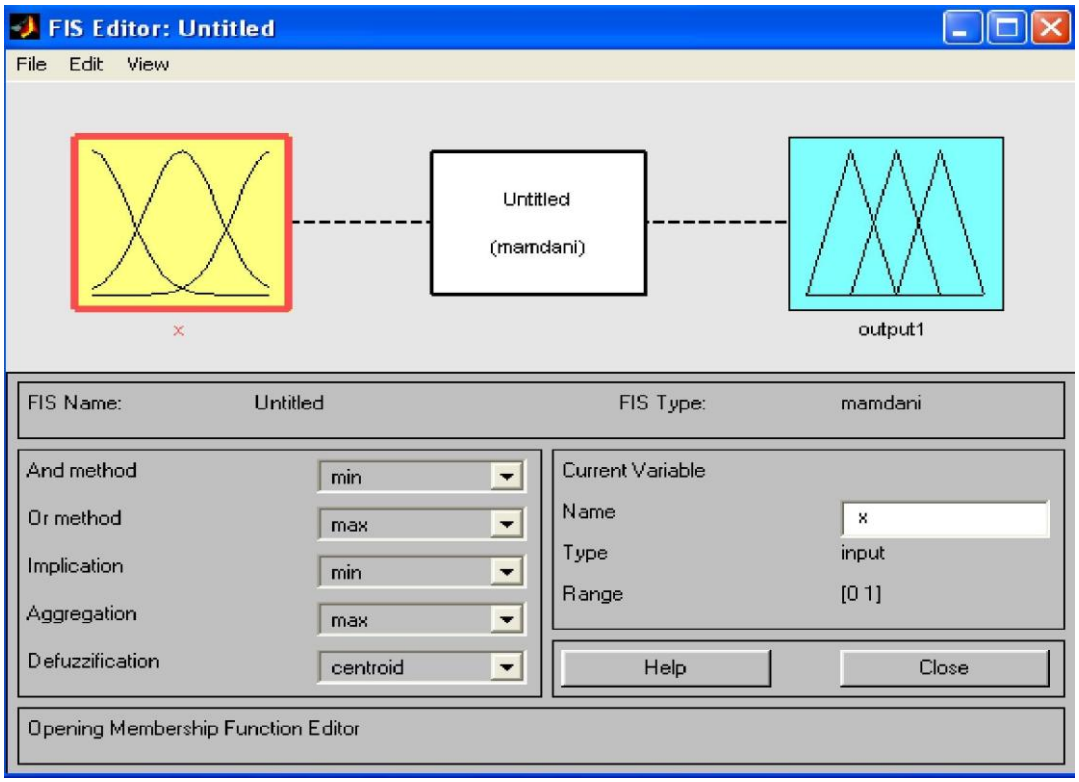

Рисунок 2 – Вид окна Fis Editor

Главное меню редактора содержит позиции:

File – работа с файлами моделей (их создание, сохранение, считывание и печать);

Edit – операции редактирования (добавление и исключение входных и выходных переменных);

View – переход к дополнительному инструментарию.

Нужно разработать нечеткую систему, отображающую зависимость между переменными x и y, заданную с помощью таблицы 1, представленные в таблице данные отражают зависимость $y = x^2$.

Таблица 1 – Значения *x* и *y*

x	-1	- 0.6	0	0.4	1
y	1	0.36	0	0.16	1

Требуемые действия необходимо отобразить следующими пунктами.

1 В позиции меню File выбрать опцию New Sugeno FIS (новая система типа Sugeno), при этом в блоке, отображаемом белым квадратом, в верхней части окна редактора появится надпись Untitled2 (sugeno).

2 Щелкнуть левой кнопкой мыши по блоку, озаглавленному inputl (вход 1). Затем в правой части редактора в поле, озаглавленном Name (Имя), вместо inputl ввести обозначение аргумента, т.е. *x*. Если теперь сделать где-нибудь (вне блоков редактора) однократный щелчок мыши, то имя отмеченного блока изменится на x; то же достигается нажатием после ввода клавиши Enter.

3 Дважды щелкнуть по этому блоку. В этом случае откроется окно редактора функций принадлежности – Membership Function Editor (см. рисунок 3). Необходимо войти в позицию меню Edit данного редактора и выбрать в нем опцию Add MFs (Add Membership Funcions – Добавить функций принадлежности). При этом появится диалоговое окно (рисунок 4), позволяющее задать тип (MF type) и количество (Number of MFs) функций принадлежности (в данном случае все относится к входному сигналу, т. е. к переменной *x*). Необходимо

выбрать гауссовы функции принадлежности (gaussmf), а их количество зададим равным пяти – по числу значений аргумента в таблице 1. Затем необходимо подтвердить ввод информации нажатием кнопки ОК, после чего произойдет возврат к окну редактора функций принадлежности.

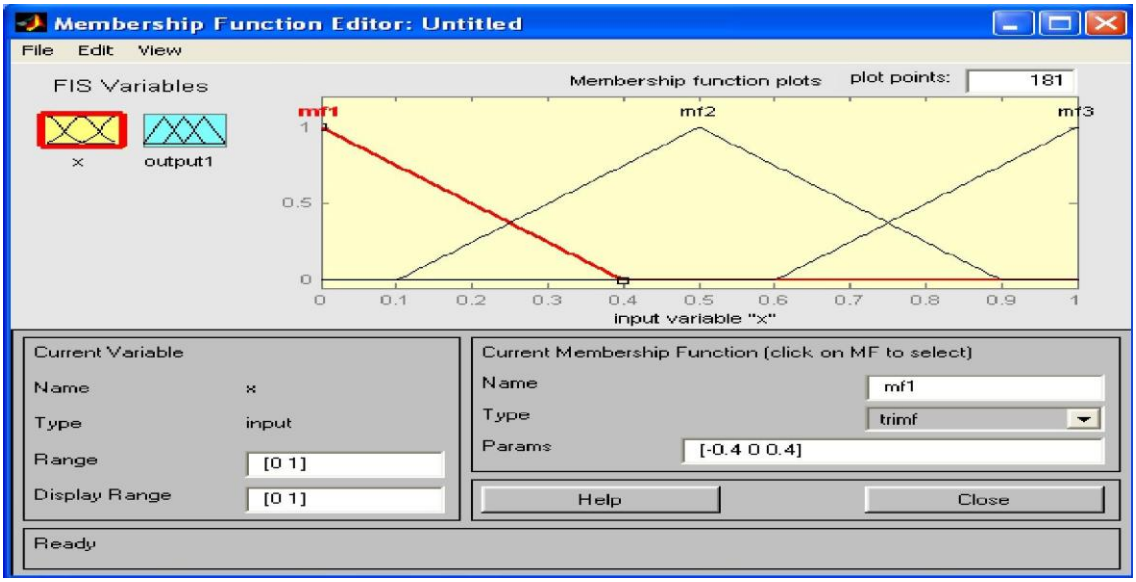

Рисунок 3 – Окно редактора функций принадлежности

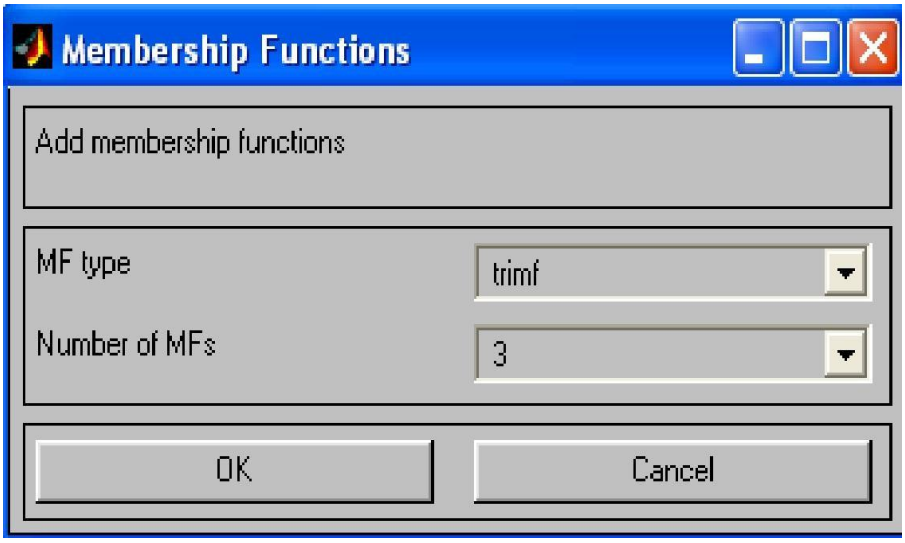

Рисунок 4 – Диалоговое окно задания типа и количества функций принадлежности

4 В поле Range (Диапазон) нужно установить диапазон изменения x от -1 до +1, т.е. диапазон, соответствующий таблице 1. Щелкнуть затем левой кнопкой мыши где-нибудь в поле редактора (или нажать клавишу ввода Enter). После этого произойдет соответствующее изменение диапазона в поле Display Range (Диапазон дисплея).

5 Обратимся к графикам заданных функций принадлежности. Для успешного решения поставленной задачи необходимо, чтобы ординаты максимумов этих функций совпадали с заданными значениями аргумента *x*. Для левой, центральной и правой функций такое условие выполнено, но две другие необходимо «подвинуть» вдоль оси абсцисс. «Передвижка» делается весьма просто: подвести курсор к нужной кривой и щелкнуть левой кнопкой мыши. Кривая выбирается, окрашиваясь в красный цвет, после чего с помощью курсора ее и можно подвинуть в нужную сторону (более точную установку можно провести, изменяя числовые значения в поле Params (Параметры) – в данном случае каждой функции принадлежности соответствуют два параметра, при этом первый определяет размах кривой, а второй – положение ее центра). Для выбранной кривой, кроме этого, в поле Name можно изменять имя (завершая ввод каждого имени нажатием клавиши Enter).

Далее необходимо задать всем пяти кривым новые имена, например:

– самой левой – bn,

– следующей – n,

– центральной – z,

– следующей за ней справа – p, самой левой – bp.

Нажать кнопку Close и выйти из редактора функций принадлежности, возвратившись при этом в окно редактора нечеткой системы (FIS Editor).

6 Сделать однократный щелчок левой кнопкой мыши по голубому квадрату (блоку), озаглавленному outputl (выход 1). В окошке Name заменить имя outputl на *y* (как в пункте 2).

7 Дважды щелкнуть по отмеченному блоку и перейти к программе – редактору функций принадлежности. В позиции меню Edit выбрать опцию Add

MFs. Появляющееся диалоговое окно вида позволяет задать теперь в качестве функций принадлежности только линейные (linear) или постоянные (constant) – в зависимости от того, какой алгоритм Sugeno (1-го или 0-го порядка) необходимо выбрать. В рассматриваемой задаче необходимо выбрать постоянные функции принадлежности с общим числом 4 (по числу различных значений y в таблице 1). Необходимо подтвердить введенные данные нажатием кнопки ОК, после чего произойдет возврат в окно редактора функций принадлежности.

8 В диапазоне (Range) изменения, устанавливаемый по умолчанию – [0, 1], менять не нужно. Изменить необходимо имена функций принадлежности (их графики при использовании алгоритма Sugeno для выходных переменных не приводятся), например, задав их как соответствующие числовые значения *y*, Затем необходимо вернуться в окно FIS-редактора.

9 Дважды щелкнуть левой кнопкой мыши по среднему (белому) блоку, при этом раскроется окно еще одной программы – редактора правил (Rule Editor). Необходимо ввести соответствующие правила. При вводе каждого правила необходимо обозначить соответствие между каждой функцией принадлежности аргумента x и числовым значением *y*.

Аналогично необходимо поступить для других значений, в результате сформируется набор из пяти правил. Построение системы на этом закончено.

Задание

Для полученного варианта (см. практическое задание №1, стр. 9-10) и разработанного графа ПрО сформировать лингвистические переменные и получить для них функции принадлежности.

Контрольные вопросы и задания

1 Что такое лингвистическая переменная?

