Дисциплина: **Интеллектуальные информационные системы**

специальность: **информатики**

курс, группа: **2 к 1 г**

форма обучения: **заочная**

**установочная сессия**

период проведения занятий, вид занятий, кол-во часов:

-**16 июня 2020 г.**: Консультация, 4 часа, оф-лайн.

преподаватель: **Астахов В.К.**

электронная почта преподавателя: **vadast@mail.ru**

**Литература:**

**Основная:**

1. Барский А.Б. Логические нейронные сети [Электронный ресурс] / А.Б. Барский. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 492 c. — 978-5-94774-646-4. — Режим доступа: http://www.iprbookshop.ru/52220.html
2. Кухаренко Б.Г. Интеллектуальные системы и технологии [Электронный ресурс]: учебное пособие / Б.Г. Кухаренко. — Электрон. текстовые данные. — М.: Московская государственная академия водного транспорта, 2015. — 116 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/47933.html
3. Пальмов, С. В. Интеллектуальные системы и технологии [Электронный ресурс]: учебное пособие — Самара: Поволжский государственный университет телекоммуникаций и информатики, 2017. — 195 c.— Режим доступа: http://www.iprbookshop.ru/75375.html

**Дополнительная литература:**

1. Баженов Р.И. Интеллектуальные информационные технологии в управлении [Электронный ресурс]: учебное пособие / Р.И. Баженов. — Электрон. текстовые данные. — Саратов: Ай Пи Эр Медиа, 2018. — 117 c. — 978-5-4486-0102-6. — Режим доступа: http://www.iprbookshop.ru/72801.html
2. Козырева Г.Ф. Функциональное и логическое программирование [Электронный ресурс]: учебно-методическое пособие / Г.Ф. Козырева. — Электрон. текстовые данные. — Саратов: Ай Пи Эр Медиа, 2018. — 120 c. — 978-5-4486-0122-4. — Режим доступа: http://www.iprbookshop.ru/71596.html
3. Яхъяева Г.Э. Нечеткие множества и нейронные сети [Электронный ресурс]: учебное пособие. — Москва, Саратов: Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 320 c. — Режим доступа: http://www.iprbookshop.ru/67390.html

**Интернет- ресурсы**

1. Видео лекции... Интеллектуальные информационные системы [Электронный ресурс]: учебный видеокурс. — Электрон. дан. — Режим доступа: https://allyslide.com/ru/ viewer/\_intellektualynye\_informacionnye\_sistemy\_lekciya\_2\_\_145879
2. Воронцов К. В. Лекции по алгоритмам восстановления регрессии. http://www.machinelearning.ru/wiki/images/a/aa/Voron-ML-Regression.pdf –
3. Лекции по Интеллектуальным информационным системам [Электронный ресурс]: учебный видеокурс. — Электрон. дан. — Режим доступа: https:// mystudents.ru› study/lecture/iis
4. Научная электронная библиотека eLIBRARY.ru [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http://www.elibrary.ru/ defaultx.asp
5. Научная электронная библиотека IPRbooks.ru [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http://www. iprbooks.ru (по паролю)
6. Научная электронная онлайн-библиотека Порталус [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http://www. portalus.ru
7. Оn-line версия СПС «КонсультантПлюс» [Электронный ресурс]: офиц.сайт. — Электрон. дан. — Режим доступа: http://www.consultant.ru
8. Оn-line версия СПС «Гарант» [Электронный ресурс]: офиц.сайт. — Электрон. дан. — Режим доступа: http://www.garant.ru
9. Оn-line версия СПС «Кодекс» [Электронный ресурс]: офиц.сайт. — Электрон. дан. — Режим доступа: http://www. kodeks.ru
10. Передовые технологии анализа данных STATISTICA Data Mining. [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http:// www.statsoft.ru/products/STATISTICA\_Data\_Miner/ –
11. Профессиональный информационно - аналитический ресурс, посвященный машинному обучению, распознаванию образов и ИАД интеллектуальному анализу данных. [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http:// http://www.machinelearning.ru –
12. Управление ИТ. Библиотека и форум. [Электронный ресурс]: раздел Информационные технологии. — Электрон. дан. — Режим доступа: http://[www.osp.ru/itsm](http://www.osp.ru/itsm/)

**Задания по темам и датам**

| **дата, время**  **занятия** | **вид**  **занятия** | **кол-во часов** | **вопросы для изучения и обсуждения** | **контрольные вопросы, задания** |
| --- | --- | --- | --- | --- |
| 16.06.2020  09.00-10.30 | Консультация 1,  оф-лайн | 2 | 1.Искусственный интеллект – основа новых информационных технологий и интеллектуальных информационных систем.  2. Генетические алгоритмы и нейронные сети | 1.Изучить теоретический материал, представленный ниже и рекомендованную литературу по вопросам занятия.  (присылать материалы преподавателю для проверки НЕ НУЖНО) |
| 16.06.2020  10.35-12.05 | Консультация 2,  оф-лайн | 2 | 1.Экспертные системы:  1.1 Технологии разработки экспертных систем  1.2 Способы представления и обработки знаний в интеллектуальных и экспертных системах | 1.Изучить теоретический материал, представленный ниже и рекомендованную литературу по вопросам занятия.  (присылать материалы преподавателю для проверки НЕ НУЖНО) |

**Рекомендации и требования к выполнению заданий**

Изучить теоретический материал. Для подготовки использовать рекомендованную литературу и теоретический материал, представленным ниже. Дополнительно отчитываться и присылать материал преподавателю не надо.

Краткий конспект лекции по дисциплине для консультаций:

<http://av.disus.ru/metodichka/1995885-1-yuyu-gromov-ivanova-alekseev-belyaev-shvec-eliseev-intellektualnie-informacionnie-sistemi-tehnologii>

**Материалы для Консультации 1**

**1.Искусственный интеллект – основа новых информационных технологий и интеллектуальных информационных систем**

Искусственный интеллект (ИИ) как наука существует около полувека. Первой интеллектуальной системой считается программа «Логик–Теоретик», предназначенная для доказательства теорем и исчисления высказываний. Её работа впервые была продемонстрирована 9 августа 1956 г. В создании программы участвовали такие известные учёные, как А. Ньюэлл, А. Тьюринг, К. Шеннон, Дж. Лоу, Г. Саймон и др. За прошедшее с тех пор время в области ИИ разработано великое множество компьютерных систем, которые принято называть интеллектуальными. Области их применения охватывают практически все сферы человеческой деятельности, связанные с обработкой информации.

На сегодняшний день не существует единого определения, которое однозначно описывает эту научную область. Академик Г.С. Поспелов в книге «Искусственный интеллект – основа новой информационной технологии» писал [7]: «под "искусственным интеллектом" понимается наука о том, как заставить машину делать то, что умеет делать умный человек». Среди многих точек зрения на область разработок искусственного интеллекта доминируют следующие три. Согласно первой исследования в области ИИ относятся к фундаментальным, в процессе которых разрабатываются новые модели и методы решения задач, традиционно считавшихся интеллектуальными и не поддававшихся ранее формализации и автоматизации. Согласно второй точке зрения это направление связано с новыми идеями решения задач на ЭВМ, с разработкой новых технологий программирования и с переходом к компьютерам не фон-неймановской архитектуры. Третья точка зрения, наиболее прагматическая, основана на том, что в результате исследований, проводимых в области ИИ, появляется множество прикладных систем, способных решать задачи, для которых ранее создаваемые системы были непригодны. По последней трактовке ИИ является экспериментальной научной дисциплиной, в которой роль эксперимента заключается в проверке и уточнении интеллектуальных систем, представляющих собой аппаратно-программные информационные комплексы.

**Основные направления исследований в области интеллектуальных информационных систем Интеллектуальные информационные**системы проникают во все сферы нашей жизни, поэтому трудно провести строгую классификацию направлений, по которым ведутся активные и многочисленные исследования в области ИИ. Рассмотрим кратко некоторые из них.

Разработка интеллектуальных информационных систем или систем, основанных на знаниях. Это одно из главных направлений ИИ.

Основной целью построения таких систем являются выявление, исследование и применение знаний высококвалифицированных экспертов для решения сложных задач, возникающих на практике. При построении систем, основанных на знаниях (СОЗ), используются знания, накопленные экспертами в виде конкретных правил решения тех или иных задач. Это направление преследует цель имитации человеческого искусства анализа неструктурированных и слабоструктурированных проблем [9]. В данной области исследований осуществляется разработка моделей представления, извлечения и структурирования знаний, а также изучаются проблемы создания баз знаний (БЗ), образующих ядро СОЗ. Частным случаем СОЗ являются экспертные системы (ЭС).

Разработка естественно-языковых интерфейсов и машинный перевод. Проблемы компьютерной лингвистики и машинного перевода разрабатываются в ИИ с 1950-х гг. Системы машинного перевода с одного естественного языка на другой обеспечивают быстроту и систематичность доступа к информации, оперативность и единообразие перевода больших потоков, как правило, научно-технических текстов [6]. Системы машинного перевода строятся как интеллектуальные системы, поскольку в их основе лежат БЗ в определённой предметной области и сложные модели, обеспечивающие дополнительную трансляцию «исходный язык оригинала – язык смысла – язык перевода».

|  |  |
| --- | --- |
|  |  |

Они базируются на структурно-логическом подходе, включающем последовательный анализ и синтез естественно-языковых сообщений.

Кроме того, в них осуществляется ассоциативный поиск аналогичных фрагментов текста и их переводов в специальных базах данных (БД).

Данное направление охватывает также исследования методов и разработку систем, обеспечивающих реализацию процесса общения человека с компьютером на естественном языке (так называемые системы ЕЯ-общения) [6].

Генерация и распознавание речи. Системы речевого общения создаются в целях повышения скорости ввода информации в ЭВМ, разгрузки зрения и рук, а также для реализации речевого общения на значительном расстоянии. В таких системах под текстом понимают фонемный текст (как слышится).

Обработка визуальной информации. В этом научном направлении решаются задачи обработки, анализа и синтеза изображений [6]. Задача обработки изображений связана с трансформированием графических образов, результатом которого являются новые изображения.

В задаче анализа исходные изображения преобразуются в данные другого типа, например в текстовые описания. При синтезе изображений на вход системы поступает алгоритм построения изображения, а выходными данными являются графические объекты (системы машинной графики).

Обучение и самообучение. Эта актуальная область ИИ включает модели, методы и алгоритмы, ориентированные на автоматическое накопление и формирование знаний с использованием процедур анализа и обобщения данных [4, 13]. К данному направлению относятся не так давно появившиеся системы добычи данных (Data Mining) и системы поиска закономерностей в компьютерных базах данных (Knowledge Discovery).

Распознавание образов. Это одно из самых ранних направлений ИИ, в котором распознавание объектов осуществляется на основании применения специального математического аппарата, обеспечивающего отнесение объектов к классам [7], а классы описываются совокупностями определённых значений признаков.

Игры и машинное творчество. Машинное творчество охватывает сочинение компьютерной музыки [5], стихов [6], интеллектуальные системы для изобретения новых объектов [2, 14]. Создание интеллектуальных компьютерных игр является одним из самых развитых коммерческих направлений в сфере разработки программного обеспечения. Кроме того, компьютерные игры предоставляют мощный арсенал разнообразных средств, используемых для обучения.

Программное обеспечение систем ИИ. Инструментальные средства для разработки интеллектуальных систем включают специальные языки программирования, ориентированные на обработку символьной информации (LISP, SMALLTALK, РЕФАЛ), языки логического программирования (PROLOG), языки представления знаний (OPS5, KRL, FRL), интегрированные программные среды, содержащие арсенал инструментальных средств для создания систем ИИ (КЕ, ARTS, GURU, G2), а также оболочки экспертных систем (BUILD, EMYCIN, EXSYS Professional, ЭКСПЕРТ), которые позволяют создавать прикладные ЭС, не прибегая к программированию [8, 11].

Новые архитектуры компьютеров. Это направление связано с созданием компьютеров не фон-неймановской архитектуры, ориентированных на обработку символьной информации. Известны удачные промышленные решения параллельных и векторных компьютеров [1, 8], однако в настоящее время они имеют весьма высокую стоимость, а также недостаточную совместимость с существующими вычислительными средствами.

Интеллектуальные роботы. Создание интеллектуальных роботов составляет конечную цель робототехники. В настоящее время в основном используются программируемые манипуляторы с жёсткой схемой управления, названные роботами первого поколения. Несмотря на очевидные успехи отдельных разработок, эра интеллектуальных автономных роботов пока не наступила. Основными сдерживающими факторами в разработке автономных роботов являются нерешённые проблемы в области интерпретации знаний, машинного зрения, адекватного хранения и обработки трёхмерной визуальной информации.

**2. Генетические алгоритмы и нейронные сети**

**Генетические алгоритмы**

Простой генетический алгоритм [23, 26] включает операцию случайной генерации начальной популяции хромосом и ряд операторов, обеспечивающих генерацию новых популяций на основе начальной.

Этими операторами являются репродукция, кроссинговер и мутация.

Репродукцией называется процесс копирования хромосом с учётом значений целевой функции, т.е. хромосомы с «лучшими» значениями целевой функции имеют большую вероятность попадания в следующую популяцию. Этот процесс является аналогией митозного деления клеток. Выбор клеток (хромосом) для репродукции проводится в соответствии принципом «выживания сильнейшего». Простейшим способом представления операции репродукции в алгоритмической форме является колесо рулетки, в котором каждая хромосома имеет поле, пропорциональное значению целевой функции.

Рассмотрим пример применения простого генетического алгоритма для максимизации функции f (x ) = x 2 на целочисленном интервале [0, 31] (пример взят из монографии В.М. Курейчика «Генетические алгоритмы» [7]).

Значения аргумента функции f (x ) = x 2, изменяющегося в интервале от 0 до 31, можно представить пятиразрядными двоичными числами. Первоначальная популяция, состоящая из четырёх строк пятиразрядных чисел, полученная с помощью процедуры генерации случайных чисел, приведена во втором столбце табл. 3.1. Значение целеАнализ начальной популяции на первом шаге простого генетического алгоритма хромосомы Номер Суммарная Среднее значение целевой функции Максимальное значение вой функции для каждой хромосомы определяется путём возведения в квадрат значения двоичного числа, кодирующего решение х. Претенденты для скрещивания (кроссинговера) могут выбираться из начальной популяции или после выполнения оператора репродукции.

Репродукция начального множества заключается в четырёхкратном вращении колеса рулетки (4 – мощность популяции), в результате чего состав исходной популяции может измениться (рис. 3.5). Вероятность выбора i-й хромосомы вычисляется по формуле где fi (x) – значение целевой функции i-й хромосомы в популяции;

sum f (x ) – суммарное значение целевой функции всех хромосом в популяции.

Ожидаемое число копий i-й хромосомы после оператора репродукции равно где n – число анализируемых хромосом.

Число копий хромосомы, переходящих в следующее поколение, определяют по формуле где f ср (x ) – среднее значение целевой функции.

**Результаты операций репродукции и кроссинговера хромосомы Номер Суммарное значение целевой функции sum f ( x) =**Среднее значение целевой функции f ср ( x ) = Максимальное значение целевой функции f (x) = Значение N для первой хромосомы будет равно 0,14 4 = 0,56 копий, для второй – 0,49 4 = 1,96 копий, для третьей – 0,06 4 = 0,24 и для четвёртой – 0,31 4 = 1,23. В результате репродукции в новой популяции (второй столбец в табл. 3.2) будут присутствовать по одной копии первой и четвёртой хромосомы и две копии второй, а третья хромосома будет исключена. Таким способом оператор репродукции отбирает лучших представителей популяции.

На шаге 2 с помощью колеса рулетки осуществляется выбор хромосом для кроссинговера. Поля колеса рулетки соответствуют нормированным значениям целевой функции. Указатель рулетки после остановки колеса определяет выбранную хромосому.

Следует заметить, что случайный механизм не гарантирует выбора лучших хромосом, т.е. иногда результатом выбора могут оказаться хромосомы с низкими значениями целевой функции.

После репродукции выполняется оператор кроссинговера, который может повторяться несколько раз. При этом каждый раз будет осуществляться выбор двух кандидатур из множества хромосом. Затем каждая пара хромосом (стрингов) пересекается. Место пересечения K выбирается случайным образом на интервале (1, L – 1), где L – длина хромосомы, определяемая количеством значащих цифр в её двоичном коде. В нашем случае L = 5. Две новые хромосомы создаются путём взаимного обмена всех значений после точки пересечения, т.е. между позициями (K + 1) и L. При выборе двух первых хромосом из популяции (см. табл. 3.1) и значения K = 4 до применения оператора кроссинговера имеем описание а после применения оператора кроссинговера получаем описание Аналогично были получены потомки от третьей и четвёртой хромосом.

Анализ полученных результатов (см. табл. 3.2) показывает, что после проведения одной генерации улучшились и среднее, и максимальное значение целевой функции по сравнению с начальной популяцией (см. табл. 3.1).

Согласно схеме простого генетического алгоритма на шаге 3 выполняется оператор мутации, который играет существенную роль в естественной генетике и эволюции, но менее значим в генетических алгоритмах. Обычно выбирают одну мутацию на 1000 бит. Оператор мутации относится к унарным операциям и реализуется в два этапа.

Этап 1. В хромосоме A = { a1, a 2, a3,..., a L 2, a L 1, a L } случайным образом определяют две позиции, например, 2 и L – 1.

