Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет»

Ю.Ю. Громов, О.Г. Иванова, В.В. Алексеев, М.П. Беляев, Д.П. Швец, А.И. Елисеев

ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

Рекомендовано федеральным государственным бюджетным образовательным учреждением высшего профессионального образования «Московский государственный технический университет имени Н.Э. Баумана» в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки 230400 «Информационные системы и технологии»

Тамбов
Издательство ФГБОУ ВПО «ТГТУ»
2013

УДК 004.8(075.8)
ББК $з$813я73
И73

Р е ц е н з е н т ы:

Заслуженный деятель науки Российской Федерации,
доктор физико-математических наук, профессор
В.Ф. Крапивин

Кандидат технических наук, профессор
Ю.Ф. Мартемьянов

И73 Интеллектуальные информационные системы и технологии : учебное пособие / Ю.Ю. Громов, О.Г. Иванова, В.В. Алексеев и др. – Тамбов : Изд-во ФГБОУ ВПО «ТГТУ», 2013. – 244 с. – 100 экз. – ISBN 978-5-8265-1178-7.

Рассматриваются методы искусственного интеллекта и их применение для решения задач из различных предметных областей. Описаны методы приобретения, представления и обработки знаний в интеллектуальных системах, а также технологии проектирования и реализации интеллектуальных систем. Особое внимание уделено практическим вопросам программирования в среде CLIPS.

Предназначено для студентов высших учебных заведений, обучающихся по направлению подготовки 230400 «Информационные системы и технологии».

УДК 004.8(075.8)
ББК $з$813я73

ISBN 978-5-8265-1178-7

ПРЕДИСЛОВИЕ

Настоящая книга посвящена интеллектуальным информационным системам и технологиям, т.е. вопросам организации, проектирования, разработки и применения систем, предназначенных для обработки информации, базирующихся на применении методов искусственного интеллекта.

В главе 1 приведены краткий обзор приложений искусственного интеллекта, общая характеристика интеллектуальных информационных систем, а также основные понятия и определения, которые используются в книге.

Известно, что только небольшую часть своих знаний человек может точно сформулировать вербальным или формальным способом. Обширная область интуитивных знаний специалистов, которые необходимы для успешной работы интеллектуальных систем, остаётся недоступной из-за отсутствия средств их извлечения и представления. Неуловимый характер человеческих знаний и их постоянное развитие помешали сторонникам «нисходящего метода» в области искусственного интеллекта удержать в своих руках пальму первенства. «Нисходящий метод» соответствует дедуктивному подходу, в рамках которого на этапе становления искусственного интеллекта разрабатывались программы, способные решать сложные задачи на основе логической обработки содержащихся в них знаний. Примерами таких программ являются знаменитый «Логик–Теоретик» и GPS – универсальный решатель задач. В их разработке участвовали известные учёные А. Ньюэлл, А. Тьюринг, К. Шеннон, Г. Саймон, Дж. Шоу и др.

«Восходящий метод» развивался в работах Дж. Маккалоха, У. Питтса, Ф. Розенблата и др., посвящённых созданию самоорганизующихся систем и «самообучающихся машин». Эти учёные опирались на идею Н. Винера об обратной связи, благодаря которой всё живое приспосабливается к окружающей среде и добивается своих целей. Так возникло направление, связанное с разработкой нейронных сетей, которое, не успев твёрдо встать на ноги, было подвергнуто суровой критике оппонентов из противоположного лагеря (М. Минский и С. Пейперт) и какое-то время считалось неперспективным. Однако стремительное развитие аппаратных компьютерных средств и не оправдавшиеся надежды на возможности экспертных систем с дедуктивными выводами стали причиной второго рождения нейросетевых технологий в 1980-х гг. Сегодня модели нейронных сетей активно разрабатываются и применяются для решения задач прогнозирования, распознавания, извлечения знаний из хранилищ данных. Этим вопросам посвящена глава 2.

Глава 3 содержит описание методов обработки информации, основанных на эволюционных аналогиях. Наличие символьной информации и отсутствие детерминированных алгоритмов её обработки в интеллектуальных системах послужило причиной возникновения задач поиска в пространстве высокой размерности, а также комбинаторных задач, для решения которых успешно применяются генетические алгоритмы, методы эволюционного программирования и эволюционные стратегии. При этом для решения сложных и плохо обусловленных задач всё чаще применяются комбинированные подходы, в которых сочетаются методы нечёткого представления знаний, модели нейронных сетей для получения результата при отсутствии чётко заданного алгоритма и генетические методы для оптимизации полученных решений.

В главе 4 рассмотрены вопросы проектирования и применения мультиагентных систем, ориентированных на автономное выполнение интеллектуальных задач в распределённых компьютерных средах. Это одно из самых новых направлений в искусственном интеллекте, которое имеет большие перспективы в связи с широким распространением Интернета и представляет особый интерес для специалистов в области экономики и бизнеса, так как является базой для создания виртуальных предприятий.

Одно из современных средств, позволяющее использовать целый ряд подходов, обеспечивающее поддержку программирования на основе правил объектно-ориентированного и процедурного программирования – это язык CLIPS. Название языка CLIPS – аббревиатура от **C Language Integrated Production System.** Язык был разработан в Центре космических исследований NASA (NASA's Johnson Space Center) в середине 1980-х годов и во многом сходен с языками, созданными на базе LISP, в частности OPS5 и ART. Хотя в то время на рынке уже появились программные средства для задач искусственного интеллекта, разработанные на языке C, специалисты из NASA решили создать такой продукт самостоятельно. В настоящее время эта система доступна во всем мире, и нужно отметить, что по своим возможностям она не уступает множеству гораздо более дорогих коммерческих продуктов, поэтому на рассмотрении возможностей данного языка авторы остановились более подробно. Вопросам реализации информационных технологий на языке CLIPS посвящена глава 5.

В главе 6 рассмотрены примеры реализации интеллектуальных информационных технологий на языке CLIPS.

В конце каждой главы учебника приведены список литературы, а также контрольные вопросы и задания. В приложениях А и Б учебника представлены содержательные примеры реализации экспертных систем на языке CLIPS.

1. ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ – ОСНОВА НОВЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

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.

1.1. Основные направления исследований в области интеллектуальных информационных систем

Интеллектуальные информационные системы проникают во все сферы нашей жизни, поэтому трудно провести строгую классифика-

цию 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]. Зада-

ча 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 (KE, 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 про-

мышленные 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.

1.2. Основные типы интеллектуальных информационных систем и их характеристика

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;

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, приложе-

ниями 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:

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 какой-либо 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.

1.3. Технологии разработки экспертных систем

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

1.1. Отличия систем искусственного интеллекта от обычных программных систем

Характеристика	Программирование в системах искусственного интеллекта	Традиционное программирование
Тип обработки	Символьный	Числовой
Метод	Эвристический поиск	Точный алгоритм
Задание шагов решения	Неявное	Явное
Искомое решение	Удовлетворительное	Оптимальное
Управление и данные	Смешаны	Разделены
Знания	Неточные	Точные
Модификации	Частые	Редкие

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:

- базу знаний;
- 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, пользователи-специалисты используют ЭС для ускорения и облегчения процесса по-

лучения 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 C, 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, C 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 задаётся 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 объектно-ориентированную 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 **C 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 – G_1 связанная с текущей G, затем вершина G_1 делается текущей, и для неё раскрывается одна наиболее значимая вершина G_2 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;

– объекты;

– 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 (рис. 1.3): идентификацию, концептуализацию, формализацию, выполнение, тестирование, опытную эксплуатацию [11].

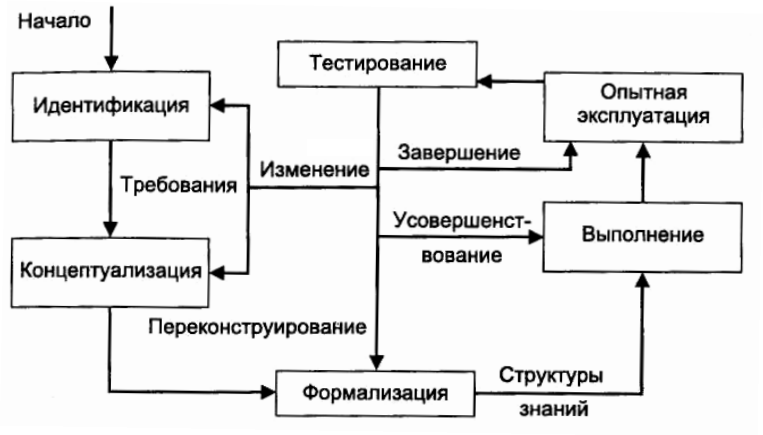


Рис. 1.3. Этапы разработки экспертных систем

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 эв-

ристического 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.

1.4. Контрольные вопросы и задания

1. Охарактеризуйте основные направления исследований, проводимые в области искусственного интеллекта.
2. Приведите известные вам примеры применения интеллектуальных систем в различных проблемных областях.
3. Перечислите признаки характерные для интеллектуальных информационных систем.
4. Назовите основные функции, присущие ИИС и способы их реализации.
5. Сформулируйте основные отличия систем искусственного интеллекта от обычных программных средств.
6. Дайте краткую характеристику систем с интеллектуальным интерфейсом, экспертных систем, самообучающихся систем и адаптивных информационных систем.
7. Перечислите основные типы систем с интеллектуальным интерфейсом и дайте им краткую характеристику.

8. Перечислите основные типы ЭС и дайте им краткую характеристику.

9. Перечислите основные типы самообучающихся информационных систем и дайте им краткую характеристику.

10. Перечислите основные типы адаптивных информационных систем и дайте им краткую характеристику.

11. Перечислите и охарактеризуйте основные компоненты статических экспертных систем.

12. Поясните отличие динамических экспертных систем от статических.

13. Охарактеризуйте экспертную систему по следующим параметрам: типу приложения, стадии существования, масштабу, типу проблемной среды, типу решаемой задачи.

14. Расскажите о подходах, применяемых к построению экспертных систем.

15. Назовите типы задач, которые решаются с применением ЭС. Приведите примеры.

16. Назовите специалистов, которые привлекаются для разработки экспертных систем, и поясните их функции.

17. Назовите парадигмы программирования и дайте их краткую характеристику.

18. Назовите типичные модели представления знаний в экспертных системах.

19. Расскажите об основных характеристиках инструментальных средств, предназначенных для разработки интеллектуальных информационных систем.

20. Назовите известные вам языки программирования и соответствующие им парадигмы программирования.

21. Перечислите этапы промышленной технологии создания интеллектуальных систем.