2 Что такое фаззификация?

3 Что такое активизация?

4 Как может быть получено чёткое количественное значение результата?

3 Практическое занятие №3. Построение моделей в системах искусственного интеллекта (декларативный язык Prolog)

Цель занятия: изучить среду визуальной разработки Visual Prolog. Создать проект и запустить его на выполнение.

Краткие теоретические сведения

Пролог был разработан в Марсельском университете во Франции Алэном Колмероэ и другими членами «группы искусственного интеллекта» в 1973 году. Данная команда энтузиастов разработала программу, предназначенную для доказательства теорем. Написана она была на Фортране, и использовалась при построении систем обработки текстов на естественном языке. Работала программа довольно медленно и назвалась Prolog (от Programmation en Logique на франц.). Программа послужила прообразом Prolog .

Язык логического программирования Prolog – декларативный, что означает, что программа, написанная на нем, выглядит как набор логических описаний, определяющих цель, ради которой она и написана.

В основе Prolog лежит раздел математической логики, называемый *исчисление предикатов*. Его базис составляет процедура доказательства теорем *методом резолюции* для *хорновских дизъюнктов*.

Логика предикатов – это простейший способ объяснить, как «работает» мышление.

По сути, Prolog представляет собой не столько язык для программирования, сколько средство для описания данных и логики их обработки. Поскольку язык не алгоритмический, то и программа на Prolog не содержит явных алгоритмических конструкций типа условных или циклических операторов, т.е. не является программой в классическом понимании. Написанная на Prolog , она представляет собой модель фрагмента предметной области, на решение проблемы которой и направлена задача.

Само решение задачи записывается не в терминах компьютера, как это принято в императивных языках, а в терминах предметной области решаемой задачи, по сути, в духе объектно-ориентированного программирования.

Prolog включает механизм вывода, который основан на сопоставлении образцов, с помощью подбора ответов на запросы он извлекает хранящуюся (известную) информацию [4].

Методические указания к работе

Для того, чтобы запустить Visual Prolog, необходимо выполнить следующие действия: Пуск | Программы | Visual Prolog 5.2 | Visual Prolog 32. При этом открывается основное окно, которое называется окном Task (рисунок 5).

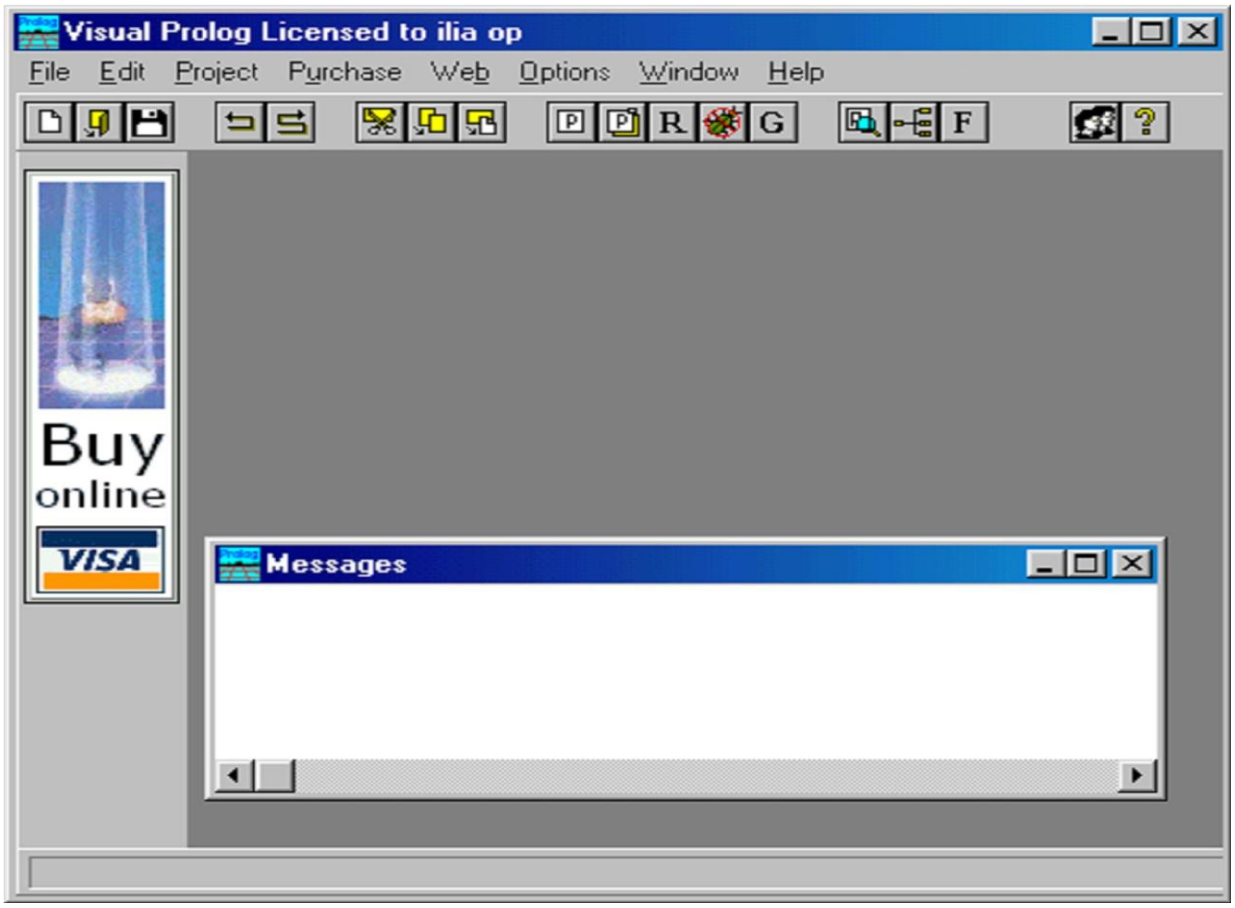

Рисунок 5 – Окно Task

Обычно в данном окне доступны меню File, Edit, Project, Options, Help и Window, но при активизации некоторых других окон в меню могут появиться дополнительные пункты.

Часто используемые команды меню могут быть выполнены и при помощи кнопок на панели инструментов (рисунок 6).

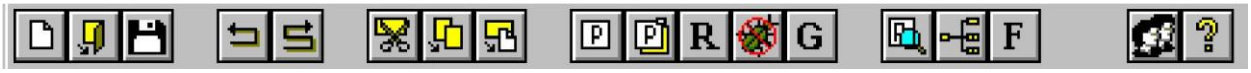

Рисунок 6 – Панель инструментов

Каждая из пиктограмм на панели инструментов выполняет ту же функцию, что и соответствующая команда меню.

В нижней части окна Task расположена строка подсказки. Она разделена на две части.

Левое поле используется для отображения контекстно-зависимой информации, например, подсказок для командных кнопок на панели инструментов или информации о текущем элементе управления в редакторе диалоговых окон и т. д.

Крайнее правое поле используется построителем программ (make facility) для отображения состояний генерации/компиляции/компоновки текущего ресурса.

Для создания проекта требуется определить некоторые (непредопределенные) опции компилятора Visual Prolog. Для этого необходимо выполнить следующие действия:

1 Запустить среду визуальной разработки Visual Prolog. При первом запуске VDE () проект не будет загружен, и отобразится окно, показанное на рисунке 4. Также поступит информация, что по умолчанию создан инициализационный файл для Visual Prolog VDE.

2 Создать новый проект.

Выбрать команду Project | New Project, активизируя диалоговое окно Application Expert.

3 Определить базовый каталог и имя проекта.

Имя в поле Project Name следует определить как «Test». Далее необходимо щелкнуть мышью внутри поля Name of .VPR File. Также установить флажок Mulltiprogrammer Mode и щелкнуть мышью внутри поля Name of.PRJ File. Появится имя файла проекта Test.prj (рисунок 7).

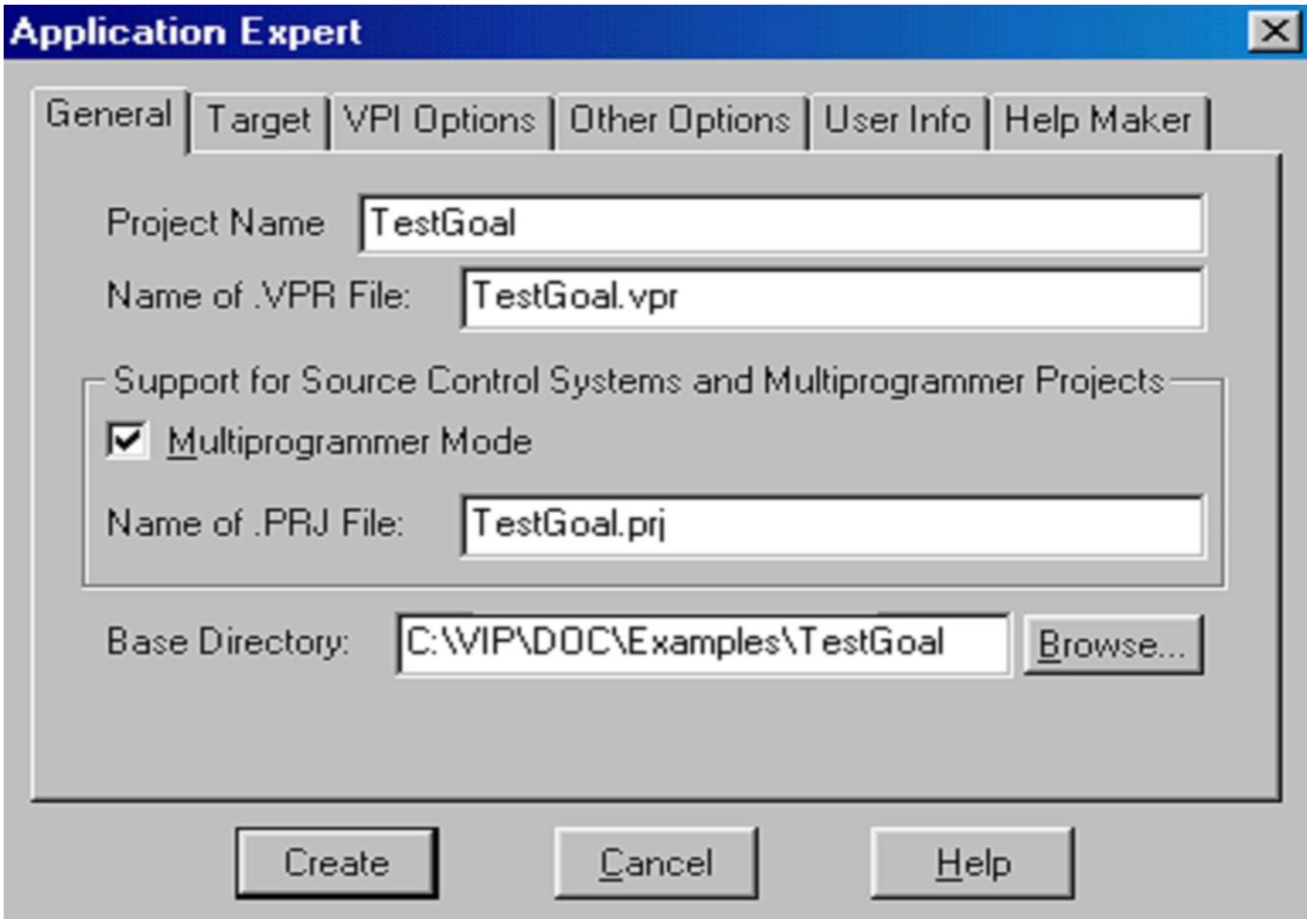

Рисунок 7 – Общие установки диалогового окна Application Expert

Определить цель проекта. На вкладке Target рекомендуется выбрать параметры, отмеченные на рисунке 8.