Этап 2. Гены, соответствующие выбранным позициям, меняют местами и формируют новую хромосому A = { a1, a L1, a3,..., Если длина обрабатываемых последовательностей невелика, то в процессе мутации можно осуществить полный перебор возможных перестановок генов и найти комбинацию с максимальным значением целевой функции. При длине хромосомы L = 50 – 200 полный перебор вариантов становится затруднительным, поэтому здесь производится случайно-направленный поиск, который может быть реализован на основе простого генетического алгоритма. Рассмотрим этот механизм на исследуемой задаче.

Выберем третью хромосому из пятого столбца табл. 3.2 со значением целевой функции f (х) = 729 и применим операцию мутации к позициям 3 и 4:

У новой хромосомы 3' значение целевой функции равно (29)2 = 841. Сделаем ещё одну перестановку 4 и 5 генов в хромосоме 3':

хромосома 3': 11101 хромосома 3": 11110.

Значение целевой функции для хромосомы 3" равно 900, что соответствует квазиоптимальному решению задачи нахождения максимального значения функции f (x ) = x 2 на интервале [0,31].

В генетических алгоритмах и эволюционном программировании используют два основных механизма воспроизводства хромосом:

воспроизводство без мутаций, соответствующее митозу, результатом которого являются потомки – копии родителей;

воспроизводство потомков, имеющих большие отличия от родителей. Этот механизм соответствует половому размножению.

В генетических алгоритмах в основном используется механизм родительского воспроизводства [4] с рекомбинацией и мутацией, а в эволюционном программировании – механизм на основе мутации без рекомбинации.

В алгоритмических реализациях механизма воспроизводства хромосом следует придерживаться следующих правил.

**1. Выбор начальной популяции можно выполнять произвольным образом, например подбрасыванием монеты.**

**2. Репродукция осуществляется на основе моделирования движения колеса рулетки.**

**3. Оператор кроссинговера реализуется как взаимный обмен короткими фрагментами двоичных строк гомологичных хромосом.**

**4. Вероятность оператора кроссинговера принимается равной Р(СО) < 1.0.**

**5. Вероятность оператора мутации принимается равной Р(МО) > 0.001.**

Разновидности генетических алгоритмов. Генетический алгоритм Девиса [25] включает следующие шаги:

**1. Инициализация популяции хромосом.**

**2. Оценка каждой хромосомы в популяции.**

**3. Создание новых хромосом посредством изменения и скрещивания текущих хромосом (применение операторов мутации и кроссинговера).**

**4. Устранение хромосом из популяции для замены их новыми.**

**5. Оценка новых хромосом и включение их в популяцию.**

**6. Проверка условия исчерпания ресурса времени, отведённого на поиск оптимального решения (если время исчерпано, то работа алгоритма**завершается и производится возврат к наилучшей хромосоме, в противном случае – переход к шагу 3).

Холланд [14, 26] предложил для генетического алгоритма оператор инверсии, который реализуется по схеме:

**1. Стринг (хромосома) B = { b1, b2,..., bL } выбирается случайным образом из текущей популяции.**

**2. Из множества Y = {0, 1, 2,..., L + 1} случайным образом выбираются два числа y1 и у2 и определяются значения x1 = min{y1, y2 } и x2 =**max{y1, y 2 }.

**3. Из хромосомы В формируется новая хромосома путём инверсии (обратного порядка) сегмента, лежащего справа от позиции х1 и слева от**позиции х2 в хромосоме В. После применения оператора инверсии строка В примет вид B = { b1, bx1, bx 2 1, bx 2 2, bx1+1, bx 2,..., bL }.

Например, для строки В = {1, 2, 3, 4, 5, 6} при выборе у1 = 6 и у2 = 2 и соответственно x1 = 2, x2 = 6 результатом инверсии будет B = {1, 2, 5, 4, 3, 6}.

Операции кроссинговера и мутации, используемые в простом ГА, изменяют структуру хромосом, в том числе разрушают удачные фрагменты найденных решений, что уменьшает вероятность нахождения глобального оптимума. Для устранения этого недостатка в генетических алгоритмах используют схемы (схематы или шаблоны), представляющие собой фрагменты решений или хромосом, которые желательно сохранить в процессе эволюции. При использовании схем в генетическом алгоритме вводится новый алфавит {0,1,\*}, где \* интерпретируется как «имеет значение 1 или 0». Например:

схема (\*0000) соответствует двум стрингам {10000 и 00000};

схема (\*111\*) соответствует четырём строкам {01110, 11110, 01111, 11111};

схема (0\*1\*\*) может соответствовать восьми пятизначным стрингам.

В общем случае хромосома длиной L максимально может иметь 3L схем (шаблонов), но только 2L различных альтернативных стрингов.

Это следует из факта, что схеме (\*\*) в общем случае могут соответствовать 32 = 9 стрингов, а именно {\*\*,\*1,\*0,1\*, 0\*,00,01,1011}, и только 22 = 4 альтернативные строки {00,01,10,11}, т.е. одной и той же строке может соответствовать несколько схем.

Если в результате работы генетического алгоритма удалось найти схемы типа (11\*\*\*) и (\*\*111), то, применив к ним оператор кроссинговера, можно получить хромосому (11111), обладающую наилучшим значением целевой функции.

Схемы небольшой длины называются строительными блоками.

Размер строительных блоков заметно влияет на качество и скорость нахождения результата. Вид строительного блока выбирается с учётом специфики решаемой задачи, а их разрыв в генетических алгоритмах допускается только в исключительных случаях, определяемых пользователем. Например, в схеме (\*\*\*\*1) строительным блоком является элемент 1, а в схеме (10\*\*\*) – составной элемент 10.

При использовании большого числа строительных блоков генетические алгоритмы, основанные на случайной генерации популяций и хромосом, переходят в разряд беспорядочных.

Стационарные генетические алгоритмы отличаются от поколенческих тем, что у первых размер популяции является заданным постоянным параметром, который определяется пользователем, а у вторых размер популяции в последующих генерациях может увеличиваться или уменьшаться.

Процедура удаления лишних хромосом в стационарных и поколенческих генетических алгоритмах основана на эвристических правилах, примерами которых являются следующие:

случайное равновероятное удаление хромосом;

удаление хромосом, имеющих худшие значения целевой функции;

удаление хромосом на основе обратного значения целевой функции;

удаление хромосом на основе турнирной стратегии.

Следует иметь в виду, что использование в генетических алгоритмах тех или иных эвристик удаления хромосом может повлечь за собой негативные последствия. Например, удаление худших хромосом приводит к преждевременной утрате разнообразия и, как следствие, к попаданию целевой функции в локальный оптимум, а при наличии большого числа хромосом с плохими значениями целевой функции утрачивается направленность поиска, и он превращается в «слепой» поиск.

Фундаментальная теорема генетического алгоритма. Пусть в момент времени t в популяции S(t) содержится множество хромосом Sj, j = 1, 2,.., n, а схема H строится на основе алфавита V = {0,1,\*}. Тогда схема может быть определена на двоичной хромосоме длины L. Очевидно, что для алфавита мощности М существует (М + l)L схем и n2L схем, содержащихся в популяции размера n, поскольку стринг представляется двумя схемами.

Для количественной оценки схем введём две характеристики: порядок схемы О(Н) и определённая длина схемы L(H). Порядок схемы определяет число закреплённых позиций (в двоичном алфавите – число единиц и нулей), представленных в шаблоне. Определённая длина схемы – это расстояние между первой и последней числовой позицией стринга.

Предположим, что заданы шаг по времени t и m примеров схем Н, содержащихся в популяции S(t), которые определяют возможное число различных схем Н при заданном t, т.е. m = m{Н,t).

В процессе репродукции вероятность попадания хромосомы Si в значения целевой функции. За время t + 1 в популяции S(t) ожидается получить m (Н, t + 1) представителей схемы Н, которое вычисляется по формуле где f (H) – среднее значение целевой функции хромосом, представленных схемой H за время t.

Так как среднее значение целевой функции для всей популяции равно Из этой формулы можно сделать вывод о том, что увеличение количества частных схем определяется отношением среднего значения целевой функции схемы к среднему значению целевой функции популяции. Поэтому схема, для которой значение целевой функции f (H) выше fср (S), имеет большую вероятность копирования.

Правило Холланда: Схема со значением целевой функции выше среднего живёт и копируется, а схема со значением ниже среднего умирает.

Если предположить, что схема H является жизнеспособной, то f (H ) f ср (S ). Тогда значение целевой функции для схемы H можно выразить через среднее значение для всей популяции, например, следующим образом: (1+ c ) f ср (S ), где с – константа. Число представителей схемы в следующем поколении будет Если принять значение с постоянным во времени, то за период 0 < t < t\* можно вычислить количество представителей схемы H по формуле m (H, t ) = (1 + c )t m (H, 0), из которой следует, что репродукция может приводить к экспоненциальному увеличению (с > 0) или уменьшению (с < 0) числа схем.

Лемма. Если на некотором шаге генетического алгоритма Р1 есть вероятность того, что хромосома А порождает потомка, и Р2 есть вероятность, что А уничтожается, то ожидаемое число потомков хромосомы А равно Р1 / Р2 [26].

Вероятность выживания хромосомы А на шаге t после операции репродукции определяется по формуле PS (t ) = (1 P2 ) P2, где t = 1, 2,..., g; g – число шагов (генераций) генетического алгоритма. Значение вероятности выживания хромосомы изменяется после операций кроссинговера и мутации. Использование оператора кроссинговера может вызывать увеличение или уменьшение числа схем в популяции.

Если кроссинговер не применяется, то обмен между хромосомами отсутствует, поэтому поисковое пространство не увеличивается, и процесс затухает.

Вероятность выживания схемы после применения оператора кроссинговера определяется по формуле где О(Н) – порядок схемы; L – длина стринга.

Если оператор кроссинговера выполняется на основе случайного выбора с вероятностью Р(СО), то вероятность выживания схемы определяется по формуле где L(H) – определённая длина схемы.

Приведённое выражение свидетельствует о том, что вероятность выживания схемы уменьшается при возрастании Р(СО).

Вычислим число схем Н в новой генерации после операций репродукции и кроссинговера, допуская их взаимную независимость:

Из этого выражения следует, что число схем m (H, t + 1) зависит от значений целевой функции для схемы и для всей популяции, а также от длины схемы L(H).

Рассмотрим влияние мутации на выживание схем. Известно, что единственная хромосома выживает с вероятностью 1 – Р(МО), где Р(МО) – вероятность оператора мутации. Если учесть тот факт, что частная схема выживает в случаях, когда выживает каждая из L(H) закреплённых позиций схемы, то для малых величин Р(МO) new, заменить новым узлом прежний всписке, причём, если прежний узел был в списке CLOSED, перенести его в список OPEN.

**нейронные сети**

Нейронные сети представляют собой классический пример технологии, основанной на примерах. Нейронные сети – обобщённое название группы математических алгоритмов, обладающих способностью обучаться на примерах, «узнавая» впоследствии черты встреченных образцов и ситуаций. Благодаря этой способности нейронные сети используются при решении задач обработки сигналов и изображений, распознавания образов, а также для прогнозирования [10].

Нейронная сеть – это кибернетическая модель нервной системы, которая представляет собой совокупность большого числа сравнительно простых элементов – нейронов, топология соединения которых зависит от типа сети. Чтобы создать нейронную сеть для решения какой-либо конкретной задачи, следует выбрать способ соединения нейронов друг с другом и подобрать значения параметров межнейронных соединений.

Искусственная нейронная сеть (ИНС) – это упрощённая модель биологического мозга, точнее нервной ткани [2, 5, 9, 12]. Естественная нервная клетка (нейрон) состоит из тела (сомы), содержащего ядро, и отростков – дендритов, по которым в нейрон поступают входные сигналы. Один из отростков, ветвящийся на конце, служит для передачи выходных сигналов данного нейрона другим нервным клеткам. Он называется аксоном. Соединение аксона с дендритом другого нейрона называется синапсом. Нейрон возбуждается и передаёт сигнал через аксон, если число пришедших по дендритам возбуждающих сигналов больше, чем число тормозящих.

Сеть ИНС представляет собой совокупность простых вычислительных элементов – искусственных нейронов, каждый из которых обладает определённым количеством входов (дендритов) и единственным выходом (аксоном), разветвления которого подходят к синапсам, связывающим его с другими нейронами. На входы нейрона поступает информация извне или от других нейронов. Каждый нейрон характеризуется функцией преобразования входных сигналов в выходной (функция возбуждения нейрона). Нейроны в сети могут иметь одинаковые или разные функции возбуждения. Сигналы, поступающие на вход нейрона, неравнозначны в том смысле, что информация из одного источника может быть более важной, чем из другого. Приоритеты входов задаются с помощью вектора весовых коэффициентов, моделирующих синаптическую силу биологических нейронов.

Модель искусственного нейрона (рис. 2.1) представляет собой дискретно-непрерывный преобразователь информации. Информация, поступающая на вход нейрона, суммируется с учётом весовых коэффициентов wi, сигналов хi, i = 1,.., n, где n – размерность пространства входных сигналов. Потенциал нейрона определяется по формуле Рис. 2.1. Схема кибернетической модели нейрона Рис. 2.2. Функции переноса искусственных нейронов:

а – линейная; б – ступенчатая; в – сигмоидальная Взвешенная сумма поступивших сигналов (потенциал) преобразуется с помощью передаточной функции f (Р) в выходной сигнал нейрона Y, который передается другим нейронам сети, т.е. Y = f (Р). Вид передаточной (активационной) функции является важнейшей характеристикой нейрона. В общем случае эта функция может быть ступенчатой (пороговой), линейной или нелинейной (рис. 2.2). Пороговая функция пропускает информацию только в том случае, если алгебраическая сумма входных сигналов превышает некоторую постоянную величину Р\*, например:

Пороговая функция не обеспечивает достаточной гибкости ИНС при обучении. Если значение вычисленного потенциала не достигает заданного порога, то выходной сигнал не формируется, и нейрон «не срабатывает». Это приводит к снижению интенсивности выходного сигнала нейрона и, как следствие, к формированию невысокого значения потенциала взвешенных входов в следующем слое нейронов.

Линейная функция Y = kP дифференцируема и легко вычисляется, что в ряде случаев позволяет уменьшить ошибки выходных сигналов в сети, так как передаточная функция сети также является линейной.

Однако она не универсальна и не обеспечивает решения многих задач.

Определённым компромиссом между линейной и ступенчатой функциями является сигмоидальная функция переноса Y = 1 / 1 + e kp, которая удачно моделирует передаточную характеристику биологического нейрона (рис. 2.2, в) Коэффициент k определяет крутизну нелинейной функции: чем больше k, тем ближе сигмоидальная функция к пороговой; чем меньше k, тем она ближе к линейной. Подобно ступенчатой функции она позволяет выделять в пространстве признаков множества сложной формы, в том числе невыпуклые и несвязные. При этом сигмоидальная функция, в отличие от ступенчатой, не имеет разрывов. Она дифференцируема, как и линейная функция, и это качество можно использовать при поиске экстремума в пространстве параметров ИНС.

Тип функции переноса выбирается с учётом конкретной задачи, решаемой с применением нейронных сетей. Например, в задачах аппроксимации и классификации предпочтение отдают логистической (сигмоидальной) кривой. Нейронная сеть представляет собой совокупность искусственных нейронов, организованных слоями. При этом выходы нейронов одного слоя соединяются с входами нейронов другого.

В зависимости от топологии соединений нейронов ИНС подразделяются на одноуровневые и многоуровневые, с обратными связями и без них. Связи между слоями могут иметь различную структуру.

В однолинейных сетях каждый нейрон (узел) нижнего слоя связан с одним нейроном верхнего слоя. Если каждый нейрон нижнего слоя соединён с несколькими нейронами следующего слоя, то получается пирамидальная сеть. Воронкообразная схема соединений предполагает связь каждого узла верхнего слоя со всеми узлами нижнего уровня.

Существуют также древовидные и рекуррентные сети, содержащие обратные связи с произвольной структурой межнейронных соединений. Чтобы построить ИНС для решения конкретной задачи, нужно выбрать тип соединения нейронов, определить вид передаточных функций элементов и подобрать весовые коэффициенты межнейронных связей [1, 2, 5 – 7, 12].

При всём многообразии возможных конфигураций ИНС на практике получили распространение лишь некоторые из них. Классические модели нейронных сетей рассмотрены ниже.

Теоретические основы нейроматематики были заложены в начале 1940-х гг. Попытки построить машины, способные к разумному поведению, были в значительной мере вдохновлены идеями «отца кибернетики» Норберта Винера, который писал в своей знаменитой работе «Кибернетика или управление и связь в животном и машине», что все машины, претендующие на «разумность», должны обладать способностью преследовать определённые цели и приспосабливаться, т.е. обучаться. Идеи Винера были применены Дж. Маккалохом и У. Питтсом, которые разработали собственную теорию деятельности головного мозга [3], основанную на предположении, что функционирование компьютера и мозга сходно. К главным результатам их работы относятся следующие:

модель нейрона в виде простейшего процессорного элемента, который вычисляет значение переходной функции от скалярного произведения вектора входных сигналов и вектора весовых коэффициентов;

конструкция нейронной сети для выполнения логических и арифметических операций;

предположение о том, что нейронная сеть способна обучаться, распознавать образы, обобщать полученную информацию.

В формализме Дж. Маккалоха и У. Питтса нейроны имеют пороговую функцию перехода из состояния в состояние. Каждый нейрон в сети определяет взвешенную сумму состояний всех других нейронов и сравнивает её с порогом, чтобы определить своё собственное состояние.

Аппаратная реализация ИНС на основе пороговых элементов, оперирующих двоичными числами, оказалась чрезвычайно трудной из-за высокой стоимости электронных элементов в то время. Самые совершенные системы тогда содержали лишь сотни нейронов, в то время как нервная система муравья содержит более 20 тыс.