22. Опишите основные технологические этапы разработки экспертных систем: идентификацию, концептуализацию, формализацию, выполнение, тестирование, опытную эксплуатацию.

23. Расскажите о механизмах вывода в экспертных системах.

24. Расскажите, что вы знаете о языке CLIPS.

25. Приведите пример конкретной экспертной системы, используя для её характеристики признаки, которые приведены в данной главе.

1.5. Список литературы

1. Амамия, М. Архитектура ЭВМ и ИИ / М. Амамия, Ю. Танака. – М. : Мир, 1993.

2. Андрейчиков, А.В. Компьютерная поддержка изобретательства (методы, системы, примеры, применения) / А.В. Андрейчиков, О.Н. Андрейчикова. – М. : Машиностроение, 1998.

3. Буров, К. Обнаружение знаний в хранилищах данных / К. Буров // Открытые системы. – 1999. – № 5, 6.

4. Гаврилова, Т.А. Базы знаний интеллектуальных систем / Т.А. Гаврилова, В.Ф. Хорошевский. – СПб. : Питер, 2000.

5. Зарипов, Р.Х. Машинный поиск вариантов при моделировании творческого процесса / Р.Х. Зарипов. – М. : Наука, 1983.

6. Искусственный интеллект: Справочник. В 3-х кн. Кн. 1: Системы общения и экспертные системы / под ред. Э.В. Попова. – М. : Радио и связь, 1990.

7. Поспелов, Г.С. Искусственный интеллект – основа новой информационной технологии / Г.С. Поспелов. – М. : Наука, 1988.

8. Искусственный интеллект: Справочник. В 3-х кн. Кн. 3: Программные и аппаратные средства / под ред. В.А. Захарова, В.Ф. Хорошевского. – М. : Радио и связь, 1990.

9. Ларичев, О.И. Системы, основанные на экспертных знаниях: история, современное состояние и некоторые перспективы / О.И. Ларичев // Седьмая национальная конференция по искусственному интеллекту с международным участием : сб. науч. тр. – М. : Изд-во физико-математической литературы, 2000.

10. Масалович, А.И. От нейрона к компьютеру / А.И. Масалович // Журнал доктора Добба. – 1992. – № 1.

11. Статические и динамические экспертные системы : учеб. пособие / Э.В. Попов, И.Б. Фоминых, Е.Б. Кисель, М.Д. Шапот. – М. : Финансы и статистика, 1996.

12. Тельное, Ю.Ф. Интеллектуальные информационные системы в экономике : учеб.пособие / Ю.Ф. Тельное. – М. : СИНТЕГ, 1998.

13. Финн, В.К. Правдоподобные рассуждения в интеллектуальных системах типа ДСМ / В.К. Финн // Итоги науки и техники. Сер. «Информатика». Т. 15. Интеллектуальные информационные системы. – М. : ВИНИТИ, 1991.

14. Цуриков, В.М. Проект «Изобретающая машина» – интеллектуальная среда поддержки инженерной деятельности / В.М. Цуриков // Журнал ТРИЗ. – 1991.– № 21.

15. Элти, Дж. Экспертные системы: концепции и примеры / Дж. Элти, М. Кумбс ; пер. с англ. – М. : Финансы и статистика, 1987.

16. Bouchet, C. SHELLY: An integrated workbench for KBS development / C. Bouchet, C. Brunet, A. Anjewierden // Proc. of 9th Int. Workshop Expert Syst. and their Appl. – France, Avignon, 1989. – No. 1.

17. Durkin, J. Expert Systems: a view of the field / J. Durkin // IEEE Expert. – 1996. – No. 2.

18. Motta, E. Methodological foundations of KEATS, the knowledge Engineer's Assistant / E. Motta, T. Rajan, J. Dominigue, M. Eisenstadt // Knowledge Acquisition. – 1991. – No. 3.

19. VITAL: A methodology-based workbench for KBS life cycle Support // ESPRIT – II Project 5365, 1990.

2. НЕЙРОННЫЕ СЕТИ

Особенностью интеллектуальных систем является способность решать слабоструктурированные и плохо формализованные задачи. Эта способность основана на применении различных методов моделирования рассуждений для обработки символьной информации. Традиционным подходом к построению механизмов рассуждения является использование дедуктивного логического вывода на правилах (rule-based reasoning), который применяется в экспертных системах продукционного и логического типа (см. главу 1). При таком подходе необходимо заранее сформулировать весь набор закономерностей, описывающих предметную область. Альтернативный подход основан на концепции обучения по примерам (case-based reasoning). В этом случае при построении интеллектуальной системы не требуется заранее знать обо всех закономерностях исследуемой области, но необходимо располагать достаточным количеством примеров для настройки разрабатываемой адаптивной системы, которая после обучения будет способна получать требуемые результаты с определённой степенью достоверности. В качестве таких адаптивных систем применяются искусственные нейронные сети.

2.1. Модель искусственного нейрона

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 w_i, сигналов x_i, $i = 1, .., n$, где n – размерность пространства входных сигналов. Потенциал нейрона определяется по формуле

$$P = \sum_{i=1}^{n} w_i x_i \ .$$

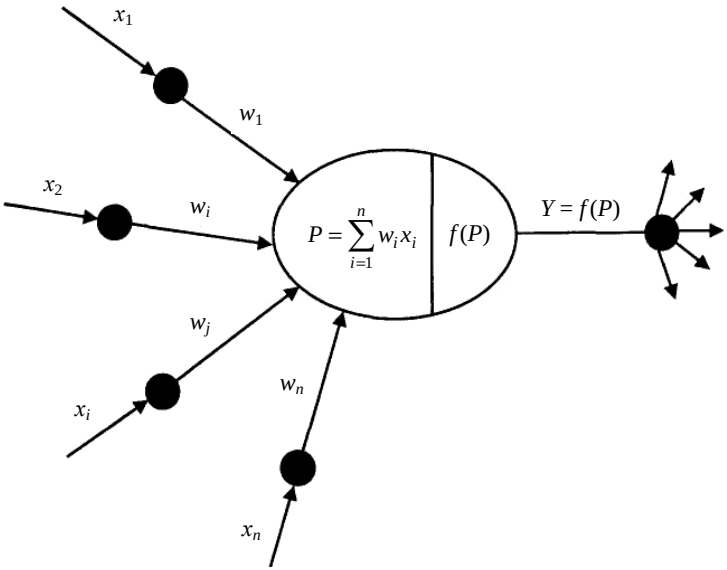
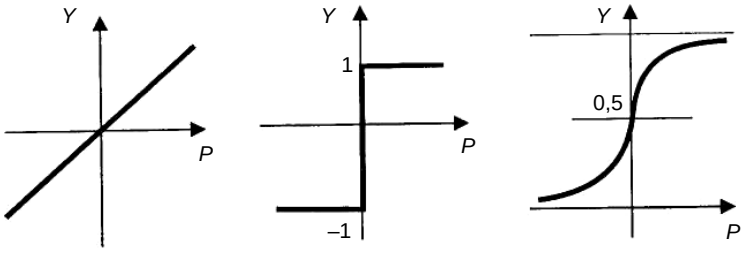


Рис. 2.1. Схема кибернетической модели нейрона



а) *б*) *в*)

Рис. 2.2. Функции переноса искусственных нейронов:
а – линейная; *б* – ступенчатая; *в* – сигмоидальная

Взвешенная сумма поступивших сигналов (потенциал) преобразуется с помощью передаточной функции $f(P)$ в выходной сигнал нейрона Y, который передается другим нейронам сети, т.е. $Y = f(P)$. 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 P^*, например:

$$Y = \begin{cases} 1, & \text{если } P \geq P^*; \\ -1, & \text{если } P < P^*. \end{cases}$$

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 $Y = 1/\left(1 + e^{-kp}\right)$, 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.

2.2. Модели нейронных сетей

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

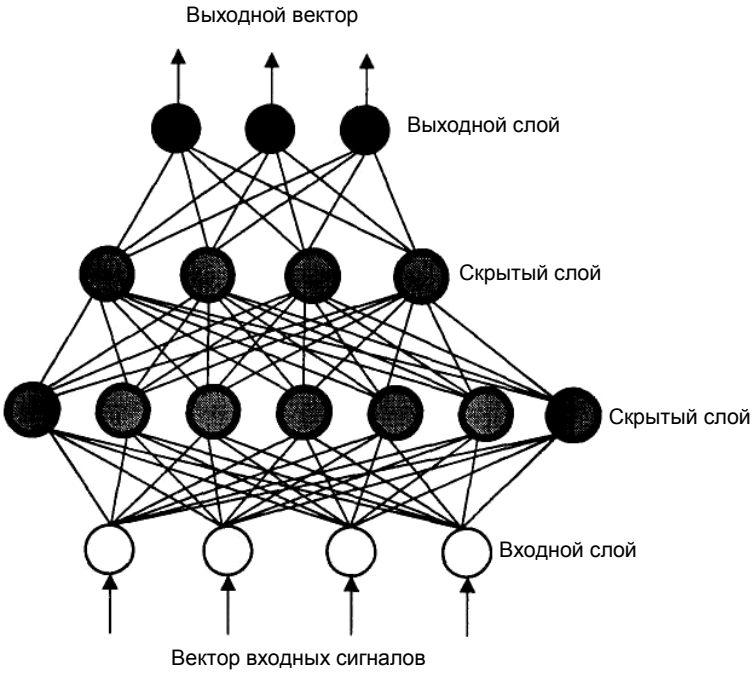


Рис. 2.3. Схема многослойного перцептрона

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.

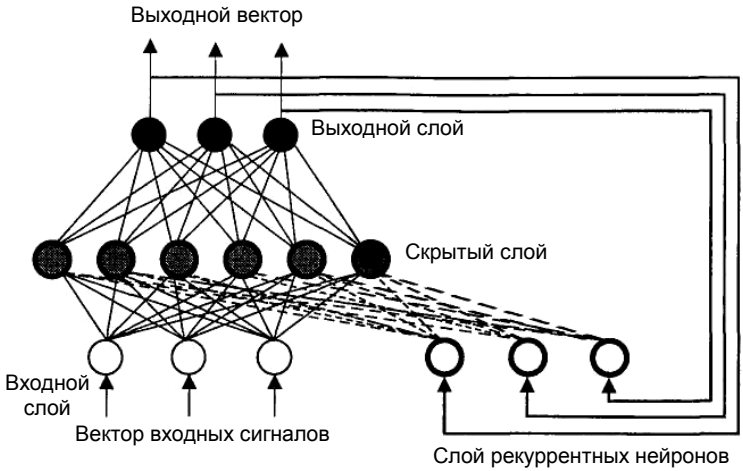


Рис. 2.4. Схема рекуррентной нейронной сети

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;
– $w_{ji} = w_{ij}$ – прямые и обратные связи симметричны;
– $w_{ii} = 0$ – 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.