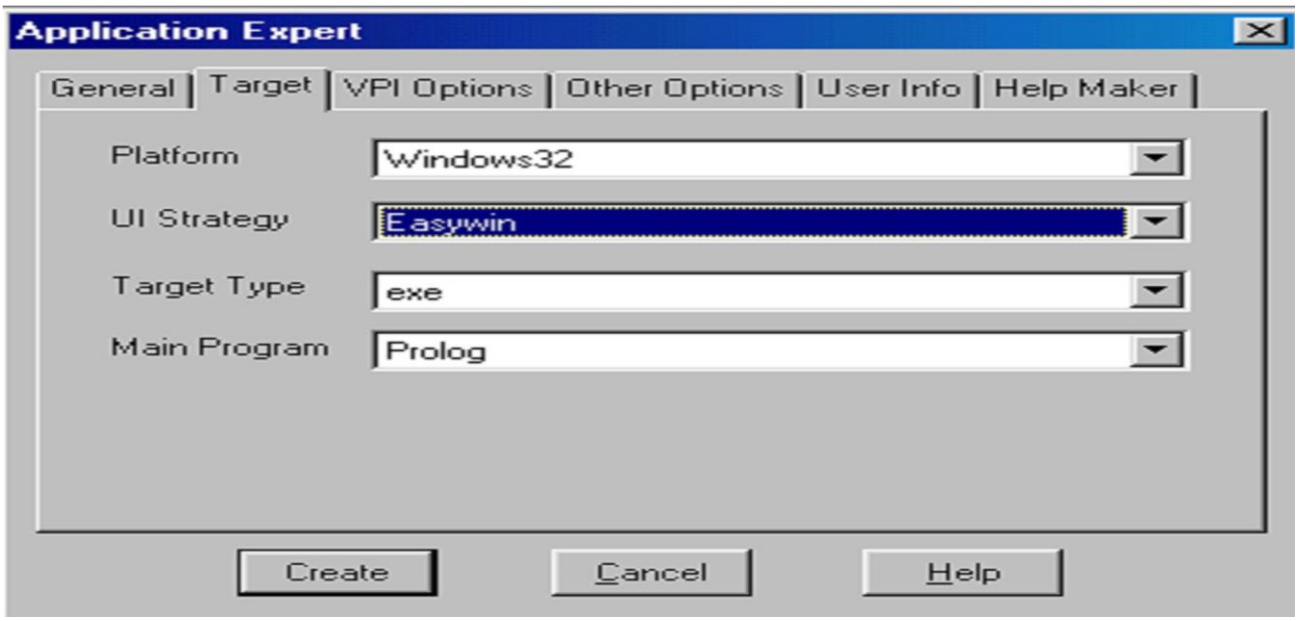

Рисунок 8 – Установки на вкладке Target диалогового окна Application Expert

Нажать кнопку Create для того, чтобы создать файлы проекта по умолчанию.

4 Установить требуемые опции компилятора для созданного проекта. Для активизации диалогового окна Compiler Options выбрать команду Options | Project | Compiler Options. Открыть вкладку Warnings. Выполнить следующие действия:

– установить переключатель Nondeterm. Это нужно для того, чтобы компилятор Visual Prolog принимал по умолчанию, что все определенные пользователем предикаты – недетерминированные (могут породить более одного решения);

– снять флажки Not Quoted Symbols, Strong Type Conversion Check и Check Type of Predicates. Это будет подавлять некоторые возможные предупреждения компилятора;

– нажать кнопку ОК, чтобы сохранить установки опций компилятора. В результате этих действий диалоговое окно Compiler Options будет выглядеть, как показано на рисунке 9.

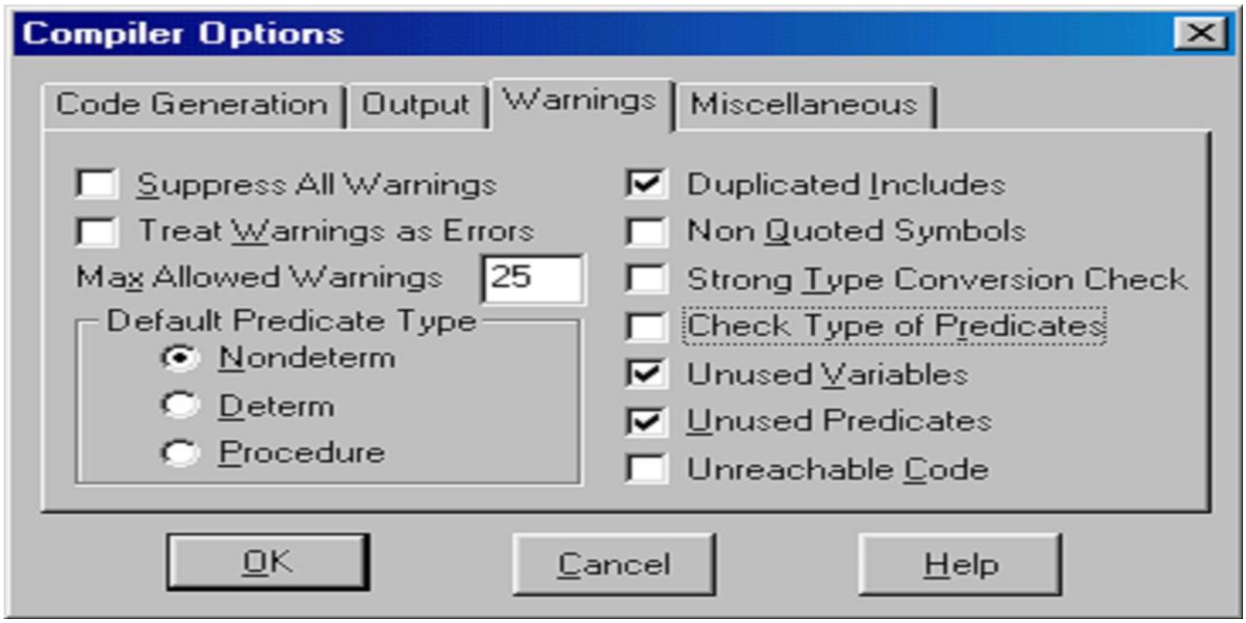

Рисунок 9 – Установки опций компилятора

Для проверки настройки системы должным образом, следует выполнить следующие действия:

1 В окне проекта открыть файл test.pro (рисунок 10)

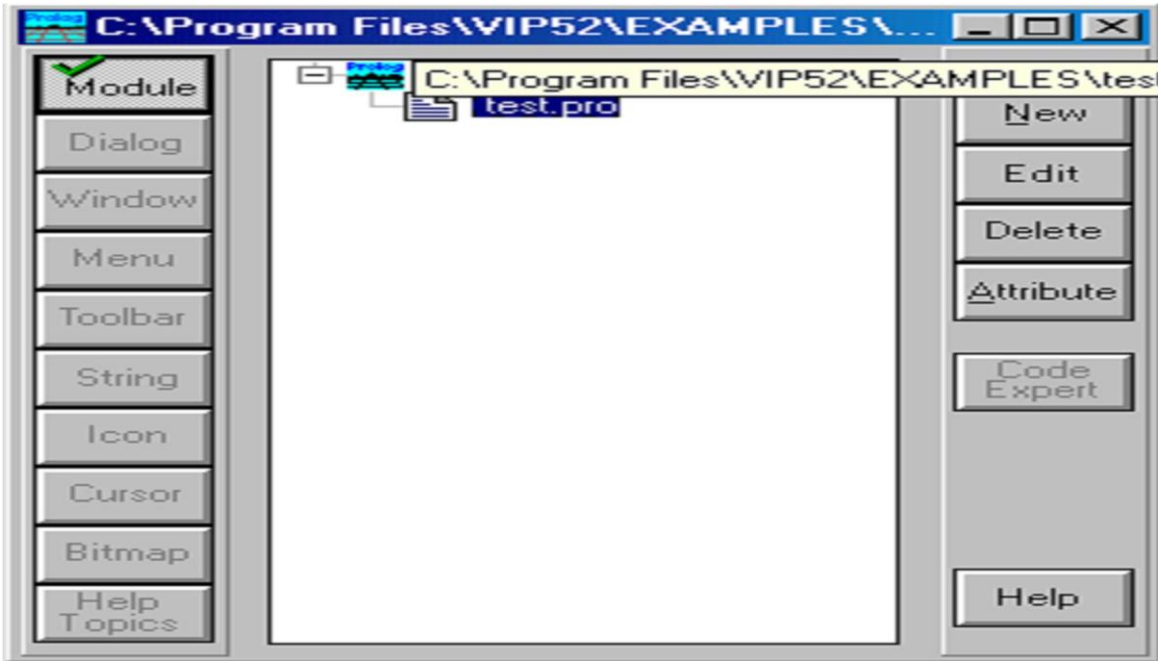

Рисунок 10 – Окно проекта

2 В разделе GOAL набрать с клавиатуры write («Hello world»), nl .

3 Нажать на панели инструментов кнопку (либо комбинацию клавиш <Ctrl>+<G>, либо активировать команду Project | Test Goal). В терминологии языка Пролог это называется GOAL, и этого достаточно для программы, чтобы она могла быть выполнена. Если система установлена правильно, то экран монитора будет выглядеть, как показано на рисунке 11.

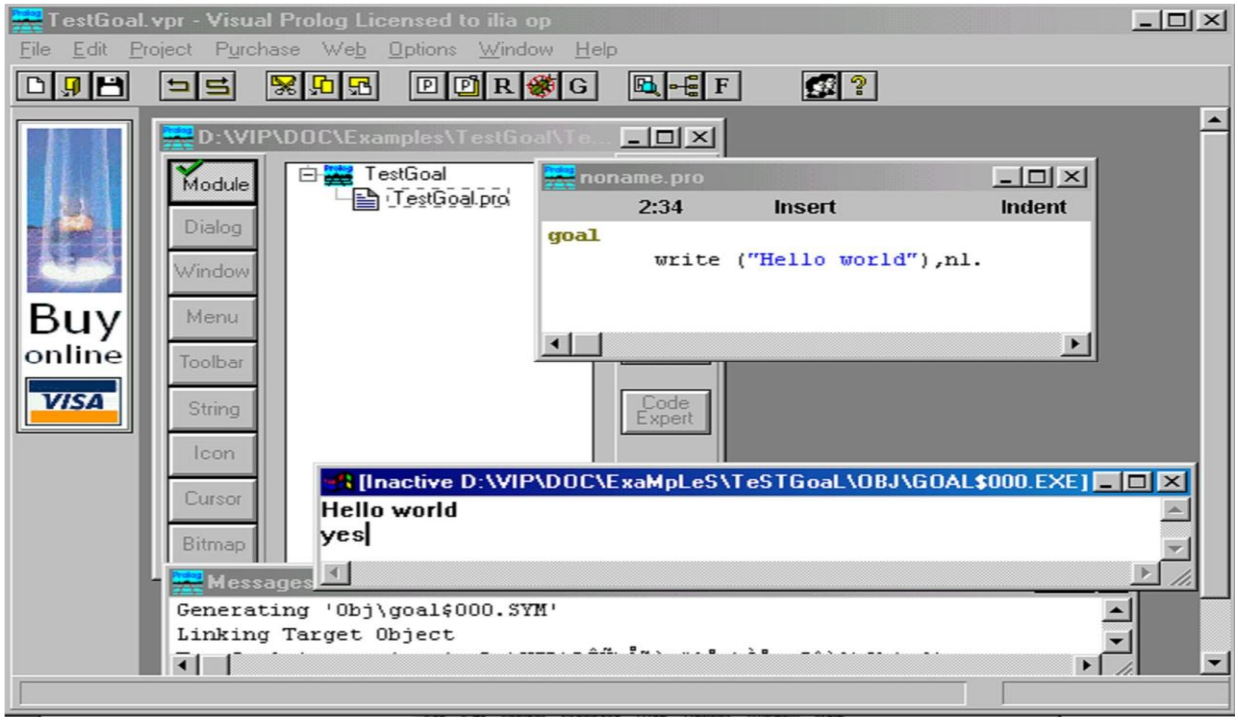

Рисунок 11 – Тестовая программа «Hello world»

Результат выполнения программы будет расположен вверху в отдельном окне, которое необходимо закрыть перед тем, как тестировать другую GOAL.