Серьёзное развитие нейрокибернетика получила в трудах американского нейрофизиолога Ф. Розенблата, который предложил свою модель нейронной сети в 1958 г. и продемонстрировал созданное на её основе электронное устройство, названное перцептроном [8]. Розенблат Ф. ввёл возможность модификации межнейронных связей, что сделало ИНС обучаемой. Первые перцептроны были способны распознавать некоторые буквы латинского алфавита. Впоследствии модель перцептрона была значительно усовершенствована, а наиболее удачным её применением стали задачи автоматической классификации.

Алгоритм обучения перцептрона включает следующие шаги.

**1. Системе предъявляется эталонный образ.**

**2. Если результат распознавания совпадает с заданным, весовые коэффициенты связей не изменяются.**

**3. Если ИНС неправильно распознаёт результат, то весовым коэффициентам даётся приращение в сторону повышения качества распознавания.**

Теоретический анализ перцептрона, проведённый М. Минским и С. Пейпертом [4], показал его ограниченные возможности, поскольку не всегда существует такая комбинация весовых коэффициентов, при которой заданное множество образов будет распознаваться правильно.

Причина этого недостатка состоит в том, что однослойный перцептрон реализует линейную поверхность, разделяющую пространство эталонов, вследствие чего происходит неверное распознавание образов в случаях, когда задача не является линейно сепарабельной. Для решения таких проблем предложены модели многослойных перцептронов, способные строить ломаную границу между распознаваемыми образами. Несмотря на то, что перцептрон Розенблата имел невысокие возможности обучения, разработка этой концепции привлекла внимание исследователей к проблеме ИНС и привела к созданию более «разумных» интеллектуальных систем.

Многослойные сети. В многослойных сетях устанавливаются связи только между нейронами соседних слоёв, как показано на рис. 2.3.

Каждый элемент может быть соединён модифицируемой связью с любым нейроном соседних слоёв, но между элементами одного слоя связей нет. Каждый нейрон может посылать выходной сигнал только в вышележащий слой и принимать входные сигналы только с нижерасположенного слоя. Входные сигналы подаются на нижний слой, а выходной вектор сигналов определяется путём последовательного вычисления уровней активности элементов каждого слоя (снизу вверх) с Рис. 2.3. Схема многослойного перцептрона использованием уже известных значений активности элементов предшествующих слоёв. При распознавании образов входной вектор соответствует набору признаков, а выходной – распознаваемым образам.

Скрытый слой (один или несколько) предназначен для отражения специфики знаний. В таких сетях обычно используются передаточные сигмоидальные функции.

Структура нейронной сети определяется типом, например 25–10–5, т.е. двадцать пять узлов находится в первом слое, десять – в скрытом и пять – в выходном. Определение числа скрытых слоёв и числа нейронов в каждом слое для конкретной задачи является неформальной проблемой, при решении которой можно использовать эвристическое правило: число нейронов в следующем слое в два раза меньше, чем в предыдущем [10, 14].

Выше отмечалось, что простой перцептрон с одним слоем обучаемых связей формирует границы областей решений в виде гиперплоскостей. Двухслойный перцептрон может выполнять операцию логического И над полупространствами, образованными гиперплоскостями первого слоя весов. Это позволяет формировать любые выпуклые области в пространстве входных сигналов. С помощью трёхслойного перцептрона, используя логическое ИЛИ для комбинирования выпуклых областей, можно получить области решений произвольной формы и сложности, в том числе невыпуклые и несвязные. То, что многослойные перцептроны с достаточным множеством внутренних нейроподобных элементов и соответствующей матрицей связей в принципе способны осуществлять любое отображение вход–выход, отмечали ещё М. Минский и С. Пейперт, однако они сомневались, что для таких процедур можно открыть мощный аналог процедуры обучения простого перцептрона. В настоящее время в результате возрождения интереса к многослойным сетям предложено несколько таких процедур. Одной из них является алгоритм обратного распространения ошибки, который будет рассмотрен ниже.

Рекуррентные сети. Они содержат обратные связи, благодаря которым становится возможным получение отличающихся значений выходов при одних и тех же входных данных. Наличие рекуррентных нейронов позволяет ИНС накапливать знания в процессе обучения.

Рекуррентные сети (рис. 2.4) являются развитием модели Хопфилда на основе применения новых алгоритмов обучения, исключающих попадание системы в локальные минимумы на поверхности энергетических состояний. Важной особенностью рекуррентных сетей является их способность предсказывать существование новых классов объектов.

Рис. 2.4. Схема рекуррентной нейронной сети Модель Хопфилда. Работы американского биофизика Дж. Хопфилда положили начало современному математическому моделированию нейронных вычислений [11]. Ему удалось привлечь к анализу нейросетевых моделей мощный математический аппарат статистической физики. В результате была сформулирована математическая модель ассоциативной памяти на нейронной сети с использованием правила Д. Хебба для модификации весовых коэффициентов. Это правило основано на простом предположении: если два нейрона возбуждаются вместе, то сила связи между ними возрастает; если они возбуждаются порознь, то сила связи между ними уменьшается.

Сеть Хопфилда строится с учётом следующих условий:

все элементы связаны со всеми;

wji = wij – прямые и обратные связи симметричны;

wii = 0 – диагональные элементы матрицы связей равны нулю, т.е. исключаются обратные связи с выхода на вход одного нейрона.

Для однослойной нейронной сети со связями типа «все ко всем»

характерна сходимость к одной из конечного множества равновесных точек, которые являются локальными минимумами функции энергии, отражающей структуру всех связей в сети. Введённая Хопфилдом функция вычислительной энергии нейронной сети описывает поведение сети через стремление к минимуму энергии, который соответствует заданному набору образов. В связи с этим сети Хопфилда могут выполнять функции ассоциативной памяти, обеспечивая сходимость к тому образу, в область притяжения которого попадает начальный паттерн (образец) активности нейронов сети.

Этот подход привлекателен тем, что нейронная сеть для конкретной задачи может быть запрограммирована без обучающих итераций.

Веса связей вычисляются на основе вида функции энергии, сконструированной для решаемой задачи.

Развитием модели Хопфилда является машина Больцмана, предложенная и исследованная Дж. Е. Хинтоном и Р. Земелом [5, 7, 12] для решения комбинаторных оптимизационных задач и задач искусственного интеллекта. В ней, как и в других моделях, нейрон имеет состояния (1,0), межнейронные связи представлены весовыми коэффициентами, а каждое состояние сети характеризуется определённым значением функции консенсуса (аналог функции энергии). Максимум функции консенсуса соответствует оптимальному решению задачи.

Сети Хопфилда получили применение на практике в основном как реализации подсистем более сложных систем. Они имеют определённые недостатки, ограничивающие возможности их применения:

предположение о симметрии связей между элементами, без которой нельзя ввести понятие энергии;

нейронная сеть – это устройство для запоминания и обработки информации, а не устройство минимизации энергии. Экономия энергии играет в этих процессах вспомогательную роль;

сети Хопфилда поддерживают множество лишних, неэффективных, иногда дублирующих друг друга связей. В реальных нервных системах такие связи не поддерживаются, так как их реализация требует определённых затрат. В биологических нервных системах происходит освобождение от лишних связей за счёт их структуризации. При этом вместо организации связей «всех ко всем» используется многослойная иерархическая система связей.

Самоорганизующиеся сети Т. Кохонена [15]. Идея сетей с самоорганизацией на основе конкуренции между нейронами базируется на применении специальных алгоритмов самообучения ИНС. Сети Кохонена обычно содержат один (выходной) слой обрабатывающих элементов с пороговой передаточной функцией. Число нейронов в выходном слое соответствует количеству распознаваемых классов. Настройка параметров межнейронных соединений проводится автоматически на основе меры близости вектора весовых коэффициентов настраиваемых связей к вектору входных сигналов в эвклидовом пространстве.

В конкурентной борьбе побеждает нейрон, имеющий значения весов, наиболее близкие к нормализованному вектору входных сигналов.

Кроме того, в самоорганизующихся сетях возможна классификация входных образцов (паттернов). На практике идея Кохонена обычно используется в комбинации с другими нейросетевыми парадигмами.

При построении модели ИНС прежде всего необходимо точно определить задачи, которые будут решаться с её помощью. В настоящее время нейросетевые технологии успешно применяются для прогнозирования, распознавания и обобщения.

Первым этапом построения нейросетевой модели является тщательный отбор входных данных, влияющих на ожидаемый результат.

Из исходной информации необходимо исключить все сведения, не относящиеся к исследуемой проблеме. В то же время следует располагать достаточным количеством примеров для обучения ИНС. Существует эмпирическое правило, которое устанавливает рекомендуемое соотношение X между количеством обучающих примеров, содержащих входные данные и правильные ответы, и числом соединений в нейронной сети: X < 10.

Для факторов, которые включаются в обучающую выборку, целесообразно предварительно оценить их значимость, проведя корреляционный и регрессионный анализ, и проанализировать диапазоны их возможных изменений.

На втором этапе осуществляется преобразование исходных данных с учётом характера и типа проблемы, отображаемой нейросетевой моделью, и выбираются способы представления информации. Эффективность нейросетевой модели повышается, если диапазоны изменения входных и выходных величин приведены к некоторому стандарту, например [0,1] или [–1,1].

Третий этап заключается в конструировании ИНС, т.е. в проектировании её архитектуры (число слоёв и число нейронов в каждом слое). Структура ИНС формируется до начала обучения, поэтому успешное решение этой проблемы во многом определяется опытом и искусством аналитика, проводящего исследования.

Четвёртый этап связан с обучением сети, которое может проводиться на основе конструктивного или деструктивного подхода. В соответствии с первым подходом обучение ИНС начинается на сети небольшого размера, который постепенно увеличивается до достижения требуемой точности по результатам тестирования. Деструктивный подход базируется на принципе «прореживания дерева», в соответствии с которым из сети с заведомо избыточным объёмом постепенно удаляют «лишние» нейроны и примыкающие к ним связи. Этот подход даёт возможность исследовать влияние удалённых связей на точность сети. Процесс обучения нейронной сети представляет собой уточнение значений весовых коэффициентов и для отдельных узлов на основе постепенного увеличения объёма входной и выходной информации.

Началу обучения должна предшествовать процедура выбора функции активации нейронов, учитывающая характер решаемой задачи. В частности, в трёхслойных перцептронах на нейронах скрытого слоя применяется в большинстве случаев логистическая функция, а тип передаточной функции нейронов выходного слоя определяется на основе анализа результатов вычислительных экспериментов на сети. Индикатором обучаемости ИНС может служить гистограмма значений межнейронных связей [13].

На пятом этапе проводится тестирование полученной модели ИНС на независимой выборке примеров.

Важнейшим свойством нейронных сетей является их способность к обучению, что делает нейросетевые модели незаменимыми при решении задач, для которых алгоритмизация является невозможной проблематичной или слишком трудоёмкой. Обучение нейронной сети заключается в изменении внутренних параметров модели таким образом, чтобы на выходе ИНС генерировался вектор значений, совпадающий с результатами примеров обучающей выборки. Изменение параметров нейросетевой модели может выполняться разными способами в соответствии с различными алгоритмами обучения. Парадигма обучения определяется доступностью необходимой информации. Выделяют три парадигмы:

обучение с учителем (контролируемое);

обучение без учителя (неконтролируемое);

смешанное обучение.

При обучении с учителем все примеры обучающей выборки содержат правильные ответы (выходы), соответствующие исходным данным (входам). В процессе контролируемого обучения синаптические веса настраиваются так, чтобы сеть порождала ответы, наиболее близкие к правильным.

Обучение без учителя используется, когда не для всех примеров обучающей выборки известны правильные ответы. В этом случае предпринимаются попытки определения внутренней структуры поступающих в сеть данных с целью распределить образцы по категориям (модели Кохонена).

При смешанном обучении часть весов определяется посредством обучения с учителем, а другая часть получается с помощью алгоритмов самообучения.

Обучение по примерам характеризуется тремя основными свойствами: ёмкостью, сложностью образцов и вычислительной сложностью. Ёмкость соответствует количеству образцов, которые может запомнить сеть. Сложность образцов определяет способности нейронной сети к обучению. В частности, при обучении ИНС могут возникать состояния «перетренировки», в которых сеть хорошо функционирует на примерах обучающей выборки, но не справляется с новыми примерами, утрачивая способность обучаться.

Рассмотрим известные правила обучения ИНС.

Правило коррекции по ошибке. Процесс обучения ИНС состоит в коррекции исходных значений весовых коэффициентов межнейронных связей, которые обычно задаются случайным образом. При вводе входных данных запоминаемого примера (стимула) появляется реакция, которая передаётся от одного слоя нейронов к другому, достигая последнего слоя, где вычисляется результат. Разность между известным значением результата и реакцией сети соответствует величине ошибки, которая может использоваться для корректировки весов межнейронных связей. Корректировка заключается в небольшом (обычно менее 1%) увеличении синаптического веса тех связей, которые усиливают правильные реакции, и уменьшении тех, которые способствуют ошибочным. Это простейшее правило контролируемого обучения (дельта-правило) используется в однослойных сетях с одним уровнем настраиваемых связей между множеством входов и множеством выходов. При этом на каждом k-м шаге для j-го нейрона вес i-й связи вычисляется по формуле w jik = w ji (k 1) + w jik, где w jik = jk x jik, jk = T jk R jk, T jk – известное (правильное) значение выхода j-го нейрона; R jk – рассчитанное значение выхода j-го нейрона; x jik – величина сигнала на i-м входе; – коэффициент скорости обучения.

Оптимальные значения весов межнейронных соединений можно определить путём минимизации среднеквадратичной ошибки с использованием детерминированных или псевдослучайных алгоритмов поиска экстремума в пространстве весовых коэффициентов. При этом возникает традиционная проблема оптимизации, связанная с попаданием в локальный минимум.

Правило Хебба [7]. Оно базируется на следующем нейрофизиологическом наблюдении: если нейроны по обе стороны синапса активизируются одновременно и регулярно, то сила их синаптической связи возрастает. При этом изменение веса каждой межнейронной связи зависит только от активности нейронов, образующих синапс. Это существенно упрощает реализацию алгоритмов обучения.

Обучение методом соревнования. В отличие от правила Хебба, где множество выходных нейронов может возбуждаться одновременно, в данном случае выходные нейроны соревнуются (конкурируют) между собой за активизацию. В процессе соревновательного обучения осуществляется модификация весов связей выигравшего нейрона и нейронов, расположенных в его окрестности («победитель забирает всё»).

Метод обратного распространения ошибки. Он является обобщением процедуры обучения простого перцептрона с использованием дельта-правила на многослойные сети [2, 6, 10]. В данном методе необходимо располагать обучающей выборкой, содержащей «правильные ответы», т.е. выборка должна включать множество пар образцов входных и выходных данных, между которыми нужно установить соответствие. Перед началом обучения межнейронным связям присваиваются небольшие случайные значения. Каждый шаг обучающей процедуры состоит из двух фаз. Во время первой фазы входные элементы сети устанавливаются в заданное состояние. Входные сигналы распространяются по сети, порождая некоторый выходной вектор. Для работы алгоритма требуется, чтобы характеристика вход–выход нейроподобных элементов была неубывающей и имела ограниченную производную. Обычно для этого используют сигмоидальные функции. Полученный выходной вектор сравнивается с требуемым (правильным).

Если они совпадают, то весовые коэффициенты связей не изменяются. В противном случае вычисляется разница между фактическими и требуемыми выходными значениями, которая передаётся последовательно от выходного слоя к входному. На основе этой информации проводится модификация связей в соответствии с обобщённым дельтаправилом, которое имеет вид: p w ji = jp yip, где изменение в силе связи wji для р-й обучающей пары p w ji пропорционально произведению сигнала ошибки j-го нейрона jp, получающего входной сигнал по этой связи, и выходного сигнала i-го нейрона yip, посылающего сигнал по этой связи. Определение сигнала ошибки является рекурсивным процессом, который начинается с выходных блоков. Для выходного блока сигнал ошибки jp = y j T jp R jp, где Tjp и Rjp – соответственно желаемое и действительное значения выходного сигнала j-го блока; y j – производная от выходного сигнала j-го блока. Сигнал ошибки для скрытого блока определяется рекурсивно через сигнал ошибки блоков, с которым соединён его выход, и веса этих связей равны jp = yi kp wkj. Для сигмоидальной функции y j = y j 1 y j, поk этому на интервале 0 < yj < 1 производная имеет максимальное значение в точке уj = 0,5, а в точках уj = 0 и уj = 1 обращается в ноль. Максимальные изменения весов соответствуют блокам (нейронам), которые ещё не выбрали своё состояние. Кроме того, при конечных значениях весовых коэффициентов выходные сигналы блоков не могут достигать значений 0 или 1. Поэтому за 0 обычно принимают значения yj < 0,1, а за 1 – значения yj > 0,9.

Модификация весов производится после предъявления каждой пары вход–выход. Однако если коэффициент, определяющий скорость обучения, мал, то можно показать, что обобщённое дельтаправило достаточно хорошо аппроксимирует минимизацию общей ошибки функционирования сети D методом градиентного спуска в пространстве весов. Общая ошибка функционирования сети определяется по формуле Обучение продолжается до тех пор, пока ошибка не уменьшится до заданной величины. Эмпирические результаты свидетельствуют о том, что при малых значениях система находит достаточно хороший минимум D. Один из основных недостатков алгоритма обратного распространения ошибки заключается в том, что во многих случаях для сходимости может потребоваться многократное (сотни раз) предъявление всей обучающей выборки. Повышения скорости обучения можно добиться, например, используя информацию о второй производной D или путём увеличения.