2.3. Построение нейронной сети

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.

2.4. Обучение нейронной сети

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 $w_{jik} = w_{ji(k-1)} + \Delta w_{jik}$, где $\Delta w_{jik} = \eta \delta_{jk} x_{jik}$, $\delta_{jk} = T_{jk} - R_{jk}$, 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; η – 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 дельта-правилом, которое имеет вид: $\Delta_p w_{ji} = \eta \delta_{jp} y_{ip}$, где изменение в силе связи w_{ji} для p-й обучающей пары $\Delta_p w_{ji}$ пропорционально произведению сигнала ошибки j-го нейрона δ_{jp}, получающего входной сигнал по этой связи, и выходного сигнала i-го нейрона y_{ip}, посылающего сигнал по этой связи. Определение сигнала ошибки является рекурсивным процессом, который начинается с выходных блоков. Для выходного блока сигнал ошибки $\delta_{jp} = y'_j \left(T_{jp} - R_{jp}\right)$, где T_{jp} и R_{jp} – 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 $\delta_{jp} = y'_i \sum_k \delta_{kp} w_{kj}$. Для сигмоидальной функции $y'_j = y_j \left(1 - y_j\right)$, по-

этому на интервале $0 < y_j < 1$ производная имеет максимальное значение в точке $y_j = 0{,}5$, а в точках $y_j = 0$ и $y_j = 1$ 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 $y_j < 0{,}1$, а за 1 – значения $y_j > 0{,}9$.

Модификация весов производится после предъявления каждой пары вход–выход. Однако если коэффициент η, определяющий скорость обучения, мал, то можно показать, что обобщённое дельта-правило достаточно хорошо аппроксимирует минимизацию общей ошибки функционирования сети D методом градиентного спуска в пространстве весов. Общая ошибка функционирования сети определяется по формуле

$$D = \frac{1}{2} \sum_p \sum_j (T_{jp} - R_{jp})^2 .$$

Обучение продолжается до тех пор, пока ошибка не уменьшится до заданной величины. Эмпирические результаты свидетельствуют о том, что при малых значениях η 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 η.

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.

2.5. 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 по 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.). Объектно-ориентированные 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.

2.6. Практическое применение нейросетевых технологий

Применение нейросетевых технологий целесообразно при решении задач, имеющих следующие 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:

$$R_i = \begin{cases} \Delta P_i; \\ -\Delta P_i; \\ \Delta P_i = P_i - P_{i-1}, \end{cases}$$

где ΔP_i – оценка операции «вчера купил, сегодня продал»; $-\Delta P_i$ – оценка операции «вчера продал, сегодня купил»; P_i – значение выбранного показателя доходности в i-й день; P_{i-1} – значение показателя в $(i-1)$-й день.

Итоговая доходность за установленный интервал времени (n дней) рассчитывается по формуле

$$R = \sum_{i=1}^{n} (1 + \Delta P_i) - 1.$$

Результаты оценки доходности предприятия за 30 дней с использованием различных моделей ИНС, а также доходов «идеального» трейдера приведены ниже.

Стандартная трёхслойная сеть 0,1919
Стандартная четырёхслойная сеть 0,1182
Рекуррентная сеть с обратной отрицательной связью
от скрытого слоя 0,1378

Рекуррентная сеть с отрицательной обратной связью 0,4545
Сеть Ворда: с тремя скрытыми блоками,
с разными передаточными функциями 0,2656
Трёхслойная сеть с обходным соединением 0,1889
Четырёхслойная сеть с обходными соединениями 0,0003
Сеть с общей регрессией 0,3835
Сеть метода группового учёта аргументов 0,1065
Сеть Ворда: с тремя скрытыми блоками, с разными
передаточными функциями, с обходным соединением 0,1166
«Идеальный» трейдер 1,1448

«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 ΔP_i, (величина относительно изменения цены закрытия по акциям рассматриваемого предприятия X на завтрашний i-й день) трейдер принимает решение о покупке ($\Delta P_i > 0$) или продаже ($\Delta P_i < 0$) 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, приведена на рис. 2.5.

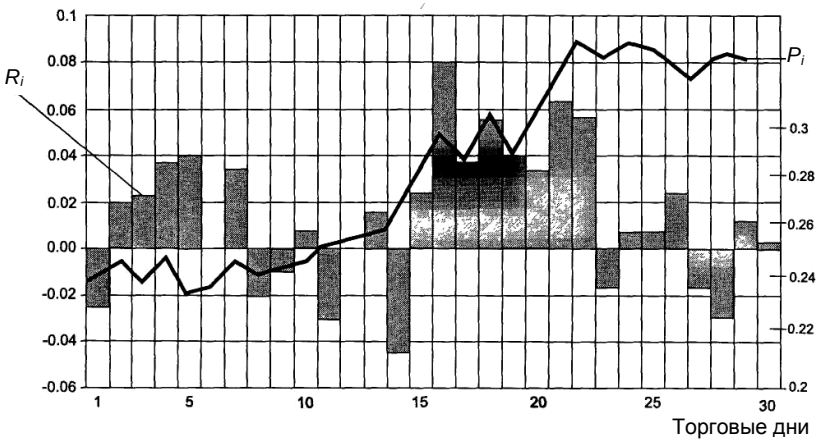


Рис. 2.5. Динамика изменения доходности (R_i) и цен закрытия (P_i) за 30 торговых дней, полученная на рекуррентной сети с отрицательной обратной связью

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 x_i – двоичная переменная со значениями 0 и 1, то примерно в половине случаев она будет иметь нулевое значение: $x_i = 0$. Поскольку x_i 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).

2.7. Контрольные вопросы и задания

1. Опишите модель искусственного нейрона. Приведите примеры передаточных функций.
2. Сравните свойства биологических и искусственных нейронных сетей.
3. Проведите сравнение однослойных и многослойных ИНС.
4. Раскройте особенности рекуррентных и самоорганизующихся сетей.
5. Расскажите о моделях сетей Хопфилда и Кохонена.
6. Дайте характеристику основных этапов построения нейронной сети.
7. Расскажите о методах обучения ИНС (коррекция по ошибке, обучение Хебба, соревновательное обучение, метод обратного распространения ошибки).
8. Опишите алгоритм обратного распространения ошибки. Сформулируйте его достоинства и недостатки.
9. Назовите и охарактеризуйте парадигмы обучения нейронной сети.
10. Расскажите об известных вам способах реализации ИНС.
11. Поясните условия применимости ИНС. Сформулируйте основные проблемы, возникающие при применении нейронных сетей.
12. Назовите негативные последствия переобучения нейронной сети.
13. Подготовьте набор содержательных примеров для обучения нейронной сети с заданной целью.
14. Изобразите наиболее известные функции активации и дайте им характеристику.
15. Сформулируйте постановку прикладной задачи, для решения которой возможно и целесообразно применить нейронную сеть. Опишите, как это можно сделать.
16. Сформулируйте постановку содержательной задачи для решения методами нейронных сетей. Подготовьте обучающую и тестирующую выборки примеров.

17. Сформулируйте постановку задачи извлечения знаний для решения с помощью технологии нейронных сетей. Подготовьте необходимые данные.

18. Составьте задачу классификации (диагностики) для решения с помощью технологии нейронных сетей. Подготовьте необходимые данные, выберите топологию сети.

19. Сформулируйте задачу прогнозирования для решения с помощью технологии нейронных сетей. Подготовьте необходимые данные, выберите топологию сети.

20. Аргументировано выберете случай, в котором целесообразно применение ИНС:

а) выявление тенденций, взаимосвязей в больших объёмах данных, искажённых шумами;

б) построение аппроксимации функции по результатам эксперимента, когда количество опытов невелико.

21. Расскажите про выбор архитектуры и настройку многослойной нейронной сети.

22. Расскажите о задачах, решаемых при помощи самоорганизующихся карт Кохонена.

23. Назовите достоинства и недостатки алгоритма обратного распространения ошибки.

24. Назовите и дайте краткую характеристику базовым архитектурам нейронных сетей.

25. Расскажите о проблемах практического использования искусственных нейронных сетей.

2.8. Список литературы

1. Барцев, С.И. Адаптивные сети обработки информации / С.И. Барцев, В.А. Охонин. – Красноярск : Институт физики СО АН СССР, 1986.

2. Галушкин, А.И. Нейронные сети. Основы теории / А.И. Галушкин. – М. : Горячая Линия-Телеком, 2012. – 496 с.

3. Комьютер обретает разум / под ред. В.А. Стефанюк. – М. : Мир, 1990. – 240 с.

4. Осовский, С. Нейронные сети для обработки информации / С. Осовский ; пер. с польс. И.Д. Рудинского. – М. : Финансы и статистика, 2002.

5. Лозин, Н.В. Моделирование нейронных структур / Н.В. Лозин. – М. : Наука, 1970.

6. Розенблат, Ф. Принципы нейродинамики / Ф. Розенблат. – М. : Мир, 1965.

7. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. – М. : Горячая Линия-Телеком, 2007.

8. Соколов, Е.Н. Нейроинтеллект: от нейрона к нейрокомпьютеру / Е.Н. Соколов, Г.Г. Вайтнявичус. – М. : Наука, 1989.

9. Толкачев, С. Нейронное программирование диалоговых систем / С. Толкачев. – М. : Корона-Век, 2011.

10. Трикоз, Д.В. Нейронные сети: как это делается? / Д.В. Трикоз // Компьютеры + программы. – 1992. – № 4, 5.

11. Уоссермен, Ф. Нейрокомпьютерная техника: Теория и практика / Ф. Уоссермен. – М. : Мир, 1992.

12. Фролов, Ю.В. Интеллектуальные системы и управленческие решения / Ю.В. Фролов. – М. : Изд-во МГПУ, 2000.

13. Хинтон, Дж.Е. Как обучаются нейронные сети / Дж.Е. Хинтон // В мире науки. – 1992.– № 11, 12.

14. Элементарное введение в технологию нейронных сетей с примерами программ / Р.Тадеусевич [и др.]. – М. : Горячая Линия-Телеком, 2011.