Утилита среды визуальной разработки интерпретирует GOAL как специальную программу, которая компилируется, компонуется, генерируется в исполняемый файл и Test Goal запускает его на выполнение. Эта утилита внутренне расширяет заданный код GOAL, чтобы сгенерированная программа находила все возможные решения и показывала значения всех используемых переменных. Утилита Test Goal компилирует этот код с использованием опций компилятора, заданных для открытого проекта (рекомендуемые опции компилятора для TestGoal-проекта определили ранее).

Утилита Test Goal компилирует только тот код, который определен в активном окне редактора (код в других открытых редакторах или модулях проектов, если они есть, игнорируется).

При компоновке исполняемого файла TestGoal использует стратегию EASYWIN. Нельзя определить какие-либо опции компоновки для TestGoa1, т.к. игнорируются любые установки Make Options, заданные для открытого проекта. Поэтому TestGoa1 не может использовать никакие глобальные предикаты, определенные в других модулях.

Утилита имеет ограничение на количество переменных, которые могут быть использованы в GOAL. На данный момент их 12 для 32-разрядной среды визуальной разработки, но это число может быть изменено без дополнительных уведомлений.

Если допущены ошибки в программе и осуществляется попытка скомпилировать ее, то среда визуальной разработки отобразит окно Errors (Warnings), которое будет содержать список обнаруженных ошибок.

Дважды щелкнув на одной из этих ошибок, можно попасть на место ошибки в исходном тексте. Можно воспользоваться клавишей <F1> для вывода на экран интерактивной справочной системы Visual Prolog. Когда окно помощи откроется, необходимо щелкнуть по кнопке Search, набрать номер ошибки, и на экране появится окно помощи с более полной информацией о ней.

Команды построения

1 Команда Project / Compile Module

Эта команда (ей соответствует комбинация клавиш <Ctrl>+<F9>) делает попытку компилировать модуль, содержащий редактируемый в данный момент файл. Выполнение команды зависит от следующих свойств файла:

– если файл имеет расширение pro и является модулем текущего проекта, то VDE пытается компилировать этот файл;

– если файл не является модулем текущего проекта и его расширение – pro, pre, inc, con или dom, то VDE пытается найти модуль проекта, который включает этот файл, и откомпилировать первый найденный модуль;

− во всех остальных случаях VDE пытается компилировать модуль, выбранный в окне проекта. VDE не может компилировать файл, который не является частью открытого проекта. Вместо этого файла VDE будет компилировать модуль, выбранный в окне проекта.

Если в VDE не открыт ни один проект, то никакие файлы компилироваться не будут. Команда меню Project | Compile Module заблокирована; комбинация клавиш <Ctrl>+<F9> не работает. Единственно возможное действие – это запустить утилиту Test Goal.

2 Команда Project / Build

Если со времени последнего построения проекта были изменены какие-либо ресурсы, то эксперты кода могут обновить некоторые секции в исходных файлах перед построением.

Эта команда (ей соответствует комбинация клавиш <Alt>+<F9>) строит проект, проверяя метки времени всех исходных файлов в проекте, поэтому если исходные файлы (или файлы, которые в них включены) являются более новыми, чем зависимые OBJ-файлы, то соответствующие модули проекта будут перекомпилированы.

Команда Build также строит файлы ресурсов и файл интерактивной справки (если необходимо). Затем проект компонуется для генерации целевого модуля (исполняемая программа или DLL).

3 Команда Project / Rebuild All

Эта команда (ей соответствует комбинация клавиш <Ctrl>+<Alt>+<F9>) выполняет то же действие, что и Project | Build, причем все файлы будут повторно сгенерированы или откомпилированы и скомпонованы независимо от их меток времени.

4 Команда Project I Stop Building

Эта команда (ей соответствует комбинация клавиш <Alt>+<F10>) используется для остановки компиляции/компоновки.

5 Команда Project I Run

Если необходимо, то эта команда (ей соответствует клавиша <F9>) выполнит действие Project | Build и затем запустит сгенерированный исполняемый файл.

6 Команда Project I Link Only

Эта команда (ей соответствует комбинация клавиш <Shift>+<F9>) используется для выполнения компоновки. В этом случае построитель программ вызывает компоновщика и не проверяет, нужно ли повторно компилировать какие-либо модули проекта (или даже впервые компилировать).

7 Команда Project / Test Goal

Эта команда (ей соответствует комбинация клавиш <Ctrl>+<G>) используется для тестирования простых целей (Goals). Программа компилируется и компонуется в специальном режиме, и затем запускается соответствующий исполняемый файл.

Утилита Test Goal ищет все решения для определенной в программе цели. Для каждого решения Test Goal отображает значения всех переменных из секции GOAL и число решений. Эта особенность – удобный способ проверить локальные предикаты в модуле.

Например, следующая цель: GOAL X = 2; X = 1, Y = X + 1 приведет к такому результату (рисунок 12).

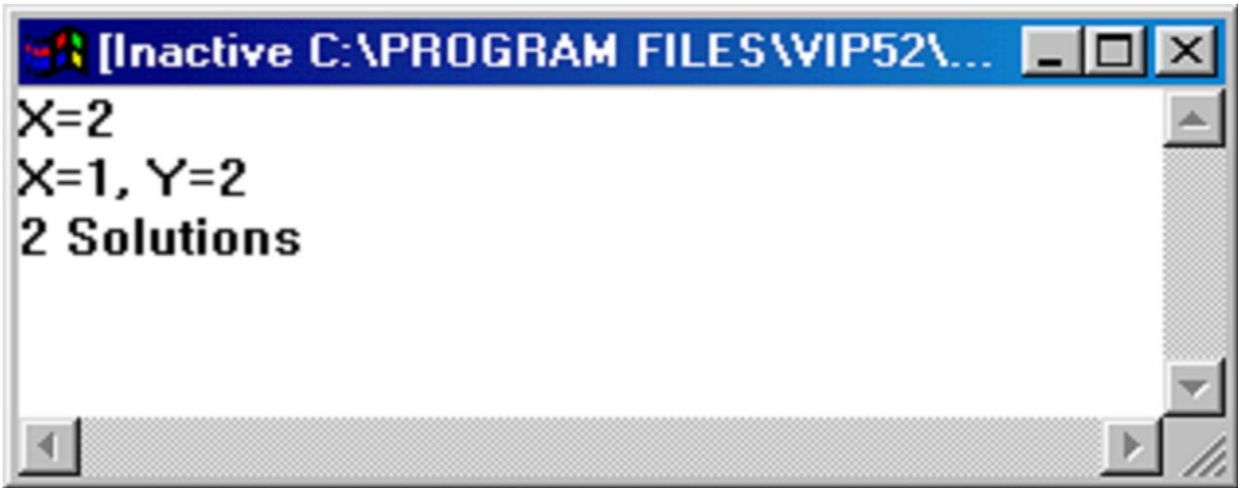

Рисунок 12 – Вывод режима тестирования цели

8 Команда Resource I Build Resource Only

Когда окно проекта активизировано, в меню Project появляется команда Resource. При выборе этого пункта (или нажатии комбинации клавиш <Alt>+<F8>) генерируются выбранные файлы с расширениями rc и res и необходимые файлы констант.

Пример

Загрузить программу в среду визуальной разработки Visual Prolog и запустить ее утилитой Test Goal.

```
predicates
likes(symbol,symbol)
clauses
likes(ellen,tennis).
likes(john,football).
likes(tom,baseball).
likes(eric,swimming).
likes(mark,tennis) .
likes(bill,Activity):-likes (tom, Activity) .
goal
likes(bill, baseball).
```

Утилита Test Goal ответит в окне приложения: yes (да)

Попробовать следующий запрос в GOAL-разделе: likes(bill, tennis). Утилита Test Goal ответит: no (нет).

Команда Project | Debug запускает процесс отладки. Отладчик также можно запустить сочетанием клавиш CTRL+SHIFT+F9.

При помощи диалога View можно открывать дополнительные информационные окна, которые отображают различные состояния среды и переменных в режиме отладки:

View → Call Stack (Открывает информационное окно стека вызова) View → Local Variables (Открывает информационное окно локальных переменных)

Для выполнения шагов отладки используются следующие команды:

Run → Trace Intro [F7]

Run → Step Over [F8]

Run → Run to Cursor [F4]

Задание

Подготовить программу на языке ПРОЛОГ для полученного варианта (см. практическое задание №1, стр. 9-10). Запустить программу в среде Visual Prolog в режиме отладки.

Контрольные вопросы и задания

1 Что такое логика предикатов?

2 Дайте характеристику языку Prolog.

3 Назовите особенности записи решения задачи на языке Prolog.

4 Опишите механизм вывода, реализованного в языке Prolog.

4 Практическое занятие №4. Продукции в системах искусственного интеллекта

Цель занятия: изучение механизма вывода в продукционных системах.

Краткие теоретические сведения

Основной особенностью интеллектуальных систем является то, что они основаны на знаниях, а вернее, на некотором их представлении. Знания понимаются как хранимая (с помощью ЭВМ) информация, формализованная в соответствии с некоторыми правилами, которую ЭВМ может использовать при логическом выводе по определенным алгоритмам. Наиболее фундаментальной и важной проблемой является описание смыслового содержания проблем самого широкого диапазона, т.е. должна использоваться такая форма описания знаний, которая гарантировала бы правильную обработку их содержимого по некоторым формальным правилам. Эта проблема называется проблемой представления знаний [5].

Продукционные правила – наиболее простой способ представления знаний. Он основан на представлении знаний в форме правил, структурированных в соответствии с образцом «ЕСЛИ – ТО». Часть правила «ЕСЛИ» называется посылкой, а «ТО» – выводом или действием. Правило в общем виде записывается так:

ЕСЛИ A_1, A_2, …,A_n, ТО B.

Такая запись означает, что «если все условия от A1 до An являются истинными, то B также истинно» или «когда все условия от A1 до An выполняются, то следует выполнить действие B».

Переменные *x, y* и *r* показывают, что правило содержит некое универсальное, общее знание, абстрагированное от конкретных значений переменных. Одна и та же переменная, использованная в выводе и различных посылках, может получать различные конкретные значения.

Знания, представленные в интеллектуальной системе, образуют базу знаний. В интеллектуальную систему входит также механизм выводов, который позволяет на основе знаний, имеющихся в базе знаний, получать новые знания.

Формализованная процедура, использующая сопоставление (при котором устанавливается, совпадают ли между собой две формы представления, включая подстановку возможных значений переменных), поиск в базе знаний, возврат к исходному состоянию при неудачной попытке решения, представляет собой механизм выводов.

Простота и наглядность представления знаний с помощью продукций обусловила его применение во многих системах, которые называются продукционными [6].

Методические указания к работе

Продукционная система состоит из трех основных компонентов. Первый из них – это набор правил, используемый как база знаний, иногда его еще называют базой правил. Второй компонент – это база фактов или рабочая память – память для временного хранения, в которой хранятся предпосылки, касающиеся конкретных задач предметной области, и результаты выводов, получаемые на их основании. Третий компонент реализует механизм логического вывода, обрабатывающий правила в соответствии с содержанием рабочей памяти; другое название этого компонента – машина логического вывода.

Существует и другой способ вывода, называемый обратным выводом. При его использовании система начинает работу с формулировки того, что требуется доказать, например, предполагая, что ситуация присутствует, и выполняет только те правила, которые имеют отношение к доказательству предположения.

Если на каждом этапе логического вывода существует множество применяемых правил, то это множество носит название конфликтного набора, а выбор одного из них называется разрешением конфликта. Чтобы повысить эффективность продукционной системы, необходимо решить проблему управления последовательностью применения правил или управления выводом.

Пример

База правил:

- если B и C и D, то A;
- если F и G, то B;
- если H и D, то E;
- если E, то A.
- рабочая память G, H, D, F.