Алгоритм обратного распространения ошибки используется также для обучения сетей с обратными связями. При этом используется эквивалентность многослойной сети с прямыми связями и синхронной сети с обратными связями на ограниченном интервале времени (слой соответствует такту времени).

В настоящее время предложены алгоритмы обучения, более привлекательные в смысле биологической аналогии. Примером является алгоритм рециркуляции для сетей, в которых скрытые блоки соединены с входными. При обучении веса связей перестраиваются таким образом, чтобы минимизировать частоту смены активности каждого блока. Таким образом, обученная сеть имеет стабильные состояния и может функционировать в режиме ассоциативной памяти.

**Способы реализации нейронных сетей**

Нейронные сети могут быть реализованы программным или аппаратным способом.Вариантами аппаратной реализации являются нейрокомпьютеры, нейроплаты и нейроБИС (большие интегральные схемы). Одна из самых простых и дешёвых нейроБИС – модель MD 1220 фирмы Micro Devices, которая реализует сеть с 8 нейронами и 120 синапсами.

Среди перспективных разработок можно выделить модели фирмы Adaptive Solutions (США) и Hitachi (Япония). Разрабатываемая фирмой Adaptive Solutions нейроБИС является одной из самых быстродействующих: объявленная скорость обработки составляет 1,2 млрд. межнейронных соединений в секунду (мнс/с). Схемы, производимые фирмой Hitachi, позволяют реализовывать ИНС, содержащие до 576 нейронов.

Большинство современных нейрокомпьютеров представляют собой персональный компьютер или рабочую станцию, в состав которых входит дополнительная нейроплата. К их числу относятся, например, компьютеры серии FMR фирмы Fujitsu. Возможностей таких систем вполне хватает для решения большого числа прикладных задач методами нейроматематики, а также для разработки новых алгоритмов.

Наибольший интерес представляют специализированные нейрокомпьютеры, в которых реализованы принципы архитектуры нейросетей.

Типичными представителями таких систем являются компьютеры семейства Mark фирмы TRW (первая реализация перцептрона, разработанная Ф. Розенблатом, называлась Mark I). Модель Mark III фирмы TRW представляет собой рабочую станцию, содержащую до 15 процессоров семейства Motorola 68000 с математическими сопроцессорами. Все процессоры объединены шиной VME. Архитектура системы, поддерживающая до 65 000 виртуальных процессорных элементов с более чем 1 млн. настраиваемых соединений, позволяет обрабатывать до 450 тыс. мнс/с.

Другим примером является нейрокомпьютер NETSIM, созданный фирмой Texas Instruments на базе разработок Кембриджского университета. Его топология представляет собой трёхмерную решётку стандартных вычислительных узлов на базе процессоров 80188. Компьютер NETSIM используется для моделирования сетей Хопфилда– Кохонена. Его производительность достигает 450 млн. мнс/с.

В тех случаях, когда разработка или внедрение аппаратных реализаций нейронных сетей обходятся слишком дорого, применяют более дешёвые программные реализации. Одним из самых распространённых программных продуктов является семейство программ Brain Maker фирмы CSS (California Scientific Software). Первоначально разработанный фирмой Loral Space Systems no заказу NASA и Johnson’s Space Center пакет Brain Maker был вскоре адаптирован для коммерческих приложений и сегодня используется несколькими тысячами финансовых и промышленных компаний, а также оборонными ведомствами США для решения задач прогнозирования, оптимизации и моделирования ситуаций. Назначение пакета Brain Maker – решение задач, для которых пока не найдены формальные методы и алгоритмы, а входные данные неполны, зашумлены и противоречивы. К таким задачам относятся прогнозирование курсов валют и акций на биржах, моделирование кризисных ситуаций, распознавание образов и многие другие. Brain Maker решает поставленную задачу, используя математический аппарат теории нейронных сетей (более конкретно – сеть Хопфилда с обучением по методу обратного распространения ошибки).

|  |  |  |
| --- | --- | --- |
|  |  |  |

В оперативной памяти строится модель многослойной нейронной сети, которая обладает свойством обучаться на множестве примеров, оптимизируя свою внутреннюю структуру. При правильном выборе структуры сети после её обучения на достаточно большом количестве примеров можно добиться высокой достоверности результатов (97% и выше). Существуют версии Brain Maker для MS DOS и MS Windows, а также для Apple Macintosh. Кроме базовой версии пакета в семейство Brain Maker входят следующие дополнения:

Brain Maker Student – версия пакета для университетов. Она особенно популярна у небольших фирм, специализирующихся на создании приложений для не очень сложных задач.

Toolkit Option – набор из трёх дополнительных программ, увеличивающих возможности Brain Maker. Binary, которая переводит обучающую информацию в двоичный формат для ускорения обучения;

Hypersonic Training, где используется высокоскоростной алгоритм обучения; Plotting, которая отображает факты, статистику и другие данные в графическом виде.

Brain Maker Professional – профессиональная версия пакета Brain Maker с расширенными функциональными возможностями включает в себя все опции Toolkit.

Genetic Training Option (для Brain Maker Pro) – программа автоматической оптимизации нейронной сети для решения заданного класса задач, использующая генетические алгоритмы для селекции наилучших решений.

Data Maker Editor – специализированный редактор для автоматизации подготовки данных при настройке и использовании нейронной сети.

Training Financial Data – специализированные наборы данных для настройки нейронной сети на различные виды аналитических, коммерческих и финансовых операций, которые включают реальные значения макроэкономических показателей NYSE, NADDAW, ASE, OEX, DOW и др., индексы инфляции, статистические данные биржевых сводок по различным видам продукции, а также информацию по фьючерсным контрактам и многое другое.

Brain Maker Accelerator – специализированная нейроплата– акселератор на базе сигнальных процессоров TMS320C25 фирмы Texas Instruments. Вставленная в персональный компьютер, она в несколько раз ускоряет работу пакета Brain Maker.

Brain Maker Accelerator Pro – профессиональная многопроцессорная нейронная плата. Она содержит пять сигнальных процессоров TMS320C30 и 32 Мбайт оперативной памяти.

В настоящее время на рынке программных средств имеется большое количество разнообразных пакетов для конструирования нейронных сетей и решения различных задач. Пакет Brain Maker можно назвать ветераном рынка. Кроме представителей этого семейства, к хорошо известным и распространённым программным средствам можно отнести Neuro Shell (Ward System's Group), Neural Works (Neural Ware Inc.) и Neuro Solutions (Neuro Dimension Inc.). Объектноориентированные программные среды семейства Neuro Solutions предназначены для моделирования ИНС произвольной структуры. Пользователю систем Neuro Solutions предоставлены возможности исследования и диалогового управления. Все данные в сети доступны для просмотра в процессе обучения посредством разнообразных инструментов визуализации. Проектирование ИНС в системе Neuro Solutions основано на модульном принципе, который позволяет моделировать стандартные и новые топологии. Важным преимуществом системы является наличие специальных инструментов, позволяющих моделировать динамические процессы в ИНС.

**Практическое применение нейросетевых технологий**

Применение нейросетевых технологий целесообразно при решении задач, имеющихследующие признаки:

отсутствие алгоритмов решения задач при наличии достаточно большого числа примеров;

наличие большого объёма входной информации, характеризующей исследуемую проблему;

зашумлённость, частичная противоречивость, неполнота или избыточность исходных данных.

Нейросетевые технологии нашли широкое применение в таких направлениях, как распознавание печатного текста, контроль качества продукции на производстве, идентификация событий в ускорителях частиц, разведка нефти, борьба с наркотиками, медицинские и военные приложения, управление и оптимизация, финансовый анализ, прогнозирование и др.

В сфере экономики нейросетевые технологии могут использоваться для классификации и анализа временных рядов путём аппроксимации сложных нелинейных функций. Экспериментально установлено, что модели нейронных сетей обеспечивают большую точность при выявлении нелинейных закономерностей на фондовом рынке по сравнению с регрессионными моделями [13].

Рассмотрим решение задачи прогнозирования цены закрытия на завтра по акциям некоторого предприятия X. Для моделирования воспользуемся данными наблюдений за месяц. В качестве исходных данных можно использовать индикаторы Dow Jones, NIKKEI, FTSE100, индексы и акции российских компаний, «сезонные» переменные и др.

|  |  |
| --- | --- |
|  |  |

Относительный показатель однодневной доходности предприятия можно определить из соотношений:

где Pi – оценка операции «вчера купил, сегодня продал»; Pi – оценка операции «вчера продал, сегодня купил»; Pi – значение выбранного показателя доходности в i-й день; Pi 1 – значение показателя в (i – 1)-й день.

Итоговая доходность за установленный интервал времени (n дней) рассчитывается по формуле Результаты оценки доходности предприятия за 30 дней с использованием различных моделей ИНС, а также доходов «идеального»

трейдера приведены ниже.

Стандартная трёхслойная сеть

Стандартная четырёхслойная сеть

Рекуррентная сеть с обратной отрицательной связью от скрытого слоя

Рекуррентная сеть с отрицательной обратной связью.......... 0, Сеть Ворда: с тремя скрытыми блоками, с разными передаточными функциями

Трёхслойная сеть с обходным соединением

Четырёхслойная сеть с обходными соединениями............... 0, Сеть с общей регрессией

Сеть метода группового учёта аргументов

Сеть Ворда: с тремя скрытыми блоками, с разными передаточными функциями, с обходным соединением..….. 0, «Идеальный» трейдер

«Идеальный трейдер» знает цену закрытия на следующий день и поэтому получает максимально возможную прибыль. Трейдер пользуется значением нейросетевого индикатора следующим образом: на основе прогнозируемого в (i – 1)-й день значения Pi, (величина относительно изменения цены закрытия по акциям рассматриваемого предприятия X на завтрашний i-й день) трейдер принимает решение о покупке ( Pi > 0) или продаже ( Pi < 0) акций.

Анализ результатов моделирования показывает, что лучшую доходность обеспечила рекуррентная сеть с отрицательной обратной связью (45% за 30 дней). Динамика изменения однодневных показателей доходности, полученных с помощью этой ИНС.

Нейросетевые технологии активно используются в маркетинге для моделирования поведения клиентов и распределения долей рынка.

Нейросетевые технологии позволяют отыскивать в маркетинговых базах данных скрытые закономерности.

Моделирование поведения клиентов позволяет определить характеристики людей, которые будут нужным образом реагировать на рекламу и совершать покупки определённого товара или услуги.

Сегментирование и моделирование рынков на основе нейросетевых технологий даёт возможность построения гибких классификационных систем, способных осуществлять сегментирование рынков с учётом многообразия факторов и особенностей каждого клиента.

Технологии ИНС имеют хорошие перспективы при решении задач имитации и предсказания поведенческих характеристик менеджеров и задач прогнозирования рисков при выдаче кредитов. Не менее актуально применение ИНС при выборе клиентов для ипотечного кредитования, предсказания банкротства клиентов банка, определения мошеннических сделок при использовании кредитных карточек, составления рейтингов клиентов при займах с фиксированными платежами и т.п.

Следует помнить о том, что применение нейросетевых технологий не всегда возможно и сопряжено с определёнными проблемами и недостатками.

**1. Необходимо как минимум 50, а лучше 100 наблюдений для создания приемлемой модели. Это достаточно большое число данных и они далеко не**всегда доступны. Например, при производстве сезонного товара истории предыдущих сезонов недостаточно для прогноза на текущий сезон из-за изменения стиля продукта политики продаж и т.д. Даже при прогнозировании спроса на достаточно стабильный продукт на основе информации о ежемесячных продажах трудно накопить исторические данные за период от 50 до 100 месяцев. Для сезонных товаров проблема ещё более сложна, так как каждый сезон фактически представляет собой одно наблюдение. При дефиците информации модели ИНС строят в условиях неполных данных, а затем проводят их последовательное уточнение.

**2. Построение нейронных сетей требует значительных затрат труда и времени для получения удовлетворительной модели. Необходимо учитывать,**что излишне высокая точность, полученная на обучающей выборке, может обернуться неустойчивостью результатов на тестовой выборке – в этом случае происходит «переобучение» сети.

Чем лучше система адаптирована к конкретным условиям, тем меньше она способна к обобщению и экстраполяции и тем скорее может оказаться неработоспособной при изменении этих условий. Расширение объёма обучающей выборки позволяет добиться большей устойчивости, но за счёт увеличения времени обучения.

**3. При обучении нейронных сетей могут возникать «ловушки», связанные с попаданием в локальные минимумы. Детерминированный алгоритм**обучения не в силах обнаружить глобальный экстремум или покинуть локальный минимум. Одним из приёмов, который позволяет обходить ловушки, является расширение размерности пространства весов за счёт увеличения числа нейронов скрытых слоёв. Некоторые возможности для решения этой проблемы открывают стохастические методы обучения. При модификации весов сети только на основе информации о направлении вектора градиента целевой функции в пространстве весов можно достичь локального минимума, но невозможно выйти из него, поскольку в точке экстремума «движущая сила» (градиент) обращается в нуль и причина движения исчезает. Чтобы покинуть локальный экстремум и перейти к поиску глобального, нужно создать дополнительную силу, которая будет зависеть не от градиента целевой функции, а от каких-то других факторов. Один из простейших методов состоит в том, чтобы просто создать случайную силу и добавить её к детерминистической.

**4. Сигмоидальный характер передаточной функции нейрона является причиной того, что если в процессе обучения несколько весовых коэффициентов**стали слишком большими, то нейрон попадает на горизонтальный участок функции в область насыщения. При этом изменения других весов, даже достаточно большие, практически не сказываются на величине выходного сигнала такого нейрона, а значит, и на величине целевой функции.

**5. Неудачный выбор диапазона входных переменных – достаточно элементарная, но часто совершаемая ошибка. Если хi – двоичная переменная со**значениями 0 и 1, то примерно в половине случаев она будет иметь нулевое значение: хi = 0. Поскольку хi входит в выражение для модификации веса в виде сомножителя, то эффект будет тот же, что и при насыщении: модификация соответствующих весов будет блокирована. Правильный диапазон для входных переменных должен быть симметричным, например от +1 до –1 [2, 12].

**6. Процесс решения задач нейронной сетью является «непрозрачным» для пользователя, что может вызывать с его стороны недоверие к прогнозирующим**способностям сети.

**7. Предсказывающая способность сети существенно снижается, если поступающие на вход факты (данные) имеют значительные отличия от примеров,**на которых обучалась сеть. Этот недостаток ярко проявляется при решении задач экономического прогнозирования, в частности, при определении тенденций котировок ценных бумаг и стоимости валют на фондовых и финансовых рынках.

**8. Отсутствуют теоретически обоснованные правила конструирования и эффективного обучения нейронных сетей. Этот недостаток приводит, в**частности, к потере нейронными сетями способности обобщать данные предметной области в состояниях переобучения (перетренировки).

**Материалы для Консультации 2**

**1.Экспертные системы**

**Основные типы интеллектуальных информационных систем и Интеллектуальная информационная система (ИИС) основана на концепции**использования базы знаний для генерации алгоритмов решения прикладных задач различных классов в зависимости от конкретных информационных потребностей пользователей.

Для ИИС характерны следующие признаки [12]:

**– развитые коммуникативные способности;**

**– умение решать сложные плохо формализуемые задачи;**

**– способность к самообучению;**

**– адаптивность.**

Каждому из перечисленных признаков условно соответствует свой класс ИИС. Различные системы могут обладать одним или несколькими признаками интеллектуальности с различной степенью проявления.

Средства ИИ могут использоваться для реализации различных функций, выполняемых ИИС. На рисунке 1.1 приведена классификация ИИС, признаками которой являются следующие интеллектуальные функции:

коммуникативные способности – способ взаимодействия конечного пользователя с системой;

решение сложных плохо формализуемых задач, которые требуют построения оригинального алгоритма решения в зависимости от конкретной ситуации, характеризующейся неопределённостью и динамичностью исходных данных и знаний;

Системы с интеллек- Экспертные системы Самообслуживающиеся Адаптивные Рис. 1.1. Классификация интеллектуальных информационных систем способность к самообучению – умение системы автоматически извлекать знания из накопленного опыта и применять их для решения задач;

адаптивность – способность системы к развитию в соответствии с объективными изменениями области знаний.

Системы с интеллектуальным интерфейсом. Применение ИИ для усиления коммуникативных способностей информационных систем привело к появлению систем с интеллектуальным интерфейсом, среди которых можно выделить следующие типы.

**1. Интеллектуальные базы данных. Позволяют в отличие от традиционных БД обеспечивать выборку необходимой информации, не присутствующей в**явном виде, а выводимой из совокупности хранимых данных.

**2. Естественно-языковой интерфейс. Применяется для доступа к интеллектуальным базам данных, контекстного поиска документальной текстовой**информации, голосового ввода команд в системах управления, машинного перевода с иностранных языков. Для реализации ЕЯ-интерфейса необходимо решить проблемы морфологического, синтаксического и семантического анализа, а также задачу синтеза высказываний на естественном языке. При морфологическом анализе осуществляются распознавание и проверка правильности написания слов в словаре.

Синтаксический контроль предполагает разложение входных сообщений на отдельные компоненты, проверку соответствия грамматическим правилам внутреннего представления знаний и выявление недостающих частей. Семантический анализ обеспечивает установление смысловой правильности синтаксических конструкций. В отличие от анализа синтез высказываний заключается в преобразовании цифрового представления информации в представление на естественном языке.

**3. Гипертекстовые системы. Используются для реализации поиска по ключевым словам в базах данных с текстовой информацией. Для более полного**отражения различных смысловых отношений терминов требуется сложная семантическая организация ключевых слов. Решение этих задач осуществляется с помощью интеллектуальных гипертекстовых систем, в которых механизм поиска сначала работает с базой знаний ключевых слов, а затем – с самим текстом. Аналогичным образом проводится поиск мультимедийной информации, включающей кроме текста графическую информацию, аудио- и видеообразы.