15. Kohonen, T. Self-organization and associative memory / T. Kohonen. – New York : Springer, 1984.

3. ЭВОЛЮЦИОННЫЕ АНАЛОГИИ В ИСКУССТВЕННЫХ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМАХ

Эволюционное моделирование можно определить как воспроизведение процесса естественной эволюции с помощью специальных компьютерных программ. Термин эволюция в ограниченном смысле, касающемся только смены поколений организмов, начал широко использоваться в XVII в. С появлением в 1859 г. учения Дарвина этот термин приобрёл современное толкование: «Биологическая эволюция – историческое развитие организмов». Необходимые и достаточные условия, определяющие главные факторы эволюции, были сформулированы в XX в. на основе созданной популяционно-генетической теории. К факторам, определяющим неизбежность эволюции, относятся:

– наследственная изменчивость как предпосылка эволюции, её материал;

– борьба за существование как контролирующий и направляющий фактор;

– естественный отбор как преобразующий фактор.

На рисунке 3.1 приведена конкретизация факторов эволюции, учитывающая многообразие форм их проявления, взаимосвязей и взаимовлияния [11]. Главные факторы выделены пунктиром.

Современная теория эволюции базируется на теории общей и популяционной генетики. Элементарным объектом эволюции является

популяция – сообщество свободно скрещивающихся особей. В популяциях происходят микроэволюционные процессы, приводящие к изменению их генофонда. Преобразования генетического состава популяции происходят под действием элементарных эволюционных факторов. Случайные структурные или функциональные изменения в генах, хромосомах и других воспроизводимых единицах называют мутациями, если они приводят к наследственному изменению какого-либо фенотипического признака особи. Хромосомы – это специфические структуры клеточного ядра, которые играют важнейшую роль в процессах деления клеток. Хромосомы состоят из генов. Геном называется реально существующая, независимая, комбинирующаяся и расщепляющаяся при скрещиваниях единица наследственности.

Преобразования генофонда популяции происходят под управлением естественного отбора.

Эволюция – это многоэтапный процесс возникновения органических форм с более высокой степенью организации, который характеризуется изменчивостью самих эволюционных механизмов.

История эволюционных вычислений началась с разработки ряда независимых моделей, среди которых были генетические алгоритмы и классификационные системы, созданные американским исследователем Дж. Холландом. Он предложил использовать методы и модели развития органического мира на Земле в качестве механизма комбинаторного пе-

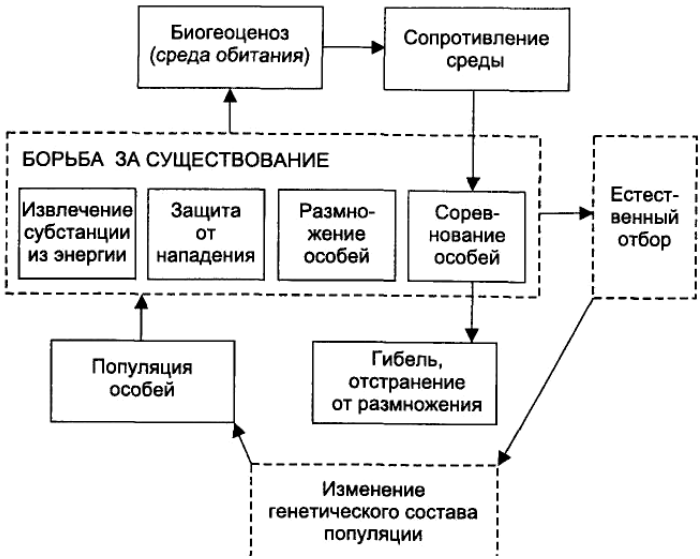


Рис. 3.1. Схема взаимодействия факторов эволюции

ребора вариантов при решении оптимизационных задач [14, 26]. Компьютерные реализации этого механизма получили название «генетические алгоритмы». В 1970-х гг. в рамках теории случайного поиска Л.А. Растригиным был предложен ряд алгоритмов, использующих идеи бионического поведения особей [10]. Развитие этих идей нашло отражение в цикле работ И.Л. Букатовой по эволюционному моделированию [2, 3]. Идеи М.Л. Цетлина, развитые в исследованиях поведения сообществ конечных автоматов, легли в основу алгоритмов поиска глобального экстремума, основанных на моделировании процессов развития и элиминации особей [15]. Большой вклад в развитие эволюционного программирования внесли работы Л. Фогеля, А. Оуэнса и М. Уолша [12, 20, 21].

К основным направлениям развития эволюционного моделирования на современном этапе относятся следующие:

– генетические алгоритмы (ГА), предназначенные для оптимизации функций дискретных переменных и использующие аналогии естественных процессов рекомбинации и селекции;

– классифицирующие системы (КС), созданные на основе генетических алгоритмов, которые используются как обучаемые системы управления;

– генетическое программирование (ГП), основанное на использовании эволюционных методов для оптимизации создаваемых компьютерных программ;

– эволюционное программирование (ЭП), ориентированное на оптимизацию непрерывных функций без использования рекомбинаций;

– эволюционные стратегии (ЭвС), ориентированные на оптимизацию непрерывных функций с использованием рекомбинаций.

Эволюционные методы целесообразно использовать в тех случаях, когда прикладную задачу сложно сформулировать в виде, позволяющем найти аналитическое решение, или тогда, когда требуется быстро найти приближенный результат, например, при управлении системами в реальном времени.

В России развитием эволюционных методов занимаются научные школы профессоров И.Л. Букатовой [2, 3], Д.И. Батищева [1], В.М. Курейчика [7 – 9] и И.П. Норенкова [6] .

3.1. Генетические алгоритмы

В основе генетических алгоритмов лежат генетика и хромосомная теория эволюции организмов. Хромосомы – это нитевидные структуры, находящиеся в клеточном ядре, которые являются носителями наследственности. Каждая хромосома уникальна морфологически и ге-

нетически и не может быть заменена другой либо восстановлена при утере (при потере хромосомы клетка, как правило, погибает). Каждый биологический вид имеет определённое, постоянное количество хромосом. Каждая клетка содержит удвоенный набор морфологически и генетически сходных хромосом. Например, в клетках человека содержится 23 пары хромосом, в клетках комара – 3.

На процесс наследования признаков существенно влияет поведение хромосом при делении клеток. Существует митозное и мейозное деление клеток. Митозное деление обеспечивает распределение исходных хромосом между двумя образующимися дочерними клетками, которые будут иметь равноценные наборы хромосом и будут очень похожи друг на друга. При этом происходит редупликация исходных хромосом, вследствие чего к моменту деления клетки каждая хромосома состоит из двух копий исходной материнской хромосомы – сестринских хроматид (рис. 3.2).

Во время мейоза происходит два последовательных деления: редукционное и эквационное. Мейоз приводит к образованию клеток, у которых число хромосом вдвое меньше по сравнению с исходной клеткой.

В фазе редукции хроматиды обмениваются генами, т.е. участками дезоксирибонуклеиновой кислоты (ДНК). После этого клетка разделяется на две новые, причём каждая из них содержит удвоенный набор хромосом, структуры которых отличаются от исходных. Механизм обмена генами называется кроссинговером.

В результате эквационного деления из двух получившихся клеток образуются четыре клетки, каждая из которых содержит одиночный набор хромосом (рис. 3.2).

Таким образом, митоз обеспечивает возобновление клеток, а мейоз отвечает за передачу наследственной информации и способствует генетическому разнообразию организмов данного вида.

Классическая генетика обосновала наследственность и изменчивость благодаря созданию фундаментальной теории гена, основные положения которой формулируются следующим образом:

– все признаки организма определяются наборами генов;

– гены – это элементарные единицы наследственной информации, которые находятся в хромосомах;

– гены могут изменяться – мутировать;

– мутации отдельных генов приводят к изменению отдельных элементарных признаков организма, или фенов.

Ген определяется как структурная единица наследственной информации, далее неделимая в функциональном отношении. Он представляет собой участок молекулы ДНК, на котором сохраняется постоянный порядок следования пар нуклеотидов. Комплекс генов, содер-

жащихся в наборе хромосом одного организма, образует геном. Роль молекул ДНК, обладающих уникальной способностью к самовоспроизведению, заключается в хранении и передаче генетической информации последующим поколениям.

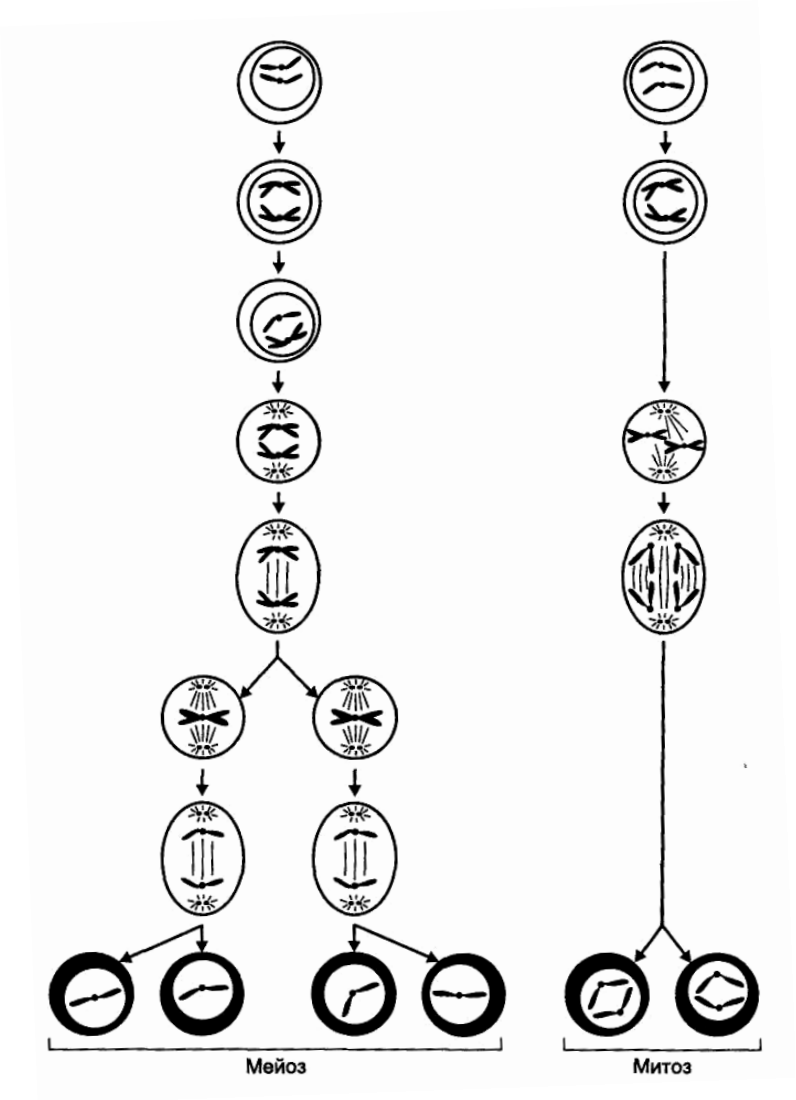


Рис. 3.2. Механизмы деления клеток

В задачах поиска оптимальных решений каждое решение из множества возможных можно представить набором информации, который может быть изменён путём введения в него элементов другого решения. Другими словами, возможные решения соответствуют хромосомам, состоящим из генов, причём в ходе оптимизации происходит обмен генами между хромосомами (рекомбинация). При построении генетических алгоритмов важен выбор принципа генетической рекомбинации. Существует несколько типов перераспределения наследственных факторов:

1) рекомбинация хромосомных и нехромосомных генов;
2) рекомбинация целевых негомологических хромосом;
3) рекомбинация участков хромосом, представленных непрерывными молекулами ДНК.