Рассмотрим, каким образом «работают» правила. Система построена так, что один раз выбранное правило из базы правил выполняться будет только один раз. Оно как бы «выгорает». Первым выгорает правило «$F \wedge G \rightarrow B$», а рабочая память дополнится фактом B, затем обрабатываем правило «$H \wedge D \rightarrow E$», в рабочую память будет помещен факт E. Далее вызывает выгорание правила «$B \wedge C \wedge D \rightarrow A$», и, как следствие, выводится ситуация A, и она заносится в базу данных и последнее правило «$E \rightarrow A$».

После чего система остановится, так как правил для выполнения больше не будет.

Такой способ получения новых фактов называется прямым выводом или выводом от данных к цели потому, что поиск новых сведений здесь происходит в направлении стрелки, разделяющей левые и правые части правил. Вывод в данном случае проведен в четыре этапа.

Если речь идет о выводе единственного конкретного факта, то в этом случае прямой вывод может оказаться неэффективным. В такой ситуации, с точки зрения затрат, более целесообразным может оказаться использование процедуры обратного вывода. При использовании данного способа вывода система начинает работу с вывода A.

Шаг 1. В систему поступает указание о необходимости вывода A. Сначала осуществляется проверка наличия A в базе фактов и после отрицательного ответа поиск правил, заканчивающихся на A, то есть имеющихся A справа от стрелки. Система находит два правила «$B \wedge C \wedge D \rightarrow A$», «$E \rightarrow A$» и принимает

решение о том, что для подтверждения A необходимо установить наличие или B и C и D, или E.

Шаг 2. Система пытается установить B в результате проверки сначала базы фактов и последующего обнаружения правил, приводящих к B. Обнаружено правило «$F \wedge G \rightarrow B$». Система приходит к решению о том, что для получения вывода о присутствии B ей необходимо установить наличие F и G.

Шаг 3. Система обнаруживает F и G в базе фактов, подтвердив возможность вывода B.

Шаг 4. Система пытается установить E в результате проверки сначала базы фактов и последующего обнаружения правил, приводящих к E. Обнаружено одно такое правило: «$H \wedge D \rightarrow E$». Система приходит к решению о том, что для получения вывода о присутствии E ей необходимо установить наличие H и D.

Шаг 5. Система выполняет правило «$H \wedge D \rightarrow E$» и устанавливает факт E.

Шаг 6. Система выполняет правило «$F \wedge G \rightarrow B$» и устанавливает факт B.

Шаг 7. Система выполняет правило «$E \rightarrow A$» и устанавливает факт A.

Шаг 8. Система выполняет правило «$B \wedge C \wedge D \rightarrow A$» и устанавливает факт A.

Сформированная в данном случае цепочка логического вывода не отличается от образованной при выполнении процедуры прямого вывода. Различие двух подходов связано со способом, с помощью которого осуществляется поиск фактов и правил.

Если на каждом этапе логического вывода существует множество применяемых правил, то это множество носит название конфликтного набора, а выбор одного из них называется разрешением конфликта. Чтобы повысить эффективность продукционной системы, необходимо решить проблему управления последовательностью применения правил или управления выводом. Ниже представлен пример определенной ранее продукционной системы с пятью правилами при выводе факта A (рисунок 13).

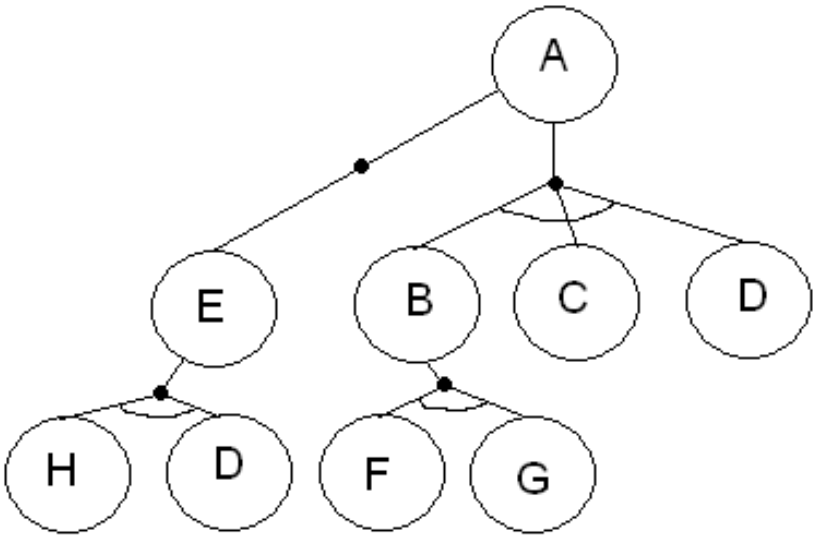

Рисунок 13 – Правила при выводе факта А

К факту А можно пройти, исходя из рабочей памяти H, D и F, G и B, C, D, в этом и заключается конфликт, то есть на данном этапе логического вывода существует множество применяемых правил, если выбрать одно из них, то будет разрешение конфликта.

Из графа видно, что для логически минимального вывода факта А требуется набор правил в следующем порядке:

– если H D, то E $H \wedge D \rightarrow E$
– если E, то A $E \rightarrow A$

Задание

База правил и рабочая память в продукционной системе имеет содержимое, заданное в вариантах (см. практическое задание №1, стр. 9-10). Проиллюстрировать графически механизм прямого и обратного логического вывода факта А. Обратить внимание на изменение содержимого рабочей памяти в процессе вывода. Провести упорядочение правил вывода. Рассмотреть возможные конфликты при прямом и обратном выводе.

Контрольные вопросы и задания

1 Назовите основную особенность интеллектуальных систем.

2 В чем суть проблемы представления знаний?

3 Что такое продукционные правила?

4 Опишите механизм вывода в системе продукций.

5 Практическое занятие №5. Фреймовые модели представления знаний

Цель занятия: изучение представления статических знаний на основе фреймов.

Краткие теоретические сведения

Фреймовая модель основана на концепции Марвина Мински (Marvin Minsky) – профессора Массачусетского технологического института, основателя лаборатории искусственного интеллекта, автора ряда фундаментальных работ. Фреймовая модель представляет собой систематизированную психологическую модель памяти человека и его сознания.

Фрейм (англ. frame – рамка, каркас) – структура данных для представления некоторого концептуального объекта. Информация, относящаяся к фрейму, содержится в составляющих его слотах.

Слот (англ. slot – щель, прорезь) может быть терминальным (листом иерархии) или представлять собой фрейм нижнего уровня.

Каждый фрейм состоит из произвольного числа слотов, причем несколько из них обычно определяются самой системой для выполнения специфических функций, а остальные определяются пользователем.

Фреймы образуют иерархию. Иерархия во фреймовых моделях порождает единую многоуровневую структуру, описывающую либо объект, если слоты описывают только свойства объекта, либо ситуацию или процесс, если отдельные слоты являются именами процедур, присоединенных к фрейму и вызываемых при его актуализации.

Формально фрейм – это тип данных вида:

$F = < N, S_1, S_2, S_3 >$

N – имя объекта;

S_1 – множество слотов, содержащих факты, определяющие декларативную семантику фрейма;

S_2 – множество слотов, обеспечивающих связи с другими фреймами (казуальные, семантические и т. д.);

S_3 – множество слотов, обеспечивающих преобразования, определяющие процедурную семантику фрейма.

Фреймы подразделяются на:

– фрейм-экземпляр – конкретная реализация фрейма, описывающая текущее состояние в предметной области;

– фрейм-образец – шаблон для описания объектов или допустимых ситуаций предметной области;

– фрейм-класс – фрейм верхнего уровня для представления совокупности фреймов образцов.

Состав фреймов и слотов в каждой конкретной фреймовой модели может быть разный, однако в рамках одной системы целесообразно единое представление для устранения лишнего усложнения.

Разнотипные объекты или объекты, соответствующие концепции «множественности миров», заключающейся, к примеру, в том, что лошадь – животное бескрылое для одного (реального) мира и одновременно крылатое (Пегас в мифическом мире) для другого, могут описываться отличающимися друг от друга фреймами.

В целом фреймовая модель допускает представление всех свойств декларативных и процедурных знаний. Глубина вложенности слотов во фрейме (число уровней) зависит от предметной области и языка, реализующего модель [7].

Методические указания к работе

Фрейм имеет определенную внутреннюю структуру, состоящую из множества элементов, называемых слотами, которым также присваиваются имена. За слотами следуют шпации, в которые помещают данные, представляющие текущие значения слотов. Каждый слот в свою очередь представляется определенной структурой данных. В значение слота подставляется конкретная информация, относящаяся к объекту, описываемому этим фреймом.

Структуру фрейма можно представить так:

ИМЯ ФРЕЙМА:

(имя 1-го слота: значение 1-го слота),

(имя 2-го слота: значение 2-го слота),

- - - -

(имя N-го слота: значение N-го слота).

Модель фрейма является достаточно универсальной, поскольку позволяет отобразить все многообразие знаний о мире:

– через фреймы-структуры, для обозначения объектов и понятий (заем, залог, вексель);

– через фреймы-роли (менеджер, кассир, клиент);

– через фреймы-сценарии (банкротство, собрание акционеров, празднование именин);

– через фреймы-ситуации (тревога, авария, рабочий режим устройства) и др.

Значением слота может быть практически что угодно: числа, формулы, тексты на естественном языке или программы, правила вывода или ссылки на другие слоты данного фрейма или других фреймов. В качестве значения слота может выступать набор слотов более низкого уровня, что позволяет реализовывать во фреймовых представлениях «принцип матрешки». Связи между фреймами задаются значениями специального слота с именем «Связь».

В общем случае структура данных фрейма может содержать более широкий набор информации.

Пример

Рассмотрим пример фреймовой системы, описывающей некую аудиторию (рисунок 14).

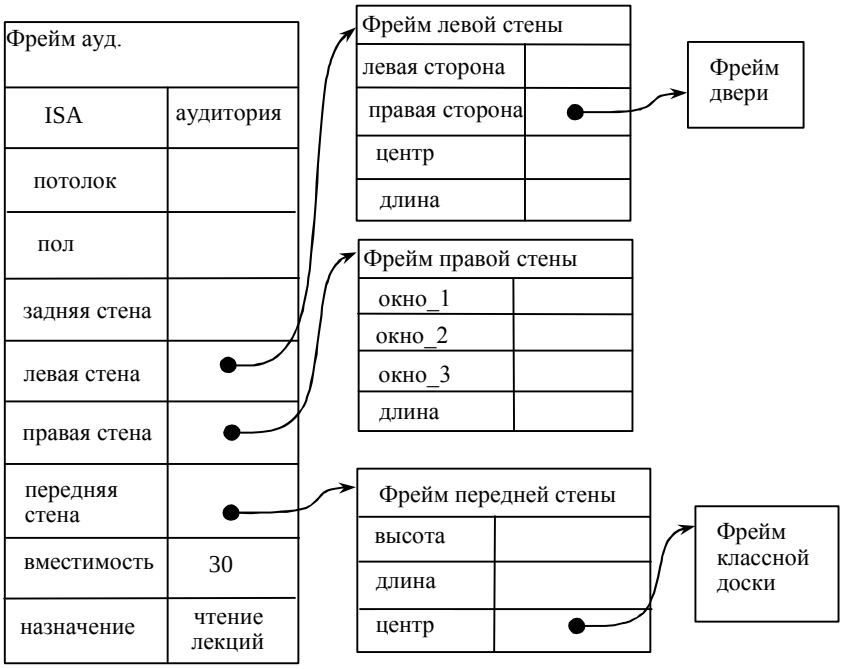

Рисунок 14 – Фрейм аудитории

Задание

Построить фреймы для следующих понятий:

1 Вариант 1. Понятие «студент».

2 Вариант 2. Понятие «профессор».

3 Вариант 3. Понятие «шкаф».

4 Вариант 4. Понятие «компьютер».

5 Вариант 5. Понятие «стол».

6 Вариант 6. Понятие «журнал».