**4. Системы контекстной помощи. Относятся к классу систем распространения знаний. Такие системы являются, как правило, приложениями к**документации. Системы контекстной помощи – частный случай гипертекстовых и ЕЯ-систем. В них пользователь описывает проблему, а система на основе дополнительного диалога конкретизирует её и выполняет поиск относящихся к ситуации рекомендаций. В обычных гипертекстовых системах, наоборот, компьютерные приложения навязывают пользователю схему поиска требуемой информации.

**5. Системы когнитивной графики. Ориентированы на общение с пользователем ИИС посредством графических образов, которые генерируются в**соответствии с изменениями параметров моделируемых или наблюдаемых процессов. Когнитивная графика позволяет в наглядном и выразительном виде представить множество параметров, характеризующих изучаемое явление, освобождает пользователя от анализа тривиальных ситуаций, способствует быстрому освоению программных средств и повышению конкурентоспособности разрабатываемых ИИС. Применение когнитивной графики особенно актуально в системах мониторинга и оперативного управления, в обучающих и тренажёрных системах, в оперативных системах принятия решений, работающих в режиме реального времени.

Экспертные системы как самостоятельное направление в искусственном интеллекте сформировалось в конце 1970-х гг. История ЭС началась с сообщения японского комитета по разработке ЭВМ пятого поколения, в котором основное внимание уделялось развитию «интеллектуальных способностей» компьютеров с тем, чтобы они могли оперировать не только данными, но и знаниями, как это делают специалисты (эксперты) при выработке умозаключений. Группа по экспертным системам при Комитете British Computer Society определила ЭС как «воплощение в ЭВМ компоненты опыта эксперта, основанной на знаниях, в такой форме, что машина может дать интеллектуальный совет или принять решение относительно обрабатываемой функции». Одним из важных свойств ЭС является способность объяснить ход своих рассуждений понятным для пользователя образом [15].

Область исследования ЭС называют «инженерией знаний». Этот термин был введён Е. Фейгенбаумом и в его трактовке означает «привнесение принципов и инструментария из области искусственного интеллекта в решение трудных прикладных проблем, требующих знаний экспертов». Другими словами, ЭС применяются для решения неформализованных проблем, к которым относят задачи, обладающие одной (или несколькими) из следующих характеристик:

задачи не могут быть представлены в числовой форме;

исходные данные и знания о предметной области обладают неоднозначностью, неточностью, противоречивостью;

цели нельзя выразить с помощью чётко определённой целевой функции;

не существует однозначного алгоритмического решения задачи;

алгоритмическое решение существует, но его нельзя использовать по причине большой размерности пространства решений и ограничений на ресурсы (времени, памяти).

Главное отличие ЭС и систем искусственного интеллекта от систем обработки данных состоит в том, что в них используется символьный, а не числовой способ представления данных, а в качестве методов обработки информации применяются процедуры логического вывода и эвристического поиска решений.

ЭС охватывают самые разные предметные области, среди которых лидируют бизнес, производство, медицина, проектирование и системы управления [4, 6, 11, 12, 15, 17].

Во многих случаях ЭС являются инструментом, усиливающим интеллектуальные способности эксперта.

Для классификации ЭС используются следующие признаки:

способ формирования решения;

способ учёта временного признака;

вид используемых данных и знаний;

число используемых источников знаний.

По способу формирования решения ЭС можно разделить на анализирующие и синтезирующие. В системах первого типа осуществляется выбор решения из множества известных решений на основе анализа знаний, в системах второго типа решение синтезируется из отдельных фрагментов знаний.

В зависимости от способа учёта временного признака ЭС делят на статические и динамические. Статические ЭС предназначены для решения задач с неизменяемыми в процессе решения данными и знаниями, а динамические ЭС допускают такие изменения.

По видам используемых данных и знаний различают ЭС с детерминированными и неопределёнными знаниями. Под неопределённостью знаний и данных понимаются их неполнота, ненадёжность, нечёткость.

ЭС могут создаваться с использованием одного или нескольких источников знаний.

В соответствии с перечисленными признаками можно выделить четыре основных класса ЭС (рис. 1.2): классифицирующие, доопределяющие, трансформирующие и мультиагентные [12].

Классифицирующие ЭС решают задачи распознавания ситуаций.

Основным методом формирования решений в таких системах является дедуктивный логический вывод.

Доопределяющие ЭС используются для решения задач с не полностью определёнными данными и знаниями. В таких ЭС возникают задачи интерпретации нечётких знаний и выбора альтернативных направлений поиска в пространстве возможных решений. В качестве методов обработки неопределённых знаний могут использоваться байесовский вероятностный подход, коэффициенты уверенности, нечёткая логика.

Трансформирующие ЭС относятся к синтезирующим динамическим экспертным системам, в которых предполагается повторяющееся преобразование знаний в процессе решения задач. В ЭС данного класса используются различные способы обработки знаний:

генерация и проверка гипотез;

логика предположений и умолчаний (когда по неполным данным формируются представления об объектах определённого класса, которые впоследствии адаптируются к конкретным условиям изменяющихся ситуаций);

использование метазнаний (более общих закономерностей) для устранения неопределённостей в ситуациях.

Мультиагентные системы – это динамические ЭС, основанные на интеграции нескольких разнородных источников знаний. Эти источники обмениваются между собой получаемыми результатами в ходе решения задач. Системы данного класса имеют следующие возможности:

Неопределён- Доопреде- Мультиисточников Рис. 1.2. Основные классы экспертных систем реализация альтернативных рассуждений на основе использования различных источников знаний и механизма устранения противоречий;

распределенное решение проблем, декомпозируемых на параллельно решаемые подзадачи с самостоятельными источниками знаний;

применение различных стратегий вывода заключений в зависимости от типа решаемой проблемы;

обработка больших массивов информации из баз данных;

использование математических моделей и внешних процедур для имитации развития ситуаций.

Самообучающиеся интеллектуальные системы основаны на методах автоматической классификации ситуаций из реальной практики, или на методах обучения на примерах. Примеры реальных ситуаций составляют так называемую обучающую выборку, которая формируется в течение определённого исторического периода. Элементы обучающей выборки описываются множеством классификационных признаков.

Стратегия «обучение с учителем» предполагает задание специалистом для каждого примера значений признаков, показывающих его принадлежность к определённому классу ситуаций. При обучении «без учителя» система должна самостоятельно выделять классы ситуаций по степени близости значений классификационных признаков.

В процессе обучения проводится автоматическое построение обобщающих правил или функций, описывающих принадлежность ситуаций к классам, которыми система впоследствии будет пользоваться при интерпретации незнакомых ситуаций. Из обобщающих правил, в свою очередь, автоматически формируется база знаний, которая периодически корректируется по мере накопления информации об анализируемых ситуациях.

Построенные в соответствии с этими принципами самообучающиеся системы имеют следующие недостатки:

относительно низкую адекватность баз знаний возникающим реальным проблемам из-за неполноты и/или зашумлённости обучающей выборки;

низкую степень объяснимости полученных результатов;

поверхностное описание проблемной области и узкую направленность применения из-за ограничений в размерности признакового пространства.

Индуктивные системы позволяют обобщать примеры на основе принципа индукции «от частного к общему». Процедура обобщения сводится к классификации примеров по значимым признакам. Алгоритм классификации примеров включает следующие основные шаги.

1. Выбор классификационного признака из множества заданных.

2. Разбиение множества примеров на подмножества по значению выбранного признака.

3. Проверка принадлежности каждого подмножества примеров одному из классов.

4. Проверка окончания процесса классификации. Если какое-то подмножество примеров принадлежит одному подклассу, т.е. у всех примеров этого подмножества совпадает значение классификационного признака, то процесс классификации заканчивается.

5. Для подмножеств примеров с несовпадающими значениями классификационных признаков процесс распознавания продолжается, начиная с первого шага. При этом каждое подмножество примеров становится классифицируемым множеством.

Нейронные сети представляют собой классический пример технологии, основанной на примерах. Нейронные сети – обобщённое название группы математических алгоритмов, обладающих способностью обучаться на примерах, «узнавая» впоследствии черты встреченных образцов и ситуаций. Благодаря этой способности нейронные сети используются при решении задач обработки сигналов и изображений, распознавания образов, а также для прогнозирования [10].

Нейронная сеть – это кибернетическая модель нервной системы, которая представляет собой совокупность большого числа сравнительно простых элементов – нейронов, топология соединения которых зависит от типа сети. Чтобы создать нейронную сеть для решения какойлибо конкретной задачи, следует выбрать способ соединения нейронов друг с другом и подобрать значения параметров межнейронных соединений.

В системах, основанных на прецедентах, БЗ содержит описания конкретных ситуаций (прецеденты). Поиск решения осуществляется на основе аналогий и включает следующие этапы:

получение информации о текущей проблеме;

сопоставление полученной информации со значениями признаков прецедентов из базы знаний;

выбор прецедента из базы знаний, наиболее близкого к рассматриваемой проблеме;

адаптация выбранного прецедента к текущей проблеме;

проверка корректности каждого полученного решения;

занесение детальной информации о полученном решении в БЗ.

Прецеденты описываются множеством признаков, по которым строятся индексы быстрого поиска. Однако в системах, основанных на прецедентах, в отличие от индуктивных систем допускается нечёткий поиск с получением множества допустимых альтернатив, каждая из которых оценивается некоторым коэффициентом уверенности. Наиболее эффективные решения адаптируются к реальным ситуациям с помощью специальных алгоритмов.

Системы, основанные на прецедентах, применяются для распространения знаний и в системах контекстной помощи.

Информационные хранилища отличаются от интеллектуальных баз данных тем, что представляют собой хранилища значимой информации, регулярно извлекаемой из оперативных баз данных. Хранилище данных – это предметно-ориентированное, интегрированное, привязанное ко времени, неизменяемое собрание данных, применяемых для поддержки процессов принятия управленческих решений [3].

Предметная ориентация означает, что данные объединены в категории и хранятся в соответствии с теми областями, которые они описывают, а не с приложениями, которые их используют. В хранилище данные интегрируются в целях удовлетворения требований предприятия в целом, а не отдельной функции бизнеса. Привязанность данных ко времени выражает их «историчность», т.е. атрибут времени всегда явно присутствует в структурах хранилища данных. Неизменяемость означает, что, попав однажды в хранилище, данные уже не изменяются в отличие от оперативных систем, где данные присутствуют только в последней версии, поэтому постоянно меняются.

Технологии извлечения знаний из хранилищ данных основаны на методах статистического анализа и моделирования, ориентированных на поиск моделей и отношений, скрытых в совокупности данных. Эти модели могут в дальнейшем использоваться для оптимизации деятельности предприятия или фирмы.

Для извлечения значимой информации из хранилищ данных имеются специальные методы (OLAP-анализа, Data Mining или Knowledge Discovery), основанные на применении методов математической статистики, нейронных сетей, индуктивных методов построения деревьев решений и др.

Технология OLAP (On-Line Analytical Processing – оперативный анализ данных) предоставляет пользователю средства для формирования и проверки гипотез о свойствах данных или отношениях между ними на основе разнообразных запросов к базе данных. Они применяются на ранних стадиях процесса извлечения знаний, помогая аналитику сфокусировать внимание на важных переменных. Средства Data Mining отличаются от OLAP тем, что кроме проверки предполагаемых зависимостей они способны самостоятельно (без участия пользователя) генерировать гипотезы о закономерностях, существующих в данных, и строить модели, позволяющие количественно оценить степень взаимного влияния исследуемых факторов на основе имеющейся информации.

Потребность в адаптивных информационных системах возникает в тех случаях, когда поддерживаемые ими проблемные области постоянно развиваются. В связи с этим адаптивные системы должны удовлетворять ряду специфических требований, а именно:

адекватно отражать знания проблемной области в каждый момент времени;

быть пригодными для лёгкой и быстрой реконструкции при изменении проблемной среды.

Адаптивные свойства информационных систем обеспечиваются за счёт интеллектуализации их архитектуры. Ядром таких систем является постоянно развиваемая модель проблемной области, поддерживаемая в специальной базе знаний – репозитории. Ядро системы управляет процессами генерации или переконфигурирования программного обеспечения.

В процессе разработки адаптивных информационных систем применяется оригинальное или типовое проектирование. Оригинальное проектирование предполагает разработку информационной системы с «чистого листа» на основе сформулированных требований. Реализация этого подхода основана на использовании систем автоматизированного проектирования, или CASE-технологий (Designer 2000, Silver Run, Natural Light Storm и др.).

При типовом проектировании осуществляется адаптация типовых разработок к особенностям проблемной области. Для реализации этого подхода применяются инструментальные средства компонентного (сборочного) проектирования информационных систем (R/3, BAANIV, Prodis и др.).

Главное отличие подходов состоит в том, что при использовании CASE-технологии на основе репозитория при изменении проблемной области каждый раз выполняется генерация программного обеспечения, а при использовании сборочной технологии – конфигурирование программ и только в редких случаях – их переработка.

**1.1 Технологии разработки экспертных систем**

**Технологии разработки экспертных систем Технология создания интеллектуального программного обеспечения существенно отличается**от разработки традиционных программ с использованием известных алгоритмических языков (табл. 1.1).

**Отличия систем искусственного интеллекта** Рассмотрим отработанные на сегодняшний день элементы технологии создания ИИС на примере разработки экспертных систем. Этот выбор обусловлен тем, что ЭС получили весьма широкое распространение во многих сферах человеческой деятельности, а технологии их создания имеют универсальный характер и не требуют аппаратных реализаций.

Экспертными системами называют сложные программные комплексы, аккумулирующие знания специалистов в конкретных предметных областях и тиражирующие этот эмпирический опыт для консультаций менее квалифицированных пользователей [4].

В самых первых ЭС не учитывалось изменение знаний, используемых в процессе решения конкретной задачи. Их назвали статическими ЭС. Типичная статическая ЭС содержит следующие основные компоненты:

рабочую память, называемую также базой данных;

решатель (интерпретатор);

систему объяснений;

компоненты приобретения знаний;

интерфейс с пользователем.

База знаний ЭС предназначена для хранения долгосрочных данных, описывающих рассматриваемую область, и правил, описывающих целесообразные преобразования данных этой области.

База данных (рабочая память) служит для хранения текущих данных решаемой задачи.

Решатель (интерпретатор) формирует последовательность применения правил и осуществляет их обработку, используя данные из рабочей памяти и знания из БЗ.

Система объяснений показывает, каким образом система получила решение задачи и какие знания при этом использовались. Это облегчает тестирование системы и повышает доверие пользователя к полученному результату.

Компоненты приобретения знаний необходимы для заполнения ЭС знаниями в диалоге с пользователем-экспертом, а также для добавления и модификации заложенных в систему знаний.

К разработке ЭС привлекаются специалисты из разных предметных областей, а именно:

эксперты той проблемной области, к которой относятся задачи, решаемые ЭС;

инженеры по знаниям, являющиеся специалистами по разработке ИИС;

программисты, осуществляющие реализацию ЭС.

Эксперты поставляют знания в ЭС и оценивают правильность получаемых результатов. Инженеры по знаниям помогают экспертам выявить и структурировать знания, необходимые для работы ЭС, выполняют работу по представлению знаний, выбирают методы обработки знаний, проводят выбор инструментальных средств для реализации ЭС, наиболее пригодных для решения поставленных задач.

Программисты разрабатывают программное обеспечение ЭС и осуществляют его сопряжение со средой, в которой оно будет использоваться.

Любая ЭС должна иметь, по крайней мере, два режима работы.

В режиме приобретения знаний эксперт наполняет систему знаниями, которые впоследствии позволят ЭС самостоятельно (без помощи эксперта) решать определённые задачи из конкретной проблемной области. Эксперт описывает проблемную область в виде совокупности данных и правил. Данные определяют объекты, их характеристики и значения, существующие в области экспертизы. Правила определяют взаимные связи, существующие между данными, и способы манипулирования данными, характерные для рассматриваемого класса задач.

В режиме консультации пользователь ЭС сообщает системе конкретные данные о решаемой задаче и стремится получить с её помощью результат. Пользователи-неспециалисты обращаются к ЭС за результатом, не умея получить его самостоятельно, пользователиспециалисты используют ЭС для ускорения и облегчения процесса получения результата. Следует подчеркнуть, что термин «пользователь»

является многозначным, так как использовать ЭС могут и эксперт, и инженер по знаниям, и программист. Поэтому, когда хотят подчеркнуть, что речь идёт о том, для кого делалась ЭС, используют термин «конечный пользователь».

В режиме консультации входные данные о задаче поступают в рабочую память. Решатель на основе входных данных из рабочей памяти и правил из БЗ формирует решение. В отличие от традиционных программ компьютерной обработки данных ЭС при решении задачи не только исполняет предписанную последовательность операций, но и сама формирует её.

Существует широкий класс приложений, в которых требуется учитывать изменения, происходящие в окружающем мире за время исполнения приложения. Для решения таких задач необходимо применять динамические ЭС, которые наряду с компонентами статических систем содержат подсистему моделирования внешнего мира и подсистему связи с внешним окружением. Подсистема моделирования внешнего мира необходима для прогнозирования, анализа и адекватной оценки состояния внешней среды. Изменения окружения решаемой задачи требуют изменения хранимых в ЭС знаний, для того чтобы отразить временную логику происходящих в реальном мире событий.

Компонента связи с внешним миром актуальна для автономных интеллектуальных систем (роботов), а также для интеллектуальных систем управления. Связь с внешним миром осуществляется через систему датчиков и контроллеров.