Для построения генетических алгоритмов наибольший интерес представляет третий тип рекомбинации, который используется для накопления в конечном решении лучших функциональных признаков, какие имелись в наборе исходных решений. Существует несколько типов рекомбинации участков хромосом: кроссинговер, сайт, иллегальная рекомбинация.

Кроссинговер соответствует регулярной рекомбинации, при которой происходит обмен определёнными участками между гомологичными хромосомами. Он приводит к появлению нового сочетания сцепленных генов.

Сайт – это вид рекомбинации, при которой на коротких специализированных участках хромосом происходит обмен генофоров (генных носителей), часто различных по объёму и составу генетической информации.

Иллегальная рекомбинация допускает негомологичные обмены, к которым относятся транслокации, инверсии и случаи неравного кроссинговера. Такие способы могут оказаться полезными при генерации новых решений.

В генетических алгоритмах наибольшее распространение получила операция кроссинговера, заключающаяся в разрыве гомологичных хроматид с последующим соединением их в новом сочетании. Схема кроссинговера, демонстрирующая образование двух новых хромосом после обмена генетическим материалом, приведена на рис. 3.3.

Основная цель кроссинговера заключается в создании из имеющегося генетического материала желаемой комбинации признаков в одном решении.

Кроссинговер может происходить в нескольких точках. Пример двойного кроссинговера между хромосомами приведён на рис. 3.4.

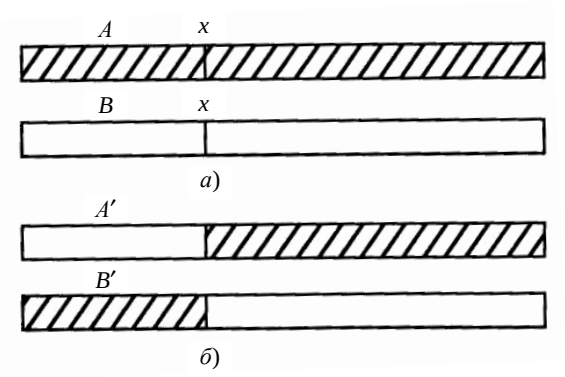


Рис. 3.3. Схема кроссинговера:
а – родительские хромосомы *A*, *B* до кроссинговера;
б – хромосомы-потомки *A'*, *B'* после кроссинговера

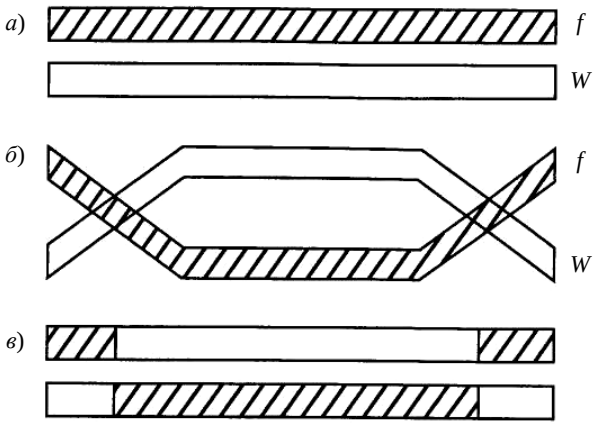


Рис. 3.4. Схема двойного кроссинговера:
а – до кроссинговера; *б* – во время кроссинговера; *в* – после кроссинговера

Помимо кроссинговера для решения различных прикладных задач полезными являются такие генетические операции, как мутация, инверсия, транслокация, селекция (инбридинг и гибридизация), генная инженерия.

Под мутацией понимается генетическое изменение, приводящее к качественно новому проявлению основных свойств генетического материала: дискретности, непрерывности или линейности. Свойство дис-

кретности позволяет выделить в исходном генетическом материале отдельные фрагменты, контролирующие те или иные функции. Непрерывность означает, что определённые комбинации генов совместно контролируют некоторую функцию. Линейность проявляется в определённой последовательности генов в пределах группы сцепления.

Процессы мутации ведут к получению более разнообразного генетического материала. В связи с этим применение операции мутации в генетических алгоритмах направлено на получение решений, которые не могут быть улучшены качественно посредством кроссинговера.

Инверсия, транслокация, транспозиция, делеция и дупликация относятся к разновидностям хромосомной мутации. При инверсии участок хромосомы поворачивается на 180°. Транслокацией называют перенос части одной хромосомы в другую. При перемещении небольших участков генетического материала в пределах одной хромосомы используют термин транспозиция. Делеция – это выпадение отдельных участков хромосом, дупликация – повторение участка генетического материала. Кроме перечисленных, существуют другие разновидности хромосомных мутаций [7].

Селекция представляет собой форму искусственного отбора, который может быть массовым или индивидуальным. Установлено, что массовый отбор по фенотипу (совокупности всех внешних и внутренних признаков) менее эффективен, чем индивидуальный, когда популяцию делят на отдельные линии, а для размножения выбирают носителей желаемых свойств. Применение процедуры селекции в генетических алгоритмах оптимизации способствует ускорению процесса синтеза искомого решения.

Генная инженерия представляет собой совокупность методов для получения рекомбинантной ДНК и операции над нею. Рекомбинантная ДНК получается путём объединения фрагментов ДНК различных организмов. Использование подходов генной инженерии позволяет в ряде задач значительно быстрее находить желаемое решение.

Механизм эволюции основан на трёх повторяющихся процессах: отборе, амплификации (процесс производства потомков) и мутации. Он используется в качестве механизма случайно направленного комбинаторного перебора при решении задач оптимизации и слабоструктурированных проблем принятия решений.

Генетический алгоритм – это поисковый алгоритм, основанный на природных механизмах селекции и генетики. Эти алгоритмы обеспечивают выживание сильнейших решений из множества сгенерированных, формируя и изменяя процесс поиска на основе моделирования эволюции исходной популяции решений. Генетические алгоритмы сконструированы таким образом, что при генерации каждой новой по-

пуляции используются фрагменты исходных решений, к которым добавляются новые элементы, обеспечивающие улучшение решений относительно сформулированного критерия отбора. Другими словами, генетические алгоритмы используют информацию, накопленную в процессе эволюции [2, 3, 7, 8, 16, 22 – 26].

В генетических алгоритмах используются специфические термины, взятые из генетики, которые трактуются следующим образом:

Генетика	Генетические алгоритмы
Хромосома	Решение, стринг, строка, последовательность, родитель, потомок
Популяция	Набор решений (хромосом)
Локус	Местоположение гена в хромосоме
Поколение	Цикл работы генетического алгоритма, в процессе которого сгенерировано множество решений
Ген	Элемент, характеристика, особенная черта, свойство, детектор
Аллель	Значение элемента, характеристики
Фенотип	Структура
Эпистасис	Множество параметров, альтернативные решения
Скрещивание, рекомбинация, кроссинговер	Оператор рекомбинации
Мутация	Оператор модификации

При разработке генетических алгоритмов преследуются две главные цели:

– абстрактное и формальное объяснение процессов адаптации в естественных системах;

– проектирование искусственных программных систем, воспроизводящих механизмы функционирования естественных систем.

Основные отличия ГА от других алгоритмов оптимизации:

– используются не параметры, а закодированные множества параметров;

– поиск осуществляется не из единственной точки, а из популяции точек;

– в процессе поиска используются значения целевой функции, а не её приращения;

– применяются вероятностные, а не детерминированные правила поиска и генерации решений;

– выполняется одновременный анализ различных областей пространства решений, в связи с чем возможно нахождение новых областей с лучшими значениями целевой функции за счёт объединения квазиоптимальных решений из разных популяций.

Согласно репродуктивному плану Холланда [14, 26] генетические схемы поиска оптимальных решений включают следующие этапы процесса эволюции:

1. Конструируется начальная популяция. Вводится начальная точка отсчёта поколений $t = 0$. Вычисляются приспособленность хромосом популяции (целевая функция) и средняя приспособленность всей популяции.

2. Устанавливается значение $t = t + 1$. Выбираются два родителя (хромосомы) для кроссинговера. Выбор осуществляется случайным образом пропорционально жизнеспособности хромосом, которая характеризуется значениями целевой функции.

3. Формируется генотип потомка. Для этого с заданной вероятностью над генотипами выбранных хромосом производится операция кроссинговера. Случайным образом выбирается один из потомков $A(t)$, который сохраняется как член новой популяции. Далее к потомку $A(t)$ последовательно с заданными вероятностями применяются операторы инверсии и мутации. Полученный в результате генотип потомка сохраняется как $A'(t)$.

4. Обновление текущей популяции путём замены случайно выбранной хромосомы $A'(t)$.

5. Определение приспособленности $A'(t)$ и пересчёт средней приспособленности популяции.

6. Если $t = t^*$, где t^* – заданное число шагов, то переход к этапу 7, в противном случае – переход к этапу 2.

7. Конец работы.

Основная идея эволюции, заложенная в различные конструкции генетических алгоритмов, проявляется в способности «лучших» хромосом оказывать большее влияние на состав новой популяции за счёт длительного выживания и более многочисленного потомства.

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 целе-

3.1. Анализ начальной популяции на первом шаге простого генетического алгоритма

Номер хромосомы	Двоичный код хромосомы	Значение x (десятичный код)	Значение целевой функции $f(x)$	Нормированное значение $f(x)/\mathrm{sum}f(x)$	Ожидаемое количество копий хромосомы в следующем поколении	Реальное количество копий хромосомы в следующем поколении
1	01101	13	169	0,14	0,56	1
2	11000	24	576	0,49	1,96	2
3	01000	8	64	0,06	0,24	0
4	10011	19	361	0,31	1,24	1
Суммарная целевая функция			1170	1,00	3,00	4
Среднее значение целевой функции			293	0,25	1,00	1
Максимальное значение целевой функции			576	0,49	1,97	2

вой функции для каждой хромосомы определяется путём возведения в квадрат значения двоичного числа, кодирующего решение *x*. 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

$$P_i = \frac{f_i(x)}{\text{sum}\, f(x)},$$

где $f_i(x)$ – значение целевой функции *i*-й хромосомы в популяции; $\text{sum}\, f(x)$ – суммарное значение целевой функции всех хромосом в популяции.