7 Вариант 7. Понятие «книга».

8 Вариант 8. Понятие «ребенок».

Контрольные вопросы и задания

1 Что такое фрейм?

2 Назовите составные элементы фрейма.

3 Назовите виды фреймов.

4 От чего зависит глубина вложенности слотов во фрейме?

6 Практическое занятие №6. Нейронные сети в системах искусственного интеллекта. Аппроксимация функций нейронной сети

Цель занятия: изучение процесса построения и обучения нейронной сети для аппроксимации таблично заданной функции.

Краткие теоретические сведения

Нейронные сети – это одно из направлений исследований в области искусственного интеллекта, основанное на попытках воспроизвести нервную систему человека. А именно: способность нервной системы обучаться и исправлять ошибки, что должно позволить смоделировать, хотя и достаточно грубо, работу человеческого мозга.

Нейронные сети построены из простых элементов – нейронов. Математическая модель нейрона состоит из трех элементов – синапсов, сумматора и нелинейного преобразователя. В качестве нелинейных преобразователей используются, в частности, знаковая, пороговая, сигмоидальная и другие функции.

Синапсы осуществляют связь между нейронами. Они умножают входной сигнал на весовой коэффициент синапса, т.е. на число, характеризующее силу связи [8].

Нейроны, объединенные в определенную структуру, образуют нейронную сеть. Некоторые задачи могут решаться нейронной сетью, если она содержит несколько сот нейронов. Для решения сложных задач НС должна содержать десятки миллионов нейронов.

Существует большое количество моделей нейронных сетей. Среди них: многослойные персептроны, сети Хопфилда, Кохонена, Коско, Гроссберга, когнитрон, неокогнитрон, сеть СМАК и другие. Характеристики нейронов мало влияют на свойства нейронной сети. Основными факторами, определяющими свойства сети являются ее конфигурация или, как принято говорить, «топология» нейросети, и правила обучения.

С точки зрения топологии выделяют следующие основные типы моделей нейронных сетей: многослойные, полносвязные, с локальными связями.

У нейронных сетей много важных свойств, но ключевое из них – это способность к обучению. Обучение нейронной сети в первую очередь заключается в изменении «силы» синаптических связей между нейронами.

На сегодняшний день нейронные сети являются одним из приоритетных направлений исследований в области искусственного интеллекта [9].

Методические указания к работе

Выбор типа сети отображен на рисунке 15.

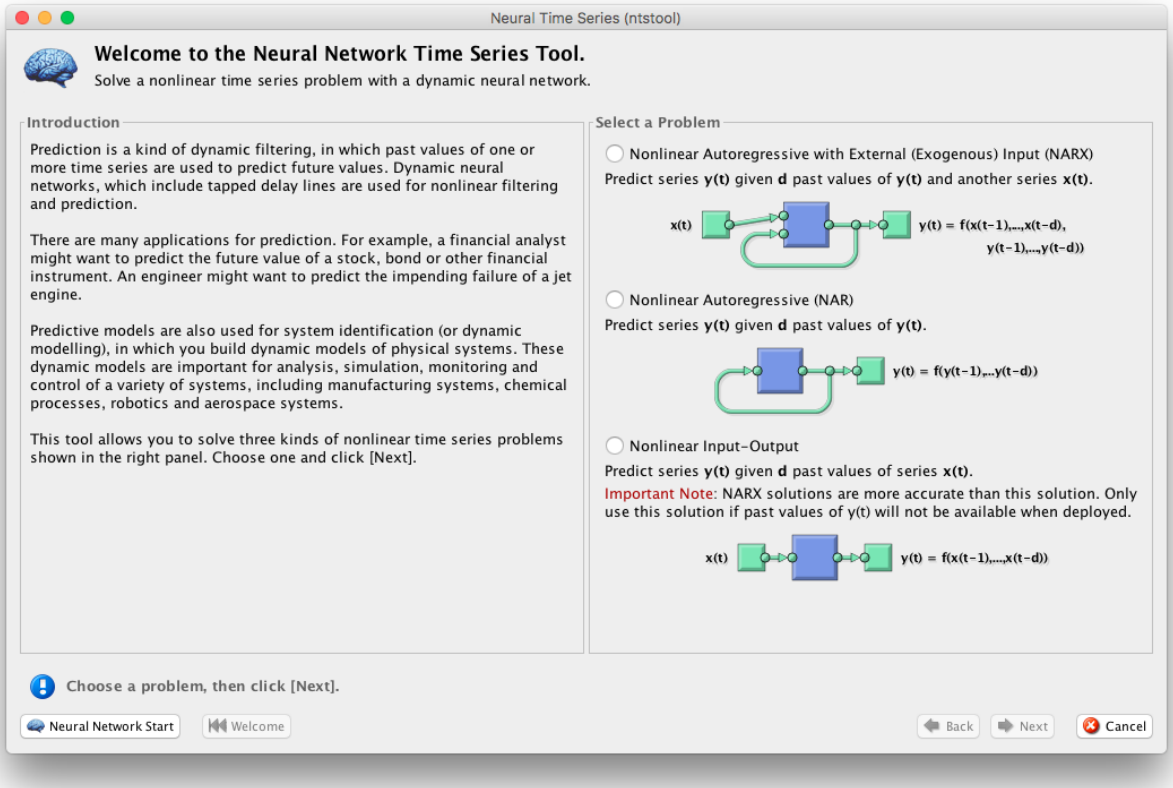

Рисунок 15 – Экран приветствия утилиты ntstool

На правой панели находится панель выбора типа нейронной сети:

– нелинейная авторегрессионная с внешним входом;
– нелинейная вход-выход (без обратных связей);
– нелинейная авторегрессионная (генератор).

Однако с помощью программного кода можно реализовать большее количество архитектур сети.

Пример создания нейронных сетей с помощью кода:

inputDelays = 1:10;

feedbackDelays = 1:10;

hiddenLayerSize = 20;

Нейронные сети без обратных связей

netx = narxnet(inputDelays,feedbackDelays,hiddenLayerSize);

net = narnet(feedbackDelays,hiddenLayerSize);

Нейронные сети с обратными связями

netxc = closeloop(netx);

netc = closeloop(net);

Для просмотра полученных сетей:

view(net).

Выбор данных для обучения, и их подготовка отображены в окне выбора параметром (рисунок 16).

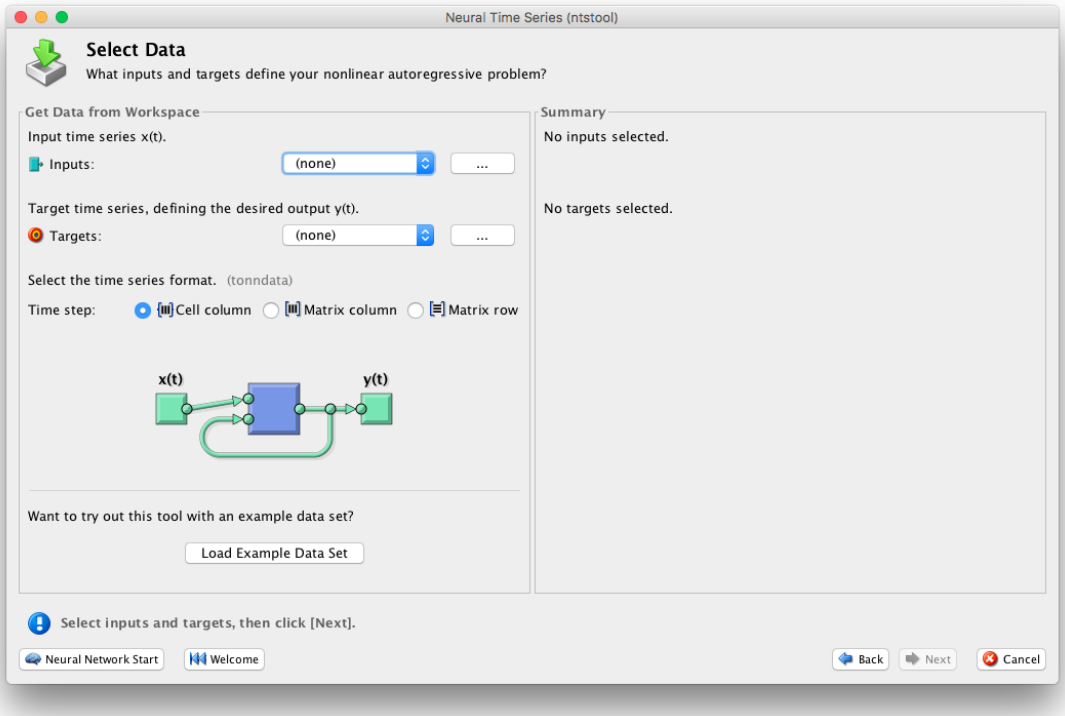

Рисунок 16 – Окно выбора параметров

В утилите ntstool выбор данных для обучения не составляет проблем, поэтому рассмотрим выбор параметров в коде.

Функция *preparets* подготавливает временные ряды для части сети, изменяя время до минимума, чтобы «чувствовать» входные данные и внутренние переменные. Использование *preparets* позволяет оставить исходные временные ряды неизменными, легко настраивая их для сети с разными размерами задержек как для открытых, так и закрытых нейронных сетей.

Синтаксис команды:

[Xs,Xi,Ai,Ts,EWs,shift] = preparets(net,Xnf,Tnf,Tf,EW).

Аргументы:

Net –Нейронная сеть;

Xnf – Не возвращаемые входные данные;

Tnf – Не возвращаемые выходные данные;

Tf – Выходные данные;

EW – Вес ошибок (По умолчанию = {1});

Возвращаемые значения:

Xs – Перемещенные входы;

Xi – Изначальные состояния входных задержек;

Ai – Изначальные состояния скрытых нейронов;

Ts – Перемещенные выходы;

EWs – Перемещенные веса ошибок;

Также, перед применением *preparets* может потребоваться использование функции *tonndata*, которая конвертирует данные в стандартный тип для нейронных сетей.

Синтаксис команды *tonndata:*

[y,wasMatrix] = tonndata(x,columnSamples,cellTime)/

Аргументы:

x – Матрица или ячейка – массив матриц;

columnSamples – True, если исходные данные представлены в виде колонок, false, если в виде строк;

cellTime – True, если исходные данные представлены в виде колоноками ячеек матрицы, false, если в виде матрицы.

Возвращаемые значения:

y – Исходные данные, конвертированные в стандартный вид;

wasMatrix – True, если исходные данные представлены в виде матриц.

Выбор назначения данных отображены на рисунке 17.