Трудоёмкость разработки ИИС в значительной степени зависит от используемых инструментальных средств. Инструментальные средства для разработки интеллектуальных приложений можно классифицировать по следующим основным параметрам:

уровень используемого языка;

парадигмы программирования и механизмы реализации;

способ представления знаний;

механизмы вывода и моделирования;

средства приобретения знаний;

технологии разработки приложений.

Уровень используемого языка. Мощность и универсальность языка программирования определяет трудоёмкость разработки ЭС.

**1. Традиционные (в том числе объектно-ориентированные) языки программирования типа С, C++ (как правило, они используются не для создания ЭС,**а для создания инструментальных средств).

**2. Специальные языки программирования (например, язык LISP, ориентированный на обработку списков; язык логического программирования PROLOG;**язык рекурсивных функций РЕФАЛ и т.д.).

Их недостатком является слабая приспособленность к объединению с программами, написанными на языках традиционного программирования.

**3. Инструментальные средства, содержащие многие, но не все компоненты ЭС (например, система OPS5, которая поддерживает продукционный**подход к представлению знаний; языки KRL и FRL, используемые для разработки ЭС с фреймовым представлением знаний). Такое программное обеспечение предназначено для разработчиков, владеющих технологиями программирования и умеющих интегрировать разнородные компоненты в программный комплекс.

**4. Оболочки ЭС общего назначения, содержащие все программные компоненты, но не имеющие знаний о конкретных предметных средах. Средства этого**типа и последующего не требуют от разработчика приложения знания программирования. Примерами являются ЭКО, Leonardo, Nexpert Object, Kappa, EXSYS, GURU, ART, KEE и др.

В последнее время всё реже употребляется термин «оболочка», его заменяют более широким термином «среда разработки». Если хотят подчеркнуть, что средство используется не только на стадии разработки приложения, но и на стадиях использования и сопровождения, то употребляют термин «полная среда» (completeen vironment). Для поддержания всего цикла создания и сопровождения программ используются интегрированные инструментальные системы типа WorkBench, например KEATS [18], Shelly [16], VITAL [19]. Основными компонентами системы KEATS являются: ACQUIST – средства фрагментирования текстовых источников знаний, позволяющие разбивать текст или протокол беседы с экспертом на множество взаимосвязанных, аннотированных фрагментов и создавать понятия (концепты); FLIK – язык представления знаний средствами фреймовой модели; GIS – графический интерфейс, используемый для создания гипертекстов и концептуальных моделей, а также для проектирования фреймовых систем;

ERI – интерпретатор правил, реализующий процедуры прямого и обратного вывода; TRI – инструмент визуализации логического вывода, демонстрирующий последовательность выполнения правил; Tables – интерфейс манипулирования таблицами, используемыми для хранения знаний в БЗ; CS – язык описания и распространения ограничений;

TMS – немонотонная система поддержания истинности.

При использовании инструментария данного типа могут возникнуть следующие трудности:

а) управляющие стратегии, заложенные в механизм вывода, могут не соответствовать методам решения, которые использует эксперт, взаимодействующий с данной системой, что может привести к неэффективным, а возможно, и неправильным решениям;

б) способ представления знаний, используемый в инструментарии, мало подходит для описания знаний конкретной предметной области.

Большая часть этих трудностей разрешена в проблемно/предметно-ориентированных средствах разработки ИИС.

**5. Проблемно/предметно-ориентированные оболочки и среды (не требуют знания программирования):**

проблемно-ориентированные средства – предназначены для решения задач определённого класса (задачи поиска, управления, планирования, прогнозирования и др.) и содержат соответствующие этому классу альтернативные функциональные модули;

предметно-ориентированные средства – включают знания о типах предметных областей, что сокращает время разработки БЗ.

При использовании оболочек и сред разработчик приложения полностью освобождается от программирования, его основные трудозатраты связаны с формированием базы знаний.

Парадигмы программирования и механизмы реализации. Способы реализации механизма исполняемых утверждений часто называют парадигмами программирования. К основным парадигмам относят следующие:

процедурное программирование;

программирование, ориентированное на данные;

программирование, ориентированное на правила;

объектно-ориентированное программирование.

Парадигма процедурного программирования является самой распространённой среди существующих языков программирования (например, С и Pascal). В процедурной парадигме активная роль отводится процедурам, а не данным; причём любая процедура активизируется вызовом. Подобные способы задания поведения удобны для описаний детерминированной последовательности действий одного процесса или нескольких взаимосвязанных процессов.

При использовании программирования, ориентированного на данные, активная роль принадлежит данным, а не процедурам. Здесь со структурами активных данных связывают некоторые действия (процедуры), которые активизируются тогда, когда осуществляется обращение к этим данным.

В парадигме, ориентированной на правила, поведение определяется множеством правил вида «условие–действие». Условие задаёт образ данных, при возникновении которого действие правила может быть выполнено. Правила в данной парадигме играют такую же роль, как и операторы в процедурной парадигме. Однако если в процедурной парадигме поведение задаётся детерминированной последовательностью операторов, не зависящей от значений обрабатываемых данных, то в парадигме, ориентированной на правила, поведение не задатся заранее предписанной последовательностью правил, а формируется на основе значений данных, которые в текущий момент обрабатываются программой. Подход, ориентированный на правила, удобен для описания поведения, гибко и разнообразно реагирующего на большое многообразие состояний данных.

Парадигма объектного программирования в отличие от процедурной парадигмы не разделяет программу на процедуры и данные.

Здесь программа организуется вокруг сущностей, называемых объектами, которые включают локальные процедуры (методы) и локальные данные (переменные). Поведение (функционирование) в этой парадигме организуется путём пересылки сообщений между объектами.

Объект, получив сообщение, осуществляет его локальную интерпретацию, основываясь на локальных процедурах и данных. Такой подход позволяет описывать сложные системы наиболее естественным образом. Он особенно удобен для интегрированных ЭС.

Способ представления знаний. Наличие многих способов представления знаний вызвано стремлением представить различные типы проблемных сред с наибольшей эффективностью. Обычно способ представления знаний в ЭС характеризуют моделью представления знаний. Типичными моделями представления знаний являются правила (продукции), фреймы (или объекты), семантические сети, логические формулы. Инструментальные средства, имеющие в своём составе более одной модели представления знаний, называют гибридными.

Большинство современных средств, как правило, использует объектноориентированную парадигму, объединённую с парадигмой, ориентированной на правила. Одно из современных средств, позволяющее использовать целый ряд подходов, обеспечивающее поддержку программирования на основе правил, объектно-ориентированного и процедурного программирования – это язык CLIPS.

Язык CLIPS (название которого представляет собой сокращение от С Language Integrated Production System – продукционная система, интегрированная с языком С) был разработан с использованием языка программирования С в Космическом центре NASA/Джонсон.

Перед разработчиками этого языка была поставлена конкретная задача – обеспечить полную переносимость, низкую стоимость и простую интеграцию с внешними системами. Первоначально CLIPS обеспечивал поддержку только программирования на основе правил (отсюда происходит часть его обозначения как «продукционной системы»). Но уже в версии 5.0 языка CLIPS введена поддержка процедурного и объектно-ориентированного программирования.

Возможности логического вывода и представления, предоставляемые основанным на правилах языком программирования CLIPS, аналогичны возможностям языка OPS5, но являются более мощными.

По своей синтаксической структуре правила CLIPS весьма напоминают правила, применяемые в таких языках, как Eclipse, CLIPS/R2 и Jess, но CLIPS поддерживает только правила прямого логического вывода.

Язык программирования CLIPS, позволяющий использовать целый ряд подходов, обеспечивает поддержку программирования на основе правил, объектно-ориентированного и процедурного программирования.

Таким образом, сегодня CLIPS – это эффективное средство разработки экспертных систем.

Механизмы вывода и моделирования. В статических ЭС единственным активным агентом, изменяющим информацию, является механизм вывода экспертной системы. В динамических ЭС изменение данных происходит не только вследствие функционирования механизма исполняемых утверждений, но также в связи с изменениями окружения задачи, которые моделируются специальной подсистемой или поступают извне. Механизмы вывода в различных средах могут отличаться способами реализации следующих процедур.

**1. Структура процесса получения решения:**

построение дерева вывода на основе обучающей выборки (индуктивные методы приобретения знаний) и выбор маршрута на дереве вывода в режиме решения задачи;

компиляция сети вывода из специфических правил в режиме приобретения знаний и поиск решения на сети вывода в режиме решения задачи;

генерация сети вывода и поиск решения в режиме решения задачи, при этом генерация сети вывода осуществляется в ходе выполнения операции сопоставления, определяющей пары «правило– совокупность данных», на которых условия этого правила удовлетворяются;

в режиме решения задач ЭС осуществляет выработку правдоподобных предположений (при отсутствии достаточной информации для решения); выполнение рассуждений по обоснованию (опровержению) предположений; генерацию альтернативных сетей вывода; поиск решения в сетях вывода.

**2. Поиск (выбор) решения:**

направление поиска – от данных к цели, от целей к данным, двунаправленный поиск;

порядок перебора вершин в сети вывода – «поиск в ширину», при котором сначала обрабатываются все вершины, непосредственно связанные с текущей обрабатываемой вершиной G; «поиск в глубину», когда сначала раскрывается одна наиболее значимая вершина – G1 связанная с текущей G, затем вершина G1 делается текущей, и для неё раскрывается одна наиболее значимая вершина G2 и т.д.

**3. Процесс генерации предположений и сети вывода:**

режим – генерация в режиме приобретения знаний, генерация в режиме решения задачи;

полнота генерируемой сети вывода – операция сопоставления применяется ко всем правилам и ко всем типам указанных в правилах сущностей в каждом цикле работы механизма вывода (обеспечивается полнота генерируемой сети); используются различные средства для сокращения количества правил и (или) сущностей, участвующих в операции сопоставления; например, применяется алгоритм сопоставления или используются знания более общего характера (метазнания).

Механизм вывода для динамических проблемных сред дополнительно содержит: планировщик, управляющий деятельностью ЭС в соответствии с приоритетами; средства, гарантирующие получение лучшего решения в условиях ограниченности ресурсов; систему поддержания истинности значений переменных, изменяющихся во времени.

В динамических инструментальных средствах могут быть реализованы следующие варианты подсистемы моделирования:

система моделирования отсутствует;

существует система моделирования общего назначения, являющаяся частью инструментальной среды;

существует специализированная система моделирования, являющаяся внешней по отношению к программному обеспечению, на котором реализуется ЭС.

Средства приобретения знаний. В инструментальных системах они характеризуются следующими признаками:

**1. Уровень языка приобретения знаний:**

формальный язык;

ограниченный естественный язык;

язык пиктограмм и изображений;

ЕЯ и язык изображений.

**2. Тип приобретаемых знаний:**

данные в виде таблиц, содержащих значения входных и выходных атрибутов, по которым индуктивными методами строится дерево вывода;

специализированные правила;

общие и специализированные правила.

**3. Тип приобретаемых данных:**

атрибуты и значения;

классы структурированных объектов и их экземпляры, получающие значения атрибутов путём наследования.

Промышленная технология создания интеллектуальных систем включает следующие этапы:

исследование выполнимости проекта;

разработку общей концепции системы;

разработку и тестирование серии прототипов;

разработку и испытание головного образца;

разработку и проверку расширенных версий системы;

привязку системы к реальной рабочей среде.

Проектирование ЭС основано на трёх главных принципах:

**1. Мощность экспертной системы обусловлена прежде всего мощностью БЗ и возможностями её пополнения и только затем – используемыми методами**(процедурами) обработки информации.

**2. Знания, позволяющие эксперту (или экспертной системе) получить качественные и эффективные решения задач, являются в основном**эвристическими, эмпирическими, неопределёнными, правдоподобными.

**3. Неформальный характер решаемых задач и используемых знаний делает необходимым обеспечение активного диалога пользователя с ЭС в процессе**её работы.

Перед тем как приступить к разработке ЭС, инженер по знаниям должен рассмотреть вопрос, следует ли разрабатывать ЭС для данного приложения. Положительное решение принимается тогда, когда разработка ЭС возможна, оправданна, и методы инженерии знаний соответствуют решаемой задаче.

Чтобы разработка ЭС была возможной для данного приложения, необходимо выполнение, по крайней мере, следующих требований:

существуют эксперты в данной области, которые решают задачу значительно лучше, чем начинающие специалисты;

эксперты сходятся в оценке предлагаемого решения, так как в противном случае будет невозможно оценить качество разработанной ЭС;

эксперты способны вербализовать (выразить на естественном языке) и объяснить используемые ими методы, иначе трудно рассчитывать на то, что знания экспертов будут «извлечены» и заложены в ЭС;

решение задачи требует только рассуждений, а не действий;

задача не должна быть слишком трудной (т.е. её решение должно занимать у эксперта несколько часов или дней, а не недель или лет);

задача хотя и не должна быть выражена в формальном виде, но всё же должна относиться к достаточно «понятной» и структурированной области, т.е. должна существовать возможность выделения основных понятий, отношений и способов получения решения задачи;

решение задачи не должно в значительной степени опираться на «здравый смысл» (т.е. широкий спектр общих сведений о мире и о способе его функционирования, которые знает и умеет использовать любой нормальный человек), так как подобные знания пока не удаётся в достаточном количестве заложить в системы искусственного интеллекта.

Приложение соответствует методам ЭС, если решаемая задача обладает совокупностью следующих характеристик:

задача может быть естественным образом решена посредством манипулирования символами (с помощью символических рассуждений), а не манипулирования числами, как принято в математических методах и в традиционном программировании;

задача должна иметь эвристическую, а не алгоритмическую природу, т.е. её решение должно требовать применения эвристических правил. Для задач, которые могут быть гарантированно решены (при соблюдении заданных ограничений) с помощью формальных процедур, существуют более эффективные подходы, чем технологии ЭС.

При разработке ЭС, как правило, используется концепция быстрого прототипа, суть которой заключается в том, что разработчики не пытаются сразу построить конечный продукт. На начальном этапе они создают прототип (возможно, не единственный) ЭС, удовлетворяющий двум противоречивым требованиям: умение решать типичные задачи конкретного приложения и незначительные время и трудоёмкость его разработки. При выполнении этих условий становится возможным параллельно вести процесс накопления и отладки знаний, осуществляемый экспертом, и процесс выбора (разработки) программных средств, выполняемый инженером по знаниям и программистами. Для удовлетворения указанным требованиям при создании прототипа используются разнообразные инструментальные средства, ускоряющие процесс проектирования.

Традиционная технология реализации ЭС включает шесть основных этапов: идентификацию, концептуализацию, формализацию, выполнение, тестирование, опытную эксплуатацию [11].

Этапы разработки экспертных систем. На этапе идентификации определяются задачи, подлежащие решению, цели разработки, эксперты и типы пользователей.

На этапе концептуализации проводится содержательный анализ проблемной области, выявляются используемые понятия и их взаимосвязи, определяются методы решения задач.

На этапе формализации выбираются инструментальные средства и способы представления всех видов знаний, формализуются основные понятия, определяются способы интерпретации знаний, моделируется работа системы, оценивается адекватность системы зафиксированных понятий, методов решения, средств представления и манипулирования знаниями рассматриваемой предметной области.

На этапе выполнения осуществляется заполнение базы знаний.

В связи с тем, что основой ЭС являются знания, данный этап является одним из самых важных и самых трудоёмких. Процесс приобретения знаний разделяют на извлечение знаний в диалоге с экспертами; организацию знаний, обеспечивающую эффективную работу системы, и представление знаний в виде, «понятном» ЭС. Процесс приобретения знаний осуществляется инженером по знаниям на основе анализа деятельности эксперта по решению реальных задач. Проблемы приобретения знаний подробно описаны в главе 4.

На этапе тестирования эксперт и инженер по знаниям в интерактивном режиме с использованием диалоговых и объяснительных средств проверяют компетентность ЭС. Процесс тестирования продолжается до тех пор, пока эксперт не решит, что система достигла требуемого уровня компетентности.

На этапе опытной эксплуатации проверяется пригодность ЭС для конечных пользователей. Полученные результаты могут показать необходимость существенной модификации ЭС.

Процесс создания ЭС не сводится к строгой последовательности перечисленных выше этапов. В ходе разработки приходится неоднократно возвращаться на более ранние этапы и пересматривать принятые там решения.

Инструментальные средства различаются в зависимости от того, какую технологию разработки ЭС они допускают. Можно выделить, по крайней мере, четыре подхода к разработке ЭС:

подход, базирующийся на поверхностных знаниях;

структурный подход;

подход, основанный на глубинных знаниях;

смешанный подход, опирающийся на использование поверхностных и глубинных знаний.

Поверхностный подход применяется для сложных задач, которые не могут быть точно описаны. Его сущность состоит в получении от экспертов фрагментов знаний, релевантных решаемой задаче. При этом не предпринимается попыток систематического или глубинного изучения области, что предопределяет использование поиска в пространстве состояний в качестве универсального механизма вывода.

Обычно в ЭС, использующих данный подход, в качестве способа представления знаний выбираются правила. Условие каждого правила определяет образец некоторой ситуации, в которой правило может быть выполнено. Поиск решения состоит в выполнении тех правил, образцы которых сопоставляются с текущими данными. При этом предполагается, что в процессе поиска решения последовательность формируемых таким образом ситуаций не оборвётся до получения решения, т.е. не возникнет неизвестной ситуации, которая не соответствует ни одному правилу. Данный подход с успехом применяется к широкому классу приложений, но оказывается неэффективным в тех случаях, когда задача может структурироваться или для её решения может использоваться некоторая модель.