Ожидаемое число копий *i*-й хромосомы после оператора репродукции равно

$$N = nP_i,$$

где *n* – число анализируемых хромосом.

Число копий хромосомы, переходящих в следующее поколение, определяют по формуле

$$A_i = \frac{f_i(x)}{f_{\text{ср}}(x)},$$

где $f_{\text{ср}}(x)$ – среднее значение целевой функции.

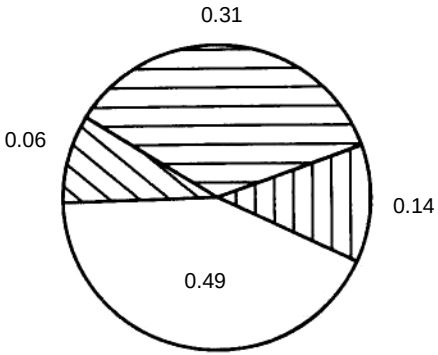


Рис. 3.5. Колесо рулетки

3.2. Результаты операций репродукции и кроссинговера в простом генетическом алгоритме

Номер хромосомы	Популяция после репродукции	Случайно выбранные пары	Точка разрыва кроссинговера	Популяция после кроссинговера	Значение x (десятичный код)	Значение $f(x)$
1	0110|1	1–2	4	01100	12	144
2	1100|0	1–2	4	11001	25	625
3	11|000	3–4	2	11011	27	729
4	10|011	3–4	2	10000	16	256

Суммарное значение целевой функции $\text{sum}\, f(x) = 1754$

Среднее значение целевой функции $f_{\text{ср}}(x) = 439$

Максимальное значение целевой функции $f(x) = 729$

Значение N для первой хромосомы будет равно $0{,}14 \times 4 = 0{,}56$ копий, для второй – $0{,}49 \times 4 = 1{,}96$ копий, для третьей – $0{,}06 \times 4 = 0{,}24$ и для четвёртой – $0{,}31 \times 4 = 1{,}23$. 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

$$\text{родители}\begin{cases}\text{хромосома 1: } 0110\,|\,1\\ \text{хромосома 2: } 1100\,|\,1,\end{cases}$$

а после применения оператора кроссинговера получаем описание

$$\text{потомки}\begin{cases}\text{хромосома 1: } 0110\,|\,0\\ \text{хромосома 2: } 1100\,|\,1.\end{cases}$$

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 $A=\{a_1, a_2, a_3, ..., a_{L-2}, a_{L-1}, a_L\}$ случайным образом определяют две позиции, например, 2 и $L-1$.

Этап 2. Гены, соответствующие выбранным позициям, меняют местами и формируют новую хромосому $A=\{a_1, a_{L-1}, a_3, ..., a_{L-2}, a_2, a_L\}$.

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 $f(x) = 729$ и применим операцию мутации к позициям 3 и 4:

$$\text{хромосома 3: } 11011 \rightarrow \text{хромосома 3': } 11101.$$

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 $\rightarrow$ 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 $P(CO) < 1.0$.

5. Вероятность оператора мутации принимается равной $P(MO) > 0.001$.

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 ал-

горитма 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) $B = \{b_1, b_2, ..., b_L\}$ выбирается случайным образом из текущей популяции.

2. Из множества $Y = \{0, 1, 2, ..., L + 1\}$ случайным образом выбираются два числа y_1 и y_2 и определяются значения $x_1 = \min\{y_1, y_2\}$ и $x_2 = \max\{y_1, y_2\}$.

3. Из хромосомы B формируется новая хромосома путём инверсии (обратного порядка) сегмента, лежащего справа от позиции x_1 и слева от позиции x_2 в хромосоме B. После применения оператора инверсии строка B примет вид $B' = \{b_1, b_{x1}, b_{x2-1}, b_{x2-2}, b_{x1+1}, b_{x2}, ..., b_L\}$.

Например, для строки $B = \{1, 2, 3, 4, 5, 6\}$ при выборе $y_1 = 6$ и $y_2 = 2$ и соответственно $x_1 = 2$, $x_2 = 6$ 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 3^L схем (шаблонов), но только 2^L различных альтернативных стрингов. Это следует из факта, что схеме (**) в общем случае могут соответствовать $3^2 = 9$ стрингов, а именно {**,*1,*0,1*, 0*,00,01,1011}, и только $2^2 = 4$ 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 S_j, $j = 1, 2, .., n$, 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 M существует $(M + 1)^L$ схем и $n2^L$ схем, содержащихся в популяции размера n, поскольку стринг представляется двумя схемами.

Для количественной оценки схем введём две характеристики: порядок схемы $O(H)$ 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 H, содержащихся в популяции $S(t)$, которые определяют возможное число различных схем H при заданном t, т.е. $m = m\{H,t\}$.

В процессе репродукции вероятность попадания хромосомы S_i в репродуцированное множество равна $P_i = \dfrac{f_i(x)}{\text{sum } f(x)}$, т.е. зависит от значения целевой функции. За время $t + 1$ в популяции $S(t)$ ожидается получить $m(H, t + 1)$ представителей схемы H, которое вычисляется по формуле

$$m(H,\ t+1) = m(H,t)\ n\frac{f\ (H)}{\text{sun } f\ (x)},$$

где $f(H)$ – среднее значение целевой функции хромосом, представленных схемой H за время t.

Так как среднее значение целевой функции для всей популяции равно

$$f_{\text{ср}}\ (S) = \text{sun } f\ (x)/n,\ \text{ то } m(H,\ t+1) = m(H,t)\frac{f\ (H)}{f_{\text{ср}}(S)}.$$

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 $f_{\text{ср}}(S)$, 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 $f(H) \geq f_{\text{ср}}(S)$. Тогда значение целевой функции для схемы H можно выразить через среднее значение для всей популяции, например, следующим образом: $(1+c) f_{\text{ср}}(S)$, где c – константа. Число представителей схемы в следующем поколении будет

$$m(H,\ t+1) = m(H,t)\frac{(1+c)\ f_{\text{ср}}\ (S)}{f_{\text{ср}}(S)} = (1+c)\ m(H,t).$$

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 ($c > 0$) или уменьшению ($c < 0$) числа схем.

Лемма. Если на некотором шаге генетического алгоритма P_1 есть вероятность того, что хромосома A порождает потомка, и P_2 есть вероятность, что A уничтожается, то ожидаемое число потомков хромосомы A равно P_1 / P_2 [26].

Вероятность выживания хромосомы A на шаге t после операции репродукции определяется по формуле $P_S(t) = (1-P_2)^{t-1} P_2$, 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

$$P_S(H) = 1 - \frac{O(H)}{L-1},$$

где $O(H)$ – порядок схемы; L – длина стринга.

Если оператор кроссинговера выполняется на основе случайного выбора с вероятностью $P(CO)$, то вероятность выживания схемы определяется по формуле

$$P_S(H) \geq 1 - \frac{P(CO)\, L(H)}{L-1},$$

где $L(H)$ – определённая длина схемы.

Приведённое выражение свидетельствует о том, что вероятность выживания схемы уменьшается при возрастании $P(CO)$.

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

$$m(H, t+1) \geq m(H,t) \frac{f(H)}{f_{\text{ср}}(S)} \left[1 - P(CO) \frac{L(H)}{L-1} \right].$$

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 $1 - P(MO)$, где $P(MO)$ – 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 $P(MO) << 1$ вероятность выживания схемы при мутации может быть представлена выражением [23]: $P_S(MO) = 1 - L(H)\,P(MO)$.

С учётом вышесказанного и согласно [26] для частной схемы H можно определить ожидаемое число копий в следующей генерации после реализации операторов репродукции, кроссинговера и мутации по следующей формуле:

$$m(H,\, t+1) \geq m(H,t)\frac{f(H)}{f_{\text{ср}}(S)}\left[1 - P(CO)\frac{L(H)}{L-1} - L(H)\,P(MO)\right].$$

Данная формула отражает фундаментальную теорему генетического алгоритма, которая определяет асимптотическое число схем, выживающих при его реализации на каждой итерации. Наиболее существенное влияние на число выживающих схем оказывают значения целевых функций отдельной схемы и всей популяции, а эффективность реализации генетических алгоритмов зависит от размера строительных блоков.

Примеры практического применения генетических алгоритмов. Генетические алгоритмы нашли широкое практическое применение в менеджменте и управлении [13] для решения задач поиска оптимальных решений, формирования моделей и прогнозирования значений различных показателей. Они осуществляют поиск лучших решений на основе заданной целевой функции. Значение целевой функции для многих задач весьма непросто вычислить, поэтому в ряде случаев при исследовании плохо обусловленных проблем с этой целью применяются нейронные сети, позволяющие найти решение при отсутствии явной модели. Кроме того, для вычисления целевых функций в условиях неопределённости применяются статистические методы и методы логического вывода в чёткой или нечёткой среде.

Формирование системы прогнозирующих правил. Генетические алгоритмы могут использоваться для нахождения оптимального набора правил, позволяющих прогнозировать страховые риски с учётом ряда определяющих его факторов [13]. Для решения этой задачи необходимо иметь базу данных, содержащую фактические значения переменных, влияющих на страховой риск.

Рассмотрим пример использования генетического алгоритма для оптимизации экспертных правил в сфере страхования.

Допустим, что компания, занимающаяся страхованием автомобилей, использует базу данных, которая помимо прочих включает следующие факторы: максимальную скорость автомобиля (км/час), возраст автомобиля (лет), возраст водителя (лет) и риск, определённый экспертно по некоторой шкале на основе анализа обращений клиентов о выплате компенсации по страховым случаям. Правила, задающие оценку страхового риска, сконструированы в виде:

ЕСЛИ максимальная скорость автомобиля лежит в диапазоне [A_l] И возрастной диапазон автомобиля [B_l] И возраст водителя находится в диапазоне [C_l], ТО страховой риск имеет значение [D_l].

Для конкретной выборки из БД это правило может иметь следующий вид:

ЕСЛИ максимальная скорость [91…100 км/час] И возраст автомобиля [11 – 15 лет] И возраст водителя [31 – 40 лет], ТО риск [3]. Здесь уровень риска отображается на интервал [1, 5], при этом высокие значения соответствуют большим страховым рискам.