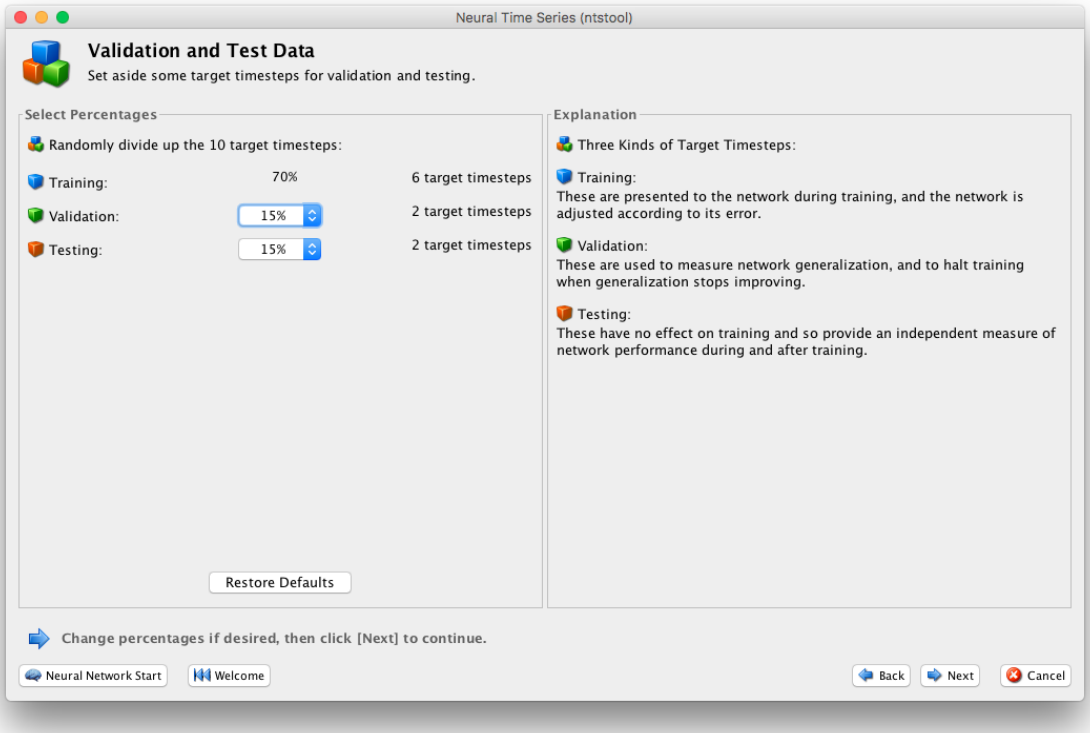

Рисунок 17 – Окно выбора процентного соотношения

Как и предыдущие шаги, работа с *ntstool* очень проста и требует простого выбора значений. В том случае, нейронная сеть проектируется с помощью программного кода, необходимо применять следующие команды:

net.divideParam.trainRatio = 85/100; %Тренировка;

net.divideParam.valRatio = 15/100; %Проверка;

net.divideParam.testRatio = 5/100; %Тестирование.

Последним шагом перед обучением сети в *ntstool* является выбор количества скрытых нейронов и количества нейронов задержки (рисунок 18).

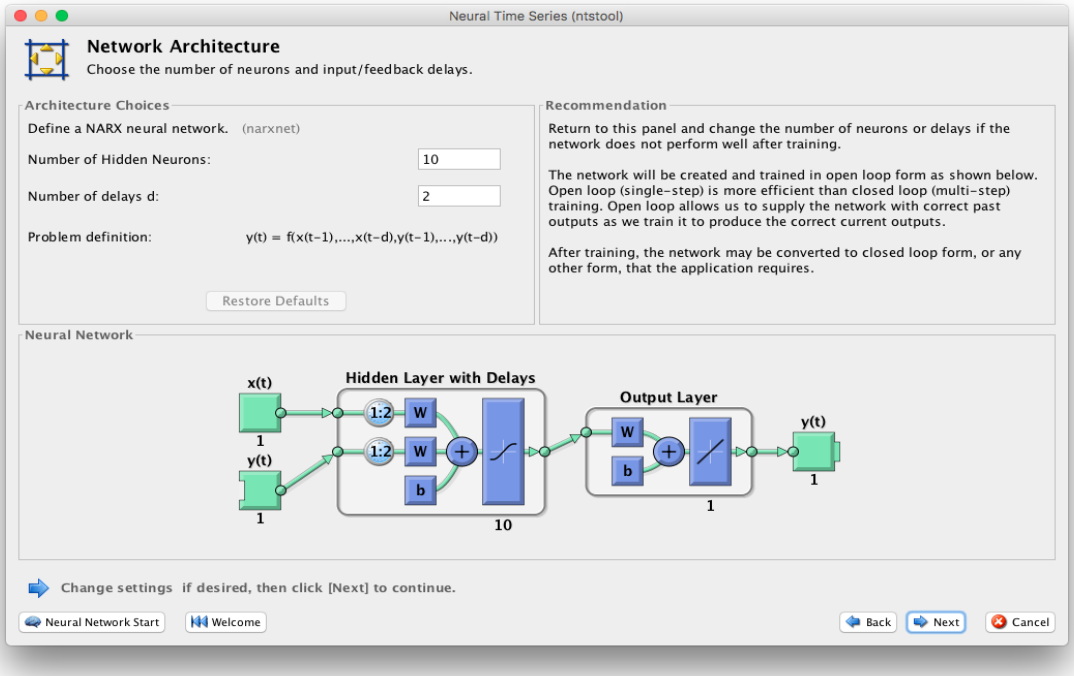

Рисунок 18 – Окно выбора размера сети

Заметим, что размеры сети при проектировании в программном коде определяются на первом шаге.

Обучение нейросети отображено на рисунке 19.

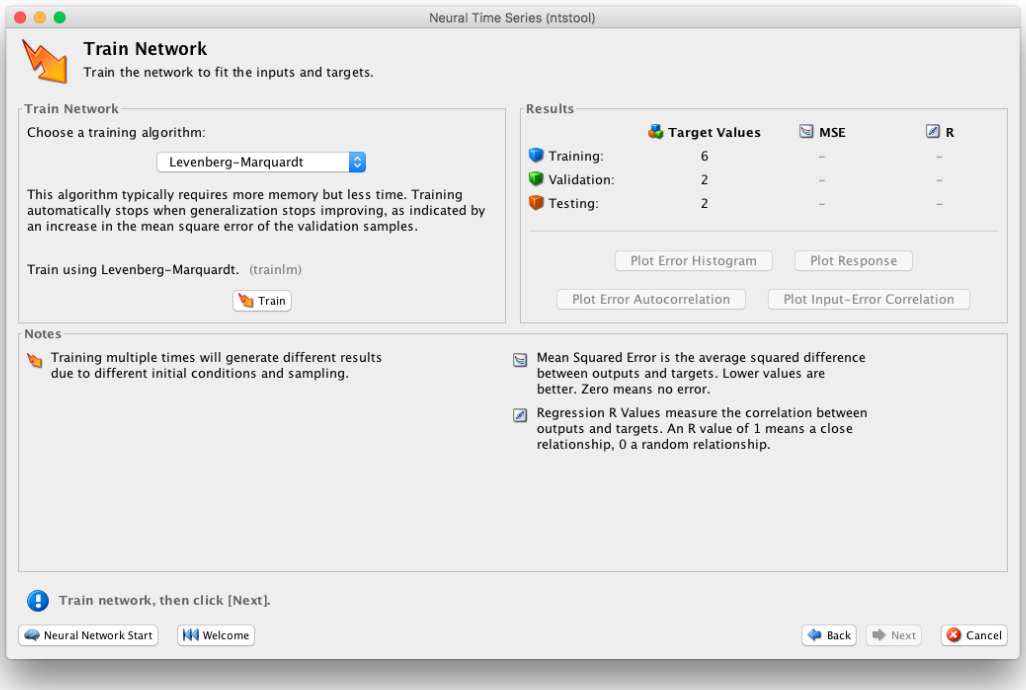

Рисунок 19 – Окно обучения нейронной сети

Операция обучения кодом выглядит так:

[net,tr] = train(net,inputs,targets,inputStates,layerStates)/

Для обучения сети в обоих случаях используется nntraintool, окно которого отображено на рисунке 20.

Рисунок 20 – Окно nntraintool

При помощи которого можно получить различные графики, характеризующие процесс и качество обучения сети. Графики также можно вызывать с помощью команд:

outputs = net(inputs,inputStates,layerStates);

errors = gsubtract(targets,outputs);

performance = perform(net,targets,outputs)

Для более подробной информации о дополнительных возможностях train, воспользуйтесь официальным руководством (возможностей там много).

Для работы с обученной сетью необходимо выполнить экспорт полученной нейросети (рисунок 21).

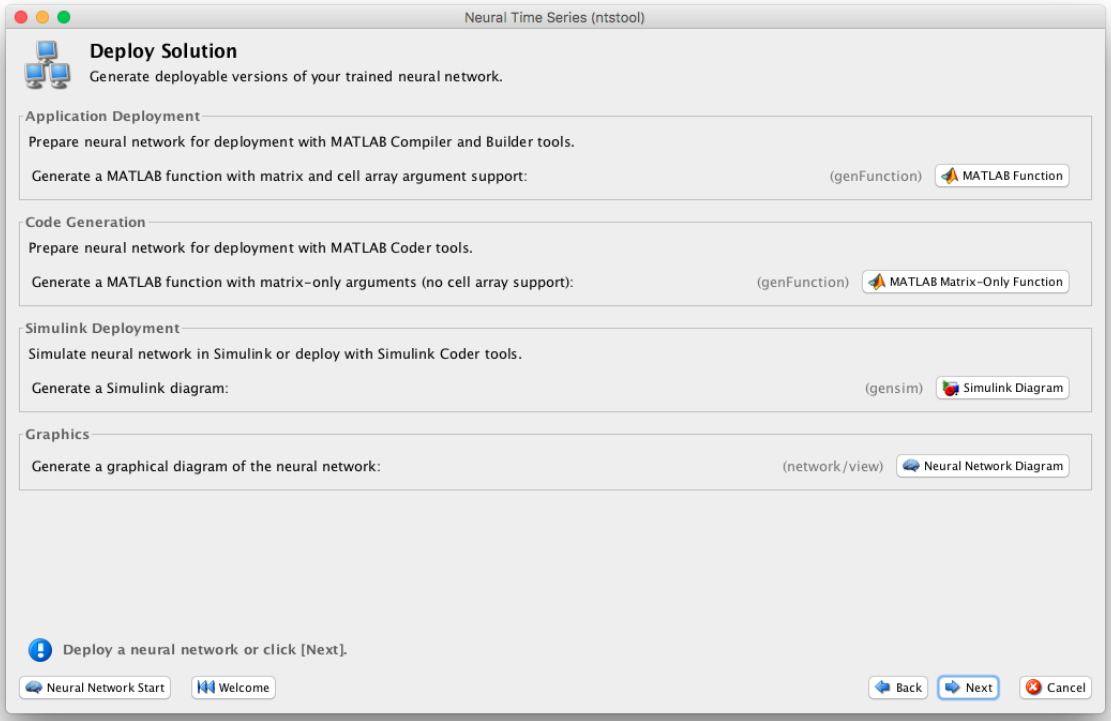

Рисунок 21 – Окно экспорта нейронной сети

Главный вопрос, который встает на этом шаге: можно ли установить конкретное время дискретизации для нейронной сети. Ответ: да, можно, но не с использованием ntstool, где всегда будет шаг дискретизации 1 секунда.

Для получения Simulink модели после всех операций над нейронной сетью с использованием кода, необходимо вызвать функцию: gensim(net,time), где *time* – время дискретизации, а *net* – нейронная сеть.

После вызова будет получена модель в Simulink, с заданной частотой дискретизации.

Пример

Приведем пример создания нейронной сети со следующими данными (таблица 2).

Таблица 2 – Исходные данные

x_i	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
$f(x_i)$	2.86	2.21	2.96	3.27	3.58	3.76	3.93	3.67	3.90	3.64	4.09

График исходной функции отображен на рисунке 22.

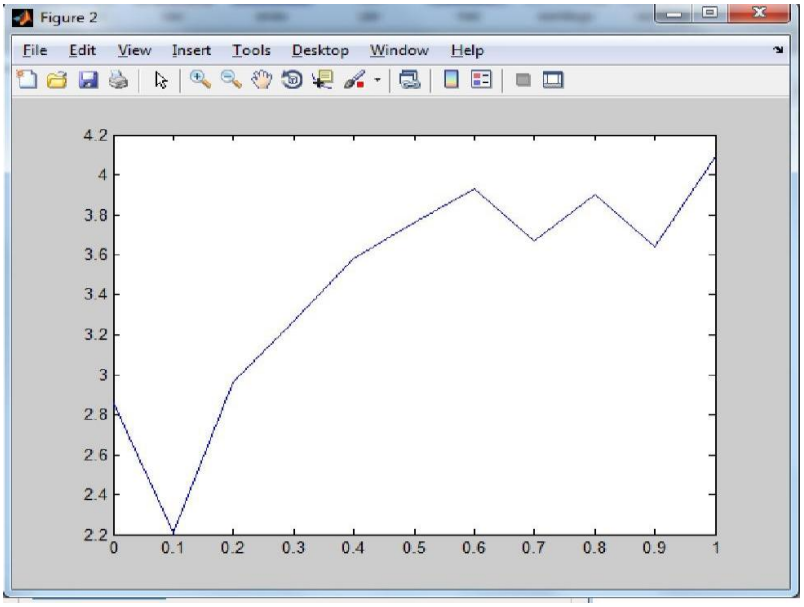

Рисунок 22 – График исходной функции

Создание и обучение нейронной сети:

```
x=[0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1];
f(x)=[2.86 2.21 2.96 3.27 3.58 3.76 3.93 3.67 3.90 3.64 4.09];
net=newff([0 3],[10,1],{'tansig','purelin'},'trainbfg');
net.trainParam.epochs=300;
net.trainParam.show=50;
net.trainParam.goal=1.37e-2;
[net,tr]=train(net,x,y);
an=sim(net,x);
plot(x,y,'+r',x,an,'-g');
hold on;
xx=[0.1850.86];
```

v=sim(net,xx);

plot(xx,v,'ob','MarkerSize',5,'LineWidth',2)

Процесс обуч,ения сети представлен на рисунке 23.