Структурный подход к построению ЭС предусматривает структуризацию знаний проблемной области. Его появление обусловлено тем, что для ряда приложений применение техники поверхностных знаний не обеспечивает решения задачи. Структурный подход к построению ЭС во многом похож на структурное программирование. Однако применительно к ЭС речь идёт не о том, чтобы структурирование задачи было доведено до точного алгоритма (как в традиционном программировании), а предполагается, что часть задачи решается с помощью эвристического поиска. Структурный подход в различных приложениях целесообразно сочетать с поверхностным или глубинным.

При глубинном подходе компетентность ЭС базируется на модели той проблемной среды, в которой она работает. Модель может быть определена различными способами (декларативно, процедурно). Необходимость использования моделей в ряде приложений вызвана стремлением исправить недостаток поверхностного подхода, связанный с возникновением ситуаций, не описанных правилами, хранящимися в БЗ. Экспертные системы, разработанные с применением глубинных знаний, при возникновении неизвестной ситуации способны самостоятельно определить, какие действия следует выполнить, с помощью некоторых общих принципов, справедливых для данной области экспертизы.

Глубинный подход требует явного описания структуры и взаимоотношений между различными сущностями проблемной области.

В этом подходе необходимо использовать инструментальные средства, обладающие возможностями моделирования: объекты с присоединёнными процедурами, иерархическое наследование свойств, активные знания (программирование, управляемое данными), механизм передачи сообщений объектам (объектно-ориентированное программирование) и т.п.

Смешанный подход в общем случае может сочетать поверхностный, структурный и глубинный подходы. Например, поверхностный подход может применяться для поиска адекватных знаний, которые затем используются некоторой глубинной моделью.

**1.2 Способы представления и обработки знаний в интеллектуальных и экспертных системах**

**Особенности создания баз данных и правил на языке CLIPS. При работе с CLIPS применяется понятие факта.**

Факт представляет собой основную единицу данных, используемую правилами. Факты помещаются в текущий список фактов fact-list.

Количество фактов в списке и объём информации, содержащейся в факте, ограничивается только размером памяти компьютера [4 – 8].

Факт может описываться индексом или адресом. Всякий раз, когда факт добавляется (изменяется) ему присваивается уникальный целочисленный индекс. Индексы в fact-list начинаются с нуля.

Идентификатор факта – это короткая запись факта, которая состоит из символа факта – f и индекса факта (f-10). Например:

f-0 (today is Sunday), f-1 (weather is warm).

Факты представляются в двух форматах: позиционные и непозиционные.

Позиционные факты – состоят из выражения символьного типа, за которым следует последовательность (возможно, пустая) из полей, разделённых пробелами. Вся запись заключается в скобки. Для того чтобы обратиться к информации, содержащейся в позиционном факте, пользователь должен знать, какие данные содержаться в факте и в каком поле они хранятся.

(altitude is 10000 feet) (grocery\_list bread milk eggs) (today is Sunday) (weather is warm) Поля в позиционных фактах могут быть любого простого типа, за исключением первого поля, которое всегда должно быть типа symbol.

В тексте программы факты можно включать в базу не по одиночке, а целым массивом. Для этого в CLIPS имеется команда deffacts.

(deffacts today (today is Sunday) (weather is warm)) Выражение начинается с команды deffacts, затем приводится имя списка фактов, который программист собирается определить (в нашем примере – today), а за ним следуют элементы списка, причём их количество не ограничивается.

Конструкция defclass. Прежде чем появится возможность создания экземпляров, в систему CLIPS необходимо передать информацию о списке допустимых слотов для данного конкретного класса. Для этой цели применяется конструкция defclass. В своей наиболее фундаментальной форме эта конструкция весьма напоминает конструкцию deftemplate [4 – 7]:

(defclass [] В этом определении терм определяет класс, от которого данный, вновь создаваемый класс должен наследовать информацию. Классом, от которого в конечном итоге наследуют информацию все определяемые пользователем классы, является системный класс USER. Определяемый пользователем класс должен наследовать информацию либо от другого определяемого пользователем класса, либо от класса USER. Синтаксическое описание определено следующим образом:

С помощью этого синтаксиса экземпляр PERSON может быть описан с использованием такой конструкции defclass:

(defclass PERSON "PERSON defclass" При определении слотов конструкции defclass могут также применяться следующие атрибуты слота из конструкции deftemplate: type, range, cardinality, allowed-symbols, allowed-strings, allowed-Iexemes, allowed-integers, allowed-floats, allowed-numbers, allowed-values, allowedinstance-names, default и default-dynamic.

Пример применения таких атрибутов:

(defclass PERSON "PERSON defclass" Атрибуты слота для конструкций defclass называют также фасетами слота.

В CLIPS существуют следующие зарезервированные слова, которые не могут использоваться как первое поле любого факта: test, and, or, not, declare, logical, object, exists, forall.

Непозиционные факты (шаблонные факты) – реализуются через конструкцию, подобную структуре или записи в языках C и PASCAL.

Шаблонные факты позволяют задавать имена каждому из полей факта.

Для задания шаблона, который затем может использоваться при доступе к полям по именам, используется конструкция (deftemplate (slot-1) (slot-2) (slot-N)), где – имя шаблона, (slot-N) – именованное поле (или слот).

Слоты могут быть ограничены по типу, значению, числовому диапазону, могут содержать значение по умолчанию. Порядок следования слотов значения не имеет.

(deftemplate student "a student record" (slot name (type STRING)) (slot age (type NUMBER) (default 18))) Каждое определение шаблона состоит из произвольного имени шаблона, необязательного комментария и некоторого количества определений слотов (начинаются с ключевого слова slot или field). Слот включает поле данных, например name, и тип данных, например STRING. Можно указать и значение по умолчанию, как в приведённом выше примере, где возраст студента по умолчанию равен 18.

Если в программу включено приведённое выше определение шаблона, то выражение (deffacts students (student (name "fred")) (student (name "jack") (age 19))) приведёт к тому, что и базу фактов после выполнения команды reset будет добавлено (student (name "fred") (age 18)) (student (name "jack") (age 19)) При работе с базами данных язык CLIPS предоставляет пользователю возможность использования следующих операций над фактами:

добавление к списку фактов (assert); удаление из списка фактов (retract); изменение списка фактов (modify), дублирование списка фактов (duplicate); очищение списка фактов (clear).

Кроме того, команды assert и retract используются в выполняемой части правила (заключении правила) и с их помощью выполняется программное изменение базы фактов. Для вывода списка фактов, имеющихся в базе, используется команда facts. Для удаления из базы массив фактов применяется оператор (команда) undeffacts.

Работа с базой правил основывается на их представлении соответствующими форматами.

В языке CLIPS правила имеют следующий формат [4 – 8]:

< необязательный комментарий > < необязательное объявление > < предпосылка\_1 > < предпосылка\_m> (defrule chores "Things to do on Sunday" (declare (salience 10)) (today is Sunday) (weather is warm) (assert (wash car)) (assert (chop wood)) В этом примере Chores – произвольно выбранное имя правила.

Предпосылки условной части правила – это (today is Sunday) (weather is warm) сопоставляются затем интерпретатором с базой фактов, а действия, перечисленные в выполняемой части правила (она начинается после пары символов =>), вставят в базу два факта:

(chop wood) в случае, если правило будет активизировано. Приведённый в тексте правила комментарий "Things todo on Sunday" (Что сделать в воскресенье) поможет в дальнейшем вспомнить, чего ради это правило включено в программу. Выражение (declare (salience 10)) указывает на степень важности правила. Пусть, например, в программе имеется другое правило (defrule fun "Better things todo on Sunday" (salience 100) (today is Sunday) (weather is warm) (assert (drink beer)) (assert (play guitar)) Поскольку предпосылки обоих правил одинаковы, то при выполнении оговорённых условий они будут «конкурировать» за внимание интерпретатора, предпочтение будет отдано правилу, у которого параметр salience имеет более высокое значение, в данном случае – правилу fun. Параметру salience может быть присвоено любое целочисленное значение в диапазоне [-10 000, 10 000]. Если параметр salience в определении правила опущен, ему по умолчанию присваивается значение 0.

Обычно в определении правила присутствуют и переменные (они начинаются с символа ?). Если, например, правило (defrule pick-a-chore "Allocating chores to days" (today is ?day) (chore is ?job) (assert (do ?job on ?day))) будет сопоставлено с фактами (today is Sunday) (chore is carwash) то в случае активизации оно включит в базу новый факт (do carwash on Sunday) Аналогично, правило (defrule drop-a-chore "Allocating chores to days" (retract ?chore)) отменит выполнение работ по дому (?chore). Обратите внимание на то, что оба экземпляра переменной ?day должны получить одно и то же значение. Переменная ?chore в результате сопоставления должна получить ссылку на факт (это делает оператор ) Пример:

(deffunction hypotenuse (?а ?b) (sqrt (+ (\* ?а ?а) (\* ?b ?b))) Аргументы-переменные должны иметь префикс ?, как это показано в приведённом примере.

Вызовы функций в CLIPS имеют префиксную форму: аргументы стоят после её названия. Вызов функции производится в скобках:

(hypotenuse 7 4) После открывающейся скобки следует имя функции, затем идут аргументы, каждый из которых отделён одним или несколькими пробелами. Аргументами функции могут быть данные простых типов, переменные или вызовы других функций.

Функция возвращает результат последнего выражения в списке.

Иногда выполнение функции имеет побочные эффекты, как в приведённом ниже примере.

(deffunction init (?day) (assert (todayis ?day)) В результате после запуска функции на выполнение командой CLIPS> (init Sunday) будет выполнена команда reset и, следовательно, очищена база фактов, а затем в неё будет включён новый факт (today is Sunday).

А в результате запуска функции hypotenuse на выполнение, командой CLIPS> (hypotenuse 3 4) будет выдан известный ответ CLIPS> 5. (deffunction between(?lb ?value ?ub) Эта функция определяет, попало ли заданное целочисленное значение в диапазон между нижним и верхним пределами.

В некоторых задачах бывает полезным оператор присвоения bind.

Например, переменной ?а присваивается значение 4:

Для более подробного изучения функциональных возможностей языка CLIPS целесообразно воспользоваться литературными источниками [4 – 8].

**Особенности решения задач планирования действий системы**

**Задачи планирования – определить последовательность действий модуля решения,**на пример системы управления. Традиционное планирование основано на знаниях, поскольку создание плана требует организации частей знаний и частичных планов в процедуру решения.

Планирование используется в экспертных системах при рассуждении о событиях, происходящих во времени. Планирование находит применение в производстве, управлении, робототехнике, в задачах понимания естественного языка.

Планы создаются путём поиска в пространстве возможных действий до тех пор, пока не будет найдена последовательность, необходимая для решения задачи. Это пространство представляет состояния мира, которые изменяются при выполнении каждого действия. Поиск заканчивается, когда достигается целевое состояние (описание мира) [3].

Приведём фрагмент программы по планированию действий робота «Робот и ящик» [3].

Имеются 2 комнаты – А и В. В комнате А находится робот, в комнате В – ящик. Задача – вытолкнуть ящик в комнату А.

Эта задача решается с помощью шаблонных фактов. Введём шаблон in, определяющий местоположение предмета:

(deftemplate in (slot object (type SYMBOL)) (slotlocation (typeSYMBOL)) Слот object будет задавать название предмета или робота, location – название места, где этот предмет или робот находится.

Чтобы задать роботу конкретную цель действий зададим шаблон goal:

(deftemplate goal (slot object (type SYMBOL)) (slot from (type SYMBOL)) (slot to (type SYMBOL)) слот object определяет название объекта, который необходимо переместить, слоты from и to определяют откуда и куда.

На основе шаблонов in и goal запишем начальные факты:

(deffacts world (in (object robot) (location RoomA)) (in (object box) (location RoomB)) (goal (action push) (object box) (from RoomB) (to RoomA)) Первый факт соответствует тому, что робот находится в комнате А, второй, что ящик в комнате В, третий – перетащить ящик из комнаты B в A.

Заключительным этапом создания данной программы является создание правил. В данной задаче необходимо реализовать три правила, которые осуществляли бы следующие действия робота:

**1) перемещение робота в комнату, где находится объект;**

**2) перемещение робота с объектом в комнату, указанную в цели;**

**3) остановка программы если цель достигнута.**

Реализуем первое действие:

(defrule move (goal (object ?X) (from ?Y)) (in (object ?X) (location ?Y)) ?robot-position (modify ?robot-position (location ?Y)) В данном правиле имеются три предпосылки. В первой предпосылке, использующей шаблон goal, задаются значения переменных ?X и ?Y. Во второй определяется наличие объекта ?X в комнате ?Y.

В третьей предпосылке проверяется, что местоположение робота не соответствует ?Y и запоминается ссылка на данный факт в переменной ?robot-position. Если все предпосылки данного правила истинны, то с помощью оператора modify меняется значение слота location на значение переменной ?Y факта ?robot-position, т.е. робот перемещается в комнату, в которой находится объект, который необходимо переместить.

Аналогично реализуется правило перемещения робота с ящиком, в комнату, указанную в цели:

(defrule push (goal (object ?X) (from ?Y) (to ?Z)) ?object-position (halt)) Полный листинг программы представлен в [3].

**Возможности наследования информации**

**Одно из преимуществ использования языка COOL состоит в том, что классы могут**наследовать информацию от других классов, что позволяет обеспечить совместный доступ к информации. Рассмотрим, какие действия пришлось бы предпринимать при наличии конструкции deftemplate, которая представляет информацию о людях [4]:

(deftemplate PERSON "PERSON deftemplate" (slot full-name) (slot age) (slot eye-color) (slot hair-color)) В таком случае, если бы потребовалось представить дополнительную информацию, относящуюся к тому, кто является служащим компании или студентом университета, пришлось бы предпринять определённые усилия. Один из возможных подходов мог бы предусматривать дополнение конструкции deftemplate с именем PERSON для включения другой необходимой информации:

(deftemplate PERSON "PERSON deftemplate" (slot full-name) (slot age) (slot eye-color) (slot hair-color) (slot job-position) (slot employer) (slot salary) (slot university) (slot major) (slot GPA)) Но ко всем людям относились бы только четыре слота этой конструкции deftemplate: full-name, age, eye-color и hair-color. С другой стороны, слоты job-position, employer и salary относились бы только к служащим, а слоты university, major и GPA – только к студентам. По мере добавления информации о людях, занимающихся другой деятельностью, приходилось бы вводить всё больше и больше слотов в конструкцию deftemplate с именем PERSON, причём по большей части эти слоты оказались бы неприменимыми для всех людей.

Ещё один подход мог бы состоять в создании отдельных конструкций deftemplate для служащих и студентов, как в следующем примере:

(deftemplate employee "Employee deftemplate" (slot full-name) (slot eye-color) (slot hair-color) (slot job-position) (slot employer) (slot salary)) (deftemplate student "Student deftemplate" (slot full-name) (slot eye-color) (slot hair-color) (slot university) При использовании такого подхода каждая конструкция deftemplate содержит только необходимую информацию, но приходится дублировать некоторые из слотов. Если бы пришлось модифицировать атрибуты одного из таких дублирующихся слотов, то потребовалось бы вносить изменения во многих местах, чтобы обеспечить единообразие представления информации. Кроме того, если бы нужно было написать правило, позволяющее отыскивать всех людей с синими глазами, то пришлось бы использовать два шаблона вместо одного (а если потребовалось бы также включить факты PERSON, количество шаблонов стало бы равным трём), как показано ниже.

(defrule find-blue-eyes (or (employee (full-name ?name) (eye-color blue)) (student (full-name ?name) (eye-color blue))) (printout t ?full-name "has blue eyes." crlf)) Классы позволяют совместно использовать общую информацию, принадлежащую к различным категориям, без дублирования, или включения ненужной информации. Вернёмся к первоначально рассматриваемому определению конструкции defclass с именем PERSON:

(defclass PERSON "PERSON defclass" (is-a USER) (slot full-name) (slot age) (slot eye-color) (slot hair-color)) Чтобы определить новые классы, которые расширяют определение класса PERSON, достаточно указать имя класса PERSON в атрибуте is-a нового класса, как показано ниже.

(defclass EMPLOYEE "Employee defclass" (is-a PERSON) (slot job-position) (slot employer) (slot salary)) (defclass STUDENT "Student defclass" (is-a PERSON) (slot university) (slot major) (slot GPA)) Атрибуты класса PERSON наследуются и в классе EMPLOYEE, и в классе STUDENT. Примеры создания экземпляров для каждого из этих трёх классов иллюстрирует следующий диалог:

CLIPS> (make-instance [John] of PERSON) CLIPS> (make-instance [Jack] of EMPLOYEE) CLIPS> (make-instance [Jill] of STUDENT) CLIPS> (send [John] print) [John] of PERSON (full-name nil) (age nil) (eye-color nil) (hair-color nil) CLIPS> (send [Jack] print) [Jack] of EMPLOYEE (full-name nil) (eye-color nil) (hair-color nil) (job-position nil) (employer nil) CLIPS> (send [Jill] print) [Jill] of STUDENT (full-name nil) (age nil) (eye-color nil) (hair-color nil) (university nil) (major nil) (GPA nil) Обратите внимание на то, что каждый экземпляр содержит только слоты, относящиеся к его классу. Как показано в следующем подразделе, в любом классе можно переопределить любой слот, который был уже определён в любом из его суперклассов.

Класс, который либо прямо, либо косвенно наследует свойство другого класса, называется подклассом того класса, от которого он наследует свойства. Класс, от которого наследуются свойства, называется суперклассом наследующего класса. Классы PERSON, EMPLOYEE и STUDENT представляют собой подклассы класса USER. Классы EMPLOYEE и STUDENT являются подклассами класса PERSON.