Подобные правила, основанные на фактических значениях переменных, случайным образом выбранных из БД, составляют исходную популяцию. Для каждой из переменных, входящих в популяцию, предварительно задаётся диапазон состояний. Например, переменная «возраст автомобиля», может иметь пять возможных состояний: 1 – 5, 6 – 10, 11 – 15, 16 – 20, 21 – 25 лет. Далее сформированная популяция обрабатывается генетическими операторами с учётом специфики рассматриваемой задачи. Целевая функция должна показывать, насколько точно сгенерированные правила описывают реальные страховые случаи, хранящиеся в БД. Например, если какое-то правило описывает 4 случая из 5, то значение целевой функции будет 4/5, или 80%.

Новые члены популяции образуются в результате скрещивания и мутации начального набора правил. В данном случае при скрещивании двух правил происходит обмен парами «атрибут–значение» на участке строки после точки кроссинговера. В результате образуются два новых правила, жизнеспособность которых оценивается по тому, насколько удачно они описывают страховые случаи, которые имели место в прошлом. Мутация правил обеспечивает необходимое разнообразие признаков и заключается в изменении значений атрибутов с заданной вероятностью. Таким образом, первоначально сформированный набор правил преобразуется случайно направленным способом в другой набор, который лучше остальных описывает накопленную статистику страховых случаев. Результирующая система правил в дальнейшем используется для прогнозирования страховых рисков.

Следует отметить, что подобный подход к формированию системы правил может приводить к некорректным правилам продукций. В то же время он освобождает разработчиков и экспертов от трудоёмкой работы по формулированию и оценке правил, так как некорректные результаты отбрасываются при сопоставлении сгенерированных продукций с реальными страховыми ситуациями. Привлечение прошлого опыта для оценки пригодности прогнозирующих правил не позволяет предвидеть новые ситуации, которые не имели места в прошлом. Поэтому при решении задач описанным способом очень важно следить за своевременным пополнением и модификацией информации в БД, которая отражает появление новых фактов, атрибутов и тенденций.

Классифицирующие системы. На основе генетических алгоритмов Дж. Холланд предложил классифицирующие системы, которые можно использовать для целей управления [11, 14, 17, 30]. Классифицирующая система состоит из трёх вложенных друг в друга подсистем (рис. 3.6): классификатора, системы обучения и генетического алгоритма. В классификатор поступают внешние сообщения и положительные оценки (поощрения) его действий. Классификатор содержит правила вида ЕСЛИ<условие>, ТО<сообщение>, с помощью которых формируются выходные сообщения. Обучающая система выполняет оценку используемых правил. Генетический алгоритм предназначен для случайно направленной модификации правил. Схема обработки правил представлена на рис. 3.7.

Каждому правилу приписывается численная оценка силы правила. Сообщения и условные части правил (антецеденты) формулируются в одних и тех же терминах. Список сообщений содержит все теку-

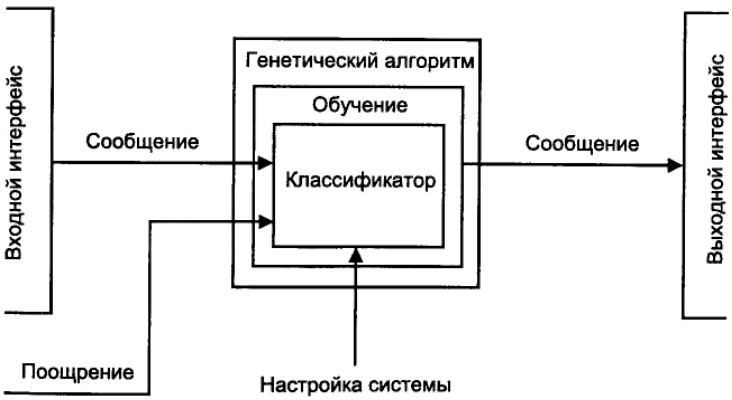


Рис. 3.6. Схема классифицирующей системы

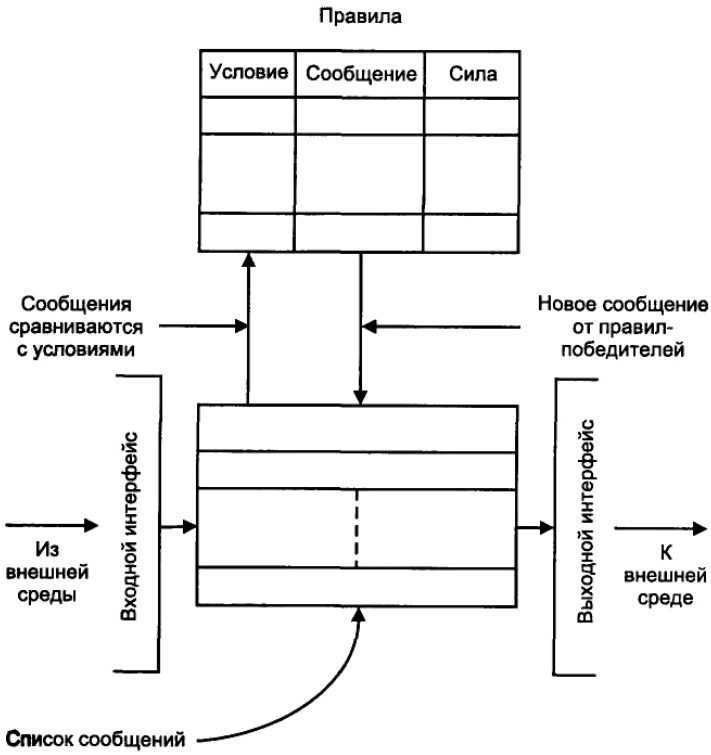


Рис. 3.7. Схема обработки правил в классифицирующей системе

щие сообщения – поступающие из внешней среды и те, что формируются внутри системы. В процессе работы КС все сообщения из списка сравниваются с условиями всех правил. Классификатор выполняет следующие действия.

Шаг 1. В список сообщений (рабочую память) добавляются все сообщения, поступившие извне.

Шаг 2. Проводится сравнение всех сообщений из списка с антецедентами всех правил. Все правила, антецеденты которых совпадают с присутствующими в рабочей памяти сообщениями, записываются в список правил *M*.

Шаг 3. Выполняются правила из списка *M*, при этом сообщения каждого правила посылаются в список новых сообщений.

Шаг 4. Обновление списка сообщений.

Шаг 5. Сообщения из списка посылаются в выходной интерфейс.

Вероятность выдачи сообщения зависит от силы правила: не каждое сообщение выдаётся на управляемый объект, часть их может быть связана с изменением внутренней структуры системы (правил).

Шаг 6. Возврат к шагу 1.

В процессе обучения каждому правилу присваивается численное значение силы, а алгоритм обучения регулирует это значение с учётом полезности правила для системы. На шаге 3 описанного алгоритма для каждого отобранного правила C вычисляется цена по формуле $B(C,t) = bR(C)s(C,t)$, где $s(C, t)$ – сила правила C в момент t; $R(C)$ – специфичность условия в правиле, равная числу символов, отличающихся от символа * в условии, делённому на длину условия; b – коэффициент, который обычно принимают равным 1/8 или 1/6.

Цена B определяет вероятность того, что правило пошлёт сообщение в список новых сообщений. Вероятностный подход позволяет аутсайдерам тоже изредка посылать сообщения, что при благоприятных условиях может сделать их лидерами.

Послать сообщение могут все правила с допустимым значением B, т.е. такие, у которых B превышает определённый порог. Правило, пославшее сообщение в новый список, расплачивается за это уменьшением своей силы:

$$s(C, t+1) = s(C, t) - B(C, t).$$

Для правил C, пославших сообщения, которые на следующем шаге работы оказались полезными (совпали с условиями правила-победителя, имеющего высокую цену), оценка силы возрастает на долю B:

$$s(C', t+1) = s(C', t) + aB(C, t),$$

где $a = 1/K$, K – число правил C', т.е. каждый поставщик получает равную долю B.

Правило полезно только тогда, когда его потребители в своих локальных действиях тоже получают выигрыш. В противном случае правило обесценивается, так как его цена s уменьшается при отсылке сообщения. В свою очередь, полезность потребителей зависит от их потребителей и т.д. Цепочка приводит к конечным потребителям, достигающим цели и получающим поощрения от внешней среды.

Классификатор и обучающая система не порождают новых правил. Эту функцию выполняет генетический алгоритм, который работает с учётом силы правил, определённой в системе обучения. Работа генетического алгоритма рассмотрена в предшествующем примере.

Комбинированные методы и интеллектуальные системы. В настоящее время активно развиваются методы, основанные на объединении технологий инженерии знаний и генетических алгоритмов. В области ГА разрабатываются операторы, ориентированные на обработку знаний.

Генетические алгоритмы используют в теории нечётких систем для настройки параметров функций принадлежности. Интеграция чётких и нечётких нейронных сетей и генетических алгоритмов обеспечивает реализацию оптимизационной задачи. Средства fuzzy–neuro–genetic используются в интеллектуальных системах и содержат следующие процедуры:

– преобразование входных примеров в нечёткое представление;

– извлечение знаний, представленных в виде продукций ЕСЛИ–ТО из нечёткой обучающей выборки с помощью нейронной сети;

– оптимизацию структуры продукционных правил с помощью генетического алгоритма.

Активно развивается направление, ориентированное на использование генетических алгоритмов для обучения нейронных сетей [5] и корректировки структуры уже обученной сети [18]. В отличие от метода обратного распространения ошибки генетические алгоритмы мало чувствительны к архитектуре сети. Напомним, что основными характеристиками нейронной сети являются следующие:

– HLN – количество скрытых слоёв;

– N_k – число нейронов в каждом слое;

– w_{ij} – весовые коэффициенты межнейронных связей;

– $F_j(X, W)$ – передаточные функции нейронов скрытых слоёв, а также нейронов выходного слоя.

Сформулируем общую задачу оптимизации сети: при заданных количествах входных и выходных нейронов на основе заданного множества обучающих примеров определить оптимальные значения HLN, N_k, $k = 1, ..., HLN$, значения всех весовых коэффициентов межнейронных связей w_{ij}, где j – индекс нейрона; i – индекс межнейронной связи (синапса); $F_j(X, W)$ – передаточные функции всех нейронов за исключением нейронов входного слоя. Критерием оптимизации является максимальное отклонение выходного вектора сети Y' от эталонного значения выхода Y, полученное в результате обработки всех примеров, т.е. необходимо найти

$$\delta^* = \min_{\substack{HLN,\, N_k,\\ W,\, F_j(X,W)}} \max_{Q} \delta,$$

где $\delta = Y' - Y$; Q – множество обучающих примеров, содержащих значения X, Y; $Y' = F(HLN, N_k, X, W)$; $F(HLN, N_k, X, W)$ – передаточная функция ИНС, которая строится на основе частных функций отдельных нейронов $F_j(X, W)$.