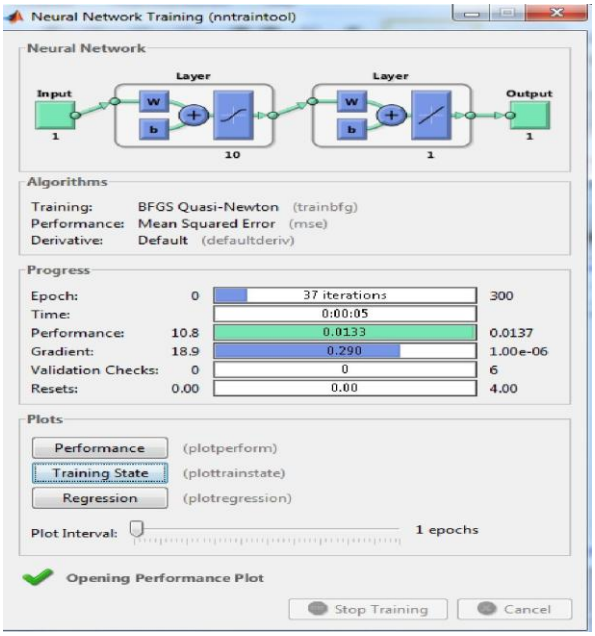

Рисунок 23 – Обучение сети

В процессе обучения сети получился график зависимости характеристики точности обучения сети от количества эпох (циклов), и вычисление средне-квадратичной ошибки сети составляет 0,013305 за 37 циклов (рисунок 24).

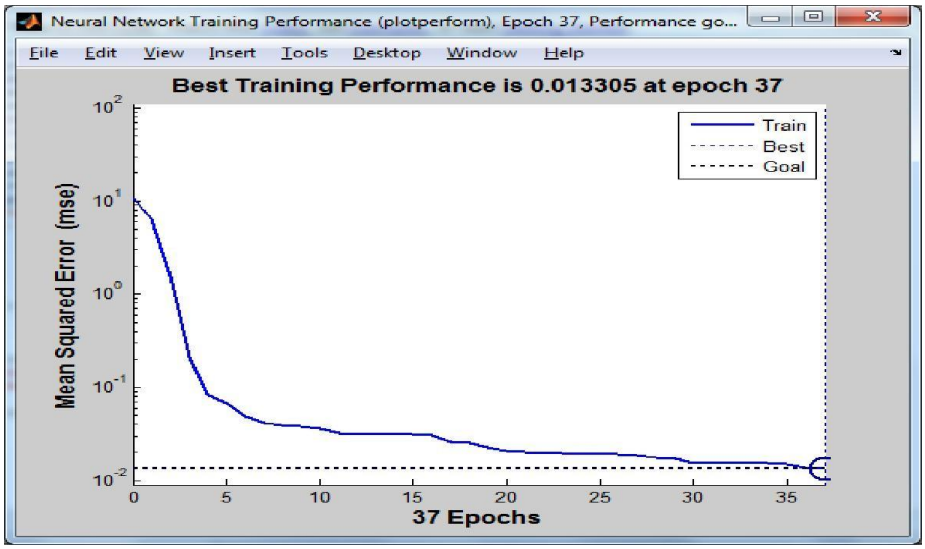

Рисунок 24 – Характеристика точности обучения в зависимости от количества эпох обучения

Далее необходимо сравнить графики исходной функции и аппроксимации (рисунок 25).

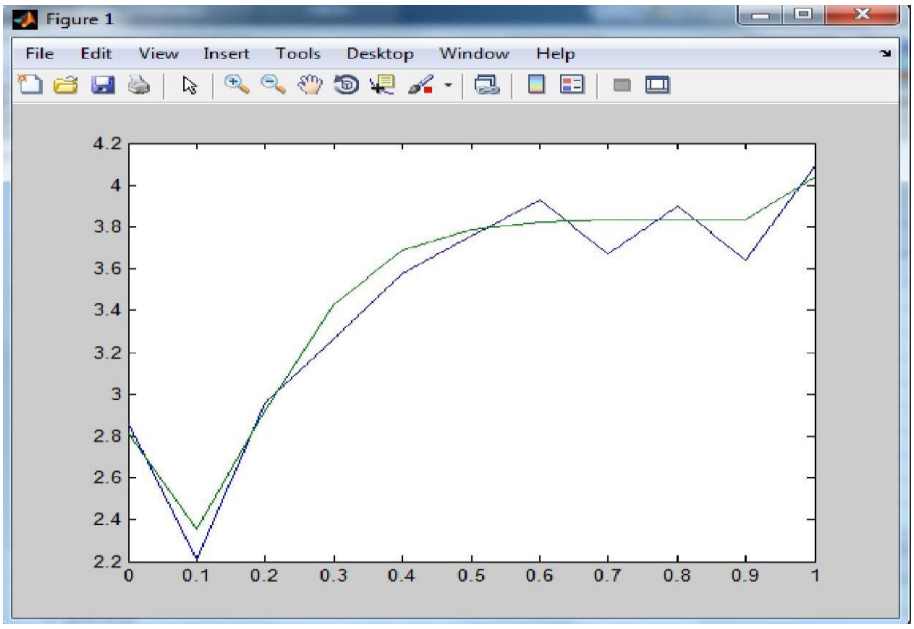

Рисунок 25 – Сравнение графиков исходной функции и аппроксимации

Аппроксимируем входящий набор точек методом наименьших квадратов. Результат для набора представлен на рисунке 26.

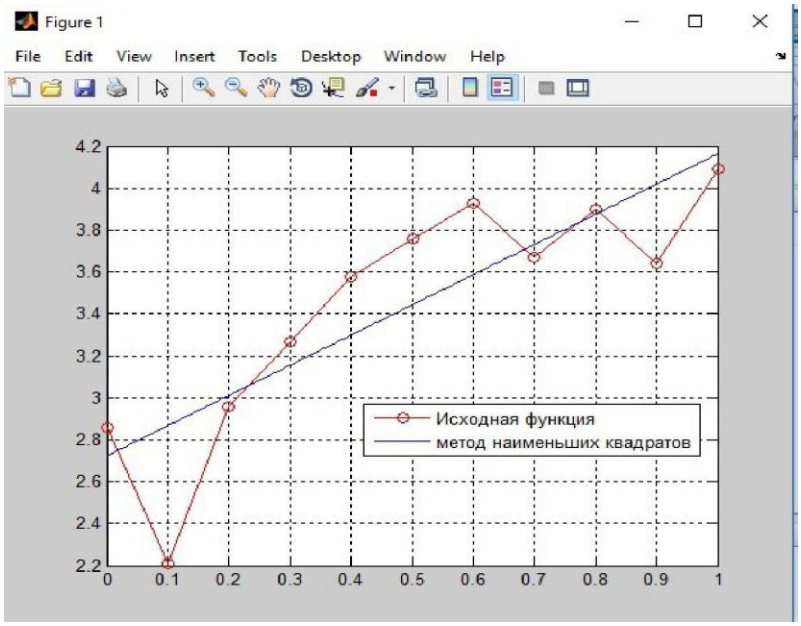

Рисунок 26 – Аппроксимация входящего набора точек методом наименьших квадратов

По полученным результатам аппроксимации заданного набора значений функции установлено, что нейронная сеть намного лучше аппроксимирует исходное значение функции, чем метод наименьших квадратов.

Задание

В среде MatLab необходимо построить и обучить нейронную сеть для аппроксимации таблично заданной функции.

Для этого необходимо создать таблицу экспериментальных данных:

x_i – задано вариантом;

$f(x_i)$ – вычисляется по следующей формуле: $x_i=a+ h_i$, i= 0,1, ,10, h = (b − a)/10 на отрезке [a,b].

x_i										
$f(x_i)$										

Варианты заданий:

1 Вариант 1: x=[0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1], a=2; b=12.

2 Вариант 2: x=[1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2], a=7; b=27.

3 Вариант 3: x=[0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2], a=2; b=12.

4 Вариант 4: x=[0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1], a=4; b=14.

5 Вариант 5: x=[0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5], a=6; b=21.

6 Вариант 6: x=[0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1], a=8; b=18.

7 Вариант 7: x=[1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2], a=9; b=29.

8 Вариант 8: x=[0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1], a=1; b=16.

9 Вариант 9: x=[0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2], a=6; b=21.

10 Вариант 10: x=[0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5], a=9; b=29.

Контрольные вопросы и задания

1 Что такое нейронные сети?

2 Назовите составные элементы нейронных сетей.

3 Назовите основные факторы, определяющие свойства нейронной сети.

4 Опишите ключевое свойство нейронных сетей.

Список использованных источников

1 Волкова, В.Н. Информационные системы в экономике : учебник для академического бакалавриата / В.Н. Волкова, В.Н. Юрьев, С.В. Широкова, А.В. Логинова ; под редакцией В Н. Волковой, В.Н. Юрьева. – Москва : Издательство Юрайт, 2019. – 402 с.

2 Абдикеев, Н.М. Проектирование интеллектуальных систем в экономике [Электронный ресурс] – Режим доступа: www.dgybga.ru/.../proektirovanie-intellektualnih-sistem-v-ekonomike-uchebnikabdigeev...

3 Тарасян, В.С. Пакет Fuzzy Logic Toolbox for Matlab : учеб. пособие / В.С. Тарасян. – Екатеринбург : Изд-во УрГУПС, 2013. – 112 с.

4 Цуканова, Н.И. Теория и практика логического программирования на языке Visual Prolog 7: учебное пособие для вузов / Н.И. Цуканова, Т.А. Дмитриева. – Электрон. дан. – М. : Горячая линия-Телеком, 2013. – 232 с. – Режим доступа: http://e.lanbook.com/book/11847.

5 Интеллектуальные информационные системы и технологии : учебное пособие / Ю.Ю. Громов, О.Г. Иванова, В.В. Алексеев и др. – Тамбов : Изд-во ФГБОУ ВПО «ТГТУ», 2013. – 244 с.

6 Павлов, С.Н. Системы искусственного интеллекта : учеб. пособие. В 2-х частях / С.Н. Павлов. – Томск: Эль Контент, 2011. –176 с

7 Смагин, А.А. Интеллектуальные информационные системы : учебное пособие / А.А. Смагин, С.В. Липатова, А.С. Мельниченко. – Ульяновск : УлГУ, 2010. – 136 с.

8 Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы: Пер.с польск. И.Д. Рудинского. ЭБС «Лань» [Электронный ресурс] / Д. Рутковская, М. Пилиньский, Л. Рутковский. – Электрон. дан. – М. : Горячая линия-Телеком, 2013. – 384 с. – Режим доступа: http://e.lanbook.com/book/.

9 Замятин, Н.В. Нечеткая логика и нейронные сети: учебное пособие / Н.В. Замятин ; рец.: И.А. Ходашинский, С.Н. Ливенцов. – Томск: Эль Контент, 2014. – 146 с.