Класс USER – суперкласс классов PERSON, EMPLOYEE и STUDENT, а класс PERSON – суперкласс классов EMPLOYEE и STUDENT. Иерархией классов с единичным наследованием называется такая иерархия, в которой каждый класс имеет только один суперкласс, связанный с ним прямыми отношениями наследования. Иерархией классов с множественным наследованием называется такая иерархия, в которой любой класс может иметь несколько суперклассов, связанных с ним прямыми отношениями наследования. В языке COOL поддерживается множественное наследование. Мы будем ограничиваться применением примеров единичного наследования. Ниже приведён пример класса, в котором используется множественное наследование (в нём рассматривается студент, который имеет работу) [4].

(defclass WORKING–STUDENT "Working Student defclass" (is-a STUDENT EMPLOYEE)) По умолчанию, если какой-то слот переопределяется в подклассе, то атрибуты слота из нового определения используются исключительно в экземплярах этого класса. Например, предположим, что определены следующие классы:

(defclass А (is-aUSER) (slot х (default 3)) (slot z (default 4))) (defclassВ (slot у (default 5)) (slot z (default 6))) В таком случае создание экземпляров классов А и В приведёт к получению следующих результатов:

CLIPS> (make-instance [a] of A) CLIPS> (make-instance [b] of В) CLIPS> (send [a] print) CLIPS> (send [b] print) [b] ofВ (х nil) (y 5) (z 6) Обратите внимание на то, что слоту х экземпляра b по умолчанию присвоено значение nil вместо 3. Это связано с тем, что при отсутствии заданного по умолчанию значения для слота х класса В полностью перекрывается заданное по умолчанию значение 3, присваиваемое слоту х в классе А. Чтобы обеспечить возможность наследовать атрибуты слота от суперклассов, можно воспользоваться атрибутом слота source.

Если этому атрибуту присваивается значение exclusive, которое применяется по умолчанию, то атрибуты для слота устанавливаются на основе наиболее конкретного класса, определяющего этот слот. В иерархии единичного наследования как таковой рассматривается класс, имеющий наименьшее количество суперклассов. Если же атрибуту source присваивается значение composite, то атрибуты, которые не определены явно в наиболее конкретном классе, определяющем слот, берутся из следующего по порядку наиболее конкретного класса, в котором определяется данный атрибут. Например, если описанные ранее конструкции defclass с именами А и В будут объявлены следующим образом:

(defclassА (is-a USER) (slot х (default 3)) (slot z (default 4))) (defclass В (is-a A) (slot x (source composite)) (slot у (default 5)) (slot z (default 6))) то после создания экземпляров классов А и В будут получены такие результаты:

CLIPS> (make-instance [a] of A) CLIPS> (make-instance [b] of В) CLIPS> (send [a] print) CLIPS> (send [b] print) Теперь, после того как слот х класса В объявлен с атрибутом source, которому присвоено значение composite, этот слот может наследовать заданный по умолчанию атрибут от класса А, и применяемое по умолчанию результирующее значение для слота х экземпляра b становится равным 3.

Возможно также запретить наследование значения слота с использованием атрибута слота propagation. Если этому атрибуту присваивается значение inherit, которое является заданным по умолчанию, то данный слот наследуется подклассами. А если этому атрибуту присваивается значение no-inherit, то слот подклассами не наследуется.

Например, если классы А и В будет определены следующим образом:

(slot х (propagation no-inherit)) то после создания экземпляров классов А и В будут получены такие результаты:

CLIPS> (make-instance [a] of A) CLIPS> (make-instance [b] of В) CLIPS> (send [a] print) CLIPS> (send [b] print) Экземпляр b класса В наследует слот у из класса А, но не наследует слот х из класса А, поскольку атрибут propagation последнего имеет значение no-inherit.

Абстрактные и конкретные классы. В языке CLIPS предусмотрена возможность определять классы [4 – 8], используемые только для наследования. Такие классы называются абстрактными классами. Создание экземпляров абстрактных классов невозможно. По умолчанию классы являются конкретными. Для указания на то, должен ли класс быть абстрактным (abstract) или конкретным (concrete), применяется атрибут класса role. Атрибут класса role должен быть указан после атрибута класса is-a, но перед любыми определениями слотов, например, как показано ниже [4].

(defclass ANIMAL

## (role abstract))

(defclass MAMMAL

## (role abstract))

(defclass CAT (defclass DOG Классы ANIMAL и MAMMAL являются абстрактными, а классы CAT и DOG – конкретными. Атрибут role наследуется, поэтому, хотя и не требуется объявлять класс MAMMAL как абстрактный, поскольку он наследует этот атрибут от класса ANIMAL, необходимо объявить классы CAT и DOG как конкретные, в связи с тем, что в противном случае они будут рассматриваться как абстрактные. Попытка создать экземпляр абстрактного класса приводит к формированию сообщения об ошибке, как в следующем примере:

CLIPS> (make-instance [animal-1] of ANIMAL) [INSMNGR3] Cannot create instances of

**Abstract**

class ANIMAL.

CLIPS> (make-instance [cat-1] of CAT) [cat-1] Настоятельная необходимость объявлять какой-либо класс как абстрактный не возникает, но при использовании такого подхода в соответствующих условиях код становится более удобным для сопровождения и проще обеспечивает повторное использование. При этом достаточно лишь исключить для пользователя возможность создавать экземпляры с помощью какого-то класса, если класс не предназначен для этой цели. Но если данный класс уже используется таким образом, то в будущих реализациях станет невозможным его исключение, поскольку это приведёт к нарушению работы существующего кода.

В рассматриваемом примере [4] ответ на вопрос о том, должны ли классы ANIMAL и MAMMAL быть абстрактными, не так уж однозначен. Если требуется создать картотеку с информацией о животных, содержащихся в некотором зоопарке, то данные классы, по-видимому, должны быть абстрактными, поскольку в природе не существует животных (в данном случае речь идёт о млекопитающих), которые соответствовали бы только этому определению и не относились бы к какому-то более конкретному виду живых существ. Но если бы предпринималась попытка идентификации какого-то животного, то вполне могла бы возникнуть необходимость создавать экземпляры класса ANIMAL или MAMMAL, например, для включения в них информации о том, что мы смогли выяснить в отношении данного животного.

(defclass A (is-a USER)) Класс А является прямым наследником класса USER. Список старшинства классов для A: A USER OBJECT.

Пример 2:

(defclass В (is-a USER)) Класс В является прямым наследником класса USER. Список старшинства классов для В: В USER OBJECT.

Пример 3:

(defclass С (is-a А В)) Класс С является прямым наследником классов А и В. Список старшинства классов для С: С А В USER OBJECT.

Пример 4:

(defclass D (is-a В A)) Класс D является прямым наследником классов А и В. Список старшинства классов для D: D В A USER OBJECT.

Пример 5:

(defclass Е (is-a А С)) В соответствии с правилом 2, А должен быть старше С. В нашем случае, С – это потомок А и является более старшим в соответствии с правилом 1. Ошибка.

Пример 6:

defclass Е (is-a С А)) Правильное определение класса из примера 5. Список старшинства для Е: Е С А В USER OBJECT.

Абстрактные и конкретные классы. Абстрактный класс предназначен только для наследования, на его основе не могут создаваться экземпляры. На основе конкретного класса могут создаваться его экземпляры [4 – 8].

Слоты. Слот – это место для хранения значений поля класса. Каждый экземпляр класса содержит копию всех слотов своего родителя.

Количество слотов класса ограничено только размером свободной памяти, имя слота – любой набор символов, за исключением зарезервированных слов. Потомок класса содержит слоты родителя. В случае конфликта имён слотов, он разрешается в соответствии с правилом старшинства.

(defclass A (is-a USER) (slot fooA) (slot barA)) (defclass В (is-a A) (slot fooB) (slot barB)) Список старшинства для A: A USER OBJECT. Экземпляр класса А будет иметь 2 слота: fooA и barA. Список старшинства для В: В A USER OBJECT. Экземпляр класса В будет иметь 4 слота: fooB, barB, fooA, barA.

Для каждого слота может быть определён набор фасетов. Фасеты описывают различные свойства слотов: значения по умолчанию, вид хранения, видимость и т.п. Более подробно фасеты будут рассмотрены далее.

Создание экземпляра класса производится командой (makeinstance a of А) – создаётся экземпляр с именем класса А. Другой вариант (создание массива экземпляра классов):

(definstances my\_inst Тип поля слота. Слот может содержать как одно, так и несколько значений. По умолчанию слот содержит только одно значение. Ключевое слово multislot устанавливает тип слота, позволяющий хранить несколько значений, а slot или singleslot устанавливает тип слота, который может содержать только одно значение. Многозначные слоты хранятся как значения с несколькими полями. Манипуляции с ними производятся посредством стандартных функций nth$ и length$. Для установки значения слота используется функция slotinsert$. Слоты с одним значением хранятся в CLIPS как обычные переменные стандартных типов.

CLIPS> (сlеаr) (defclass А (is-а USER) (rоlесоnсrеte) (multislot foo (сrеаtе-ассеssоr read) (default abc def ghi))) CLIPS> (make-instance а of А) CLIPS> (nth$ 2 (send [a] get-foo)) Если при создании слота указывается модификатор для создания методов для записи или чтения по умолчанию ((create-accessor readwrite)), то экземпляр класса будет реагировать на сообщения getимя\_слота и put-имя\_слота соответственно чтением и записью значения слота. Создание обработчиков сообщений будет рассмотрено далее.

Фасет для задания значений по умолчанию. Фасеты используются для задания значений слота по умолчанию при создании экземпляра класса. Фасет default используется для задания статических значений слота. Фасет default-dynamic используется для заданий значения слота, которое задаётся всякий раз при создании нового экземпляра класса.

Пример:

CLIPS> (сlеаr) CLIPS> (setgen 1) (defclass A (is-a USER) (rоlе concrete) (slot foo (default-dynamic (gensym)) (create-accessor read))) CLIPS> (make-instance al of А) CLIPS> (make-instance a2 of А) CLIPS> (send [а1] get-foo) CLIPS> (send [a2] get-foo) Фасет Storage. Фасет определяет, будет ли значение слота храниться локально в экземпляре класса (lосаl), либо это значение будет одно для всех экземпляров класса (shared).

Пример:

CLIPS> (clear) (defclass A (is-a USER) (role concrete) (slot foo (create-accessor write) (storage shared) (default 1)) (slot bar (create-accessor write) (storage shared) (default-dynamic 2)) (slot woz (create-accessor write) (storage local))) CLIPS> (make-instance a of A) CLIPS> (send [a] print) (woz ni1) CLIPS> (send [a] put-foo 56) СLIPS> (send [a] put-bar 104) CLIPS> (make-instance b of A) CLIPS> (send [b] print) CLIPS> (send [b] put-foo 34) CLIPS> (send [b] put-woz 68) CLIPS> (send [a] print) CLIPS> (send [b] print) Фасет типа доступа к слоту. Для слота может быть задано три типа фасетов[4 – 6]: read-write, read-only, initialize-only Пример работы с разными типами фасетов:

CLIPS> (clear) (defclass A (is-a USER) (role concrete) (slot foo (create-accessor write) (access read-write)) (slot bar (access read-only) (default abc)) (slot woz (create-accessor write) (access initialize-only))) (defmessage-handler A put-bar (?value) (dynamic-put (sym-cat bar) ?value)) CLIPS> (make-instance a of A (bar 34)) [MSGFUN3] bar slot in [a] of A: write access denied.

[PRCCODE4] Execution halted during the actions of message-handler put-bar primary in class A CLIPS> (make-instance a of A (foo 34) (woz 65)) CLIPS> (send [a] put-bar 1) [MSGFUN3] bar slot in [a] of A: write access denied.

[PRCCODE4] Execution halted during the actions of message-handler put-bar primary in class A CLIPS> (send [a] put-woz 1) [MSGFUN3] woz slot in [a] of A: write access denied.

[PRCCODE4] Execution halted during the actions of message-handler put-bar primary in class A CLIPS> (send [a] print) Изменение значений свойств объектов по правилам объектноориентированного программирования производится самими объектами, поэтому в языке CLIPS это реализовано посредством обработчиков сообщений [4 – 8].

Общий синтаксис команды создания обработчика сообщений:

(defmessage-handler Вызов обработчика сообщений экземпляра класса:

(send [имя\_экземпляра] имя\_метода параметры) Обработчик сообщений уникально идентифицируется наименованием класса и типом. Для класса обработчик сообщений может задаваться как при создании определения класса, так и после. Заметим, что при создании определения класса создаётся только заголовок обработчика сообщений. Собственно программный код обработчика создаётся позже при помощи команды defmessage-handler.

Обработчики сообщений, определяемые системой. За классами можно закрепить не только данные, но и процедурную информацию.

Процедуры, входящие в состав классов, называются обработчиками сообщений. Для каждого класса, кроме обработчиков сообщений, определяемых пользователем, автоматически создаётся также целый ряд обработчиков сообщений, определяемых системой. Эти обработчики сообщений можно вызывать для работы с некоторым экземпляром с помощью команды send. Команда send имеет следующий синтаксис [4 – 8]:

Например, сообщение print отображает информацию о слотах экземпляра:

CLIPS> (send [John] print) [John] of PERSON (full-name "John Q. Public") (eye-color blue) (hair-color black) Для каждого слота, определяемого в конструкции defclass, система CLIPS автоматически определяет обработчики сообщений слота с префиксами get- и put-, которые используются для выборки и задания значений слота. Действительные имена обработчиков сообщений формируются в результате добавления к этим префиксам имени слота. Поэтому, например, конструкция defclass с именем PERSON, имеющая слоты full-name, age, eye-color и hair-color, автоматически создаётся для данного класса с восемью обработчиками сообщений, имеющими имена get-full-name, put-full-name, get-age, put-age, get-eye-color, puteye-color, get-hair-color и put-hair-color. Обработчики сообщений get- не имеют параметров и возвращают значение слота, например [4]:

CLIPS> (send [John] get-full-name) "John Q. Public" CLIPS> (send [John] get-age) Обработчики сообщений put- принимают от нуля и больше параметров. Если параметры не задаются, то восстанавливается первоначальное, предусмотренное по умолчанию значение слота, а при передаче одного или большего количества параметров значение слота устанавливается с учётом этих параметров. Попытка поместить больше одного значения в однозначный слот приводит к возникновению ошибки. Обработчик сообщений put- возвращает значение, представляющее собой новое значение слота, например, как показано в следующем диалоге:

CLIPS> (send [Jack] get-age) CLIPS> (send [Jack] put-age 22) CLIPS> (send [Jack] get-age) CLIPS>(send [Jack] put-age) CLIPS> (send [Jack] get-age) Команда watch принимает в качестве параметров несколько элементов, подлежащих отслеживанию, которые относятся к данному экземпляру. Одним из таких элементов является slots (слоты). Если осуществляется отслеживание слотов, то при каждом изменении значения любого слота экземпляра выводится информационное сообщение. Отслеживание изменений в слотах можно отменить с помощью команды unwatch:

CLIPS > (watch slots) CLIPS> (send [Jack] put-age 24) ::= local slot age in instance Jack (unwatch slots) CLIPS> (send [Jack] put-age 22) Ещё одним заранее определённым обработчиком сообщений является delete. Как и можно было бы предположить, обработчик сообщений delete используется для удаления экземпляра. Он возвращает символ TRUE, если экземпляр был успешно удалён, в противном случае – символ FALSE:

CLIPS> (instances) [John] of PERSON [genl] of PERSON [Jack] of PERSON For a total of 3 instances.

CLIPS> (send [genl] delete) CLIPS> (instances) [John] of PERSON [Jack] of PERSON For a total of 2 instances.

Ещё одним отслеживаемым элементом является instances (экземпляры). Если отслеживаются экземпляры, то система CLIPS автоматически выводит сообщение каждый раз, когда создаётся или удаляется экземпляр. В отличие от того, какие действия выполняются при модификации значения слота факта, при модификации значения слота экземпляра не создаётся новый экземпляр с изменившимся значением и не удаляется первоначальный экземпляр, поэтому для наблюдения за изменениями значений слотов экземпляров необходимо использовать отслеживаемый элемент slots. Применение отслеживаемого элемента instances иллюстрируется в следующем примере диалогового выполнения команд:

CLIPS> (watch instances) CLIPS> (make-instance Jill of PERSON) ==>instance [Jill] of PERSON [Jill] CLIPS> (send [Jill] put-age 22) CLIPS> (send [Jill] delete) (unwatchinstances) Последовательность знаков свидетельствует о том, что экземпляр создаётся.

;;создаём класс «прямоугольник» и объявляем у него обработчик сообщений, позволяющий находить его площадь:

(defclass rectangle (is-a USER) (slot side-a (default 1)) (slot side-b (default 1)) (message-handler find-area)) ;;создаём тело обработчика сообщений:

(defmessage-handler rectangle find-area () (\* ?self:side-a ?self:side-b)) ;;создаём ещё один обработчик сообщений, позволяющий напечатать полученную площадь прямоугольника:

(defmessage-handler rectangle print-area() (printout t (send ?self find-area) crlf)) Ссылка на активный (т.е. принимающий сообщение в данный момент) экземпляр сущности может быть получена при помощи переменной ?self. Имя этого параметра зарезервировано.

Пример:

(defclass A (is-a USER) (role concrete) (slot foo (default 1)) (slot bar (default 2))) (defmessage-handler A print-all-slots () (printout t ?self:foo “ “?self:bar crlf)) CLIPS> (make-instance a of A) CLIPS> (send [a] print-all-slots) Опишите основные элементы языка CLIPS.