Даже для простых сетей эта задача является очень сложной, поэтому для её решения применяется декомпозиция, т.е. сеть оптимизируется в процессе последовательного решения частных задач оптимизации. Например, на первом шаге подбираются оптимальные значения HLN и N_k, затем определяется оптимальный вид передаточных функций нейронов, а на конечной стадии подбираются веса межнейронных связей.

Генетические алгоритмы чаще всего применяются для улучшения характеристик ИНС, уже созданных и обученных с применением других методов.

Краткий обзор программных средств. Коммерческое программное обеспечение, реализующее генетические алгоритмы, можно разделить на программные средства общего назначения, прикладные и алгоритмические программные продукты.

Программное обеспечение общего назначения включает разнообразные наборы инструментальных средств для построения конкретных программ, которые содержат библиотеки алгоритмов, программы моделирования, средства визуализации и другие инструменты. Пакеты подобного типа рассчитаны на опытных программистов, требуют знания основ теории эволюционных вычислений и характеризуются высокой трудоёмкостью освоения, которая в значительной мере зависит от квалификации пользователя.

Прикладные программные продукты ориентированы на решение проблем определённого класса в конкретных предметных областях (реинжиниринг, маркетинг, стратегическое планирование и др.). Такие средства не требуют от пользователя теоретических знаний в области методологии создания интеллектуальных систем. Достаточно, чтобы он был специалистом в своей предметной области.

Алгоритмическое программное обеспечение поддерживает один (или несколько) генетический алгоритм. Преимущества таких программных продуктов – их гибкость и простота использования. При этом пользователям необходимо иметь представление об основах теории ГА.

Перечислим некоторые популярные программные средства [13], реализующие технологии оптимизации с применением генетических алгоритмов и дадим краткую характеристику.

Система PC/Beagle представляет собой программу поиска решающих правил, классифицирующих примеры из базы данных. Она превращает данные в знания за счёт использования машинного обучения. Один из модулей системы путём репродукции и селекции порождает правила, представленные в виде логических выражений.

Система Evolver реализует шесть методов генетической оптимизации и выполнена в виде расширения MS Excel. Основные области применения пакета – оптимизация доходности с учётом уровня риска и максимизация прибыли с учётом возможных издержек.

Genesis – известный алгоритмический программный продукт, который используется в качестве инструмента тестирования генетических алгоритмов. Он позволяет создать модифицированную программную среду и обеспечивает пользователя статистической информацией на выходе.

Программный продукт общего назначения EnGENEer помогает адаптировать генетические алгоритмы к новым проблемным областям за счёт использования следующих инструментов:

– специального языка, предназначенного для описания структурных понятий генетики (генов, хромосом и т.д.);

– эволюционного модельного языка, используемого для отображения таких атрибутов, как размер популяции, типы скрещивания и мутации;

– графических инструментов мониторинга;

– библиотеки инструкций.

Объектно-ориентированная среда Game содержит пять основных частей:

– виртуальную машину;

– высокоуровневый генетический язык;

– библиотеку генетических алгоритмов;

– графический монитор;

– компилятор.

Система спроектирована так, что допускает параллельное использование нескольких алгоритмов. Для создания конкретного приложения используются библиотечные модули, из которых строится макропрограмма с помощью специального высокоуровневого языка.

Известный дистрибьютер программного обеспечения фирма «Тора–Инфо–Центр» распространяет пакет Gene Hunter, который может использоваться как приложение MS Excel и допускает составление собственных программ на языках C и Visual Basic.

3.2. Методы эволюционного программирования

Генетическое программирование. Методы генетического программирования были разработаны в начале 1990-х гг. Дж. Козой [27 – 29]. Генетическое программирование – это способ создания компьютерных программ для задач с неизвестным алгоритмом решения. Объектом эволюции является программа, а популяция содержит множество различных программ. Совершенствование объекта осуществляется на основе отбора в соответствии с определённой функцией ценности (fitness function). Программы строятся из блоков, которые представляют собой примитивные функции и терминалы. В качестве примитивных функций обычно рассматриваются арифметические и логические операции, математические функции и функции специального вида, характерные для класса решаемых задач. Множество терминалов содержит разнообразные данные, не создаваемые программой. Цель состоит в построении наилучшей программы в пространстве программ, которые могут быть составлены из заданных функций и терминалов с учётом определённых правил синтаксиса.

Технология генетического программирования включает следующие этапы.

Этап 1. Формирование множества терминалов, множества примитивных функций, синтаксических правил и критериев оценки создаваемых программ.

Этап 2. На основе закона случайности создаётся начальная популяция компьютерных программ, ориентированных на решение поставленной задачи.

Этап 3. Каждая программа выполняется, а результаты её работы оцениваются с помощью fitness function (целевой функции).

Этап 4. Формируется новая популяция программ, в которую сгенерированные программы могут попасть с вероятностью, пропорциональной значению целевой функции.

Этап 5. Реализуются генетические операторы репродукции, скрещивания и мутации. В результате репродукции осуществляется копирование уже созданных программ с хорошими значениями целевой функции. Оператор скрещивания создаёт новые программы путём комбинирования фрагментов существующих программ. Мутация заключается в замене некоторого фрагмента программы случайно порождённым символьным выражением.

Этап 6. Производится тестирование программ-членов новой популяции и принимается решение о продолжении процесса эволюции. Продолжать генерацию новых популяций имеет смысл тогда, когда максимальные и средние значения целевой функции улучшаются.

Рассмотрим пример [27] модификации программы на языке LISP, где в качестве терминалов используются переменные логического типа D_i, i = 0, 1; а для их обработки применяются логические операции NOT, OR, AND. Пусть на некотором шаге имеется следующее множество допустимых выражений:

NOT(D_1);
NOT(D_0);
AND(D_0, D_1);
OR (AND(D_0, D_1), NOT(D_1));
AND(NOT (D_0), NOT (D_1));
OR (D_1, NOT(D_0));
OR (OR (D_1, NOT(D_0)), AND (NOT (D_0), NOT(D_1))).

Эти выражения можно представить в виде деревьев (рис. 3.9). В процессе эволюции на уровне поддеревьев осуществляется рекомбинация и получаются потомки (рис. 3.10). Первый из этих потомков представляет собой реализацию операции исключающего ИЛИ:

OR (AND (NOT(D_0), NOT (D_1)), AND (D_0, D_1).

Результатом применения оператора мутации является замена части дерева другим выражением, сгенерированным случайным образом. Точка мутации также выбирается случайно.

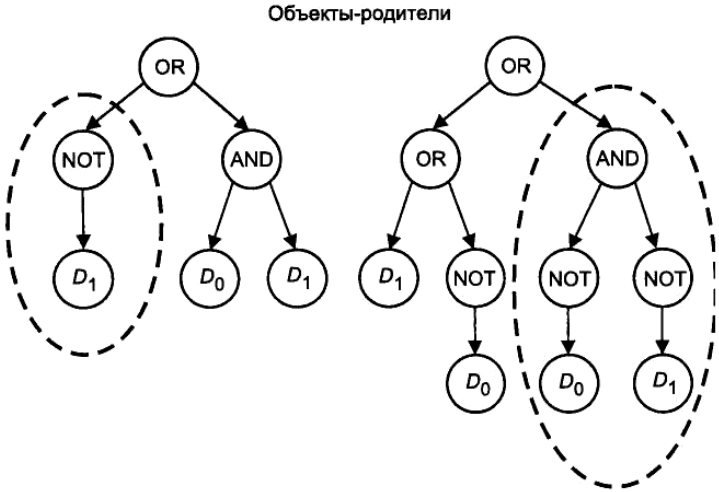


Рис. 3.9. Представление символьных выражений языка LISP в виде деревьев

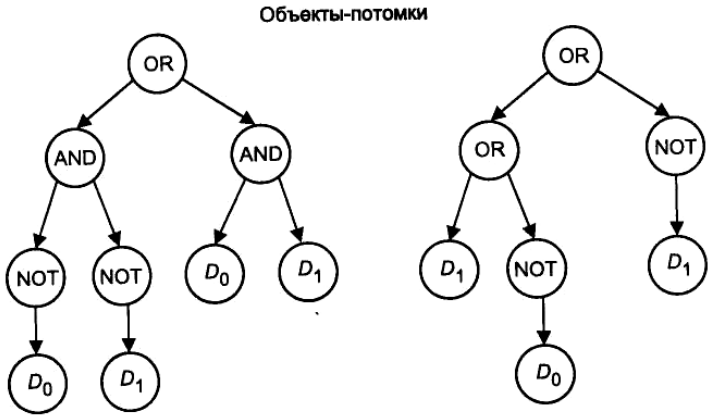


Рис. 3.10. Потомки от скрещивания родителей на уровне поддеревьев

Идеи генетического программирования положены в основу программ, которые называются симуляторами «искусственной жизни». В работе Дж. Козы [27] приводится следующий пример подобной программы. На тороидальной сетке размером 32×32, в 89 ячейках помещается «пища». Существуют некие препятствия, мешающие «насекомым» добраться до «пищи». «Насекомые» попадают на сетку из одной точки, и каждое движется согласно командам своей программы. В начальной популяции эти программы формируются случайным образом из операторов, которые проверяют наличие препятствий и предписывают движение прямо, влево или вправо. Задаётся время жизни популяции (400 шагов). Цена каждой программы определяется числом шагов, которые необходимо совершить, чтобы обойти все ячейки с «пищей». Каждая следующая популяция формируется из предыдущей с помощью генетических операторов репродукции, скрещивания и мутации с учётом ценности программ предыдущей популяции. Решение для популяции из 4000 «насекомых» было найдено за 20 итераций.

Последователи Дж. Козы исследовали в своих работах возможность использования ГП для синтеза сложных автоматов, а также для структурной идентификации динамических систем [19].

В примере построения экономической балансовой модели [13] поставлена цель уточнения эконометрического уравнения обмена, связывающего уровень цен, валовой национальный продукт (ВНП), запас денег и скорость оборота денег в экономике. В качестве терминалов здесь используются следующие переменные: ВНП82 – уровень ВНП за 1982 г.; GD – дефлятор ВНП (выходная переменная модели), нормали-

зованный к единице для 1982 г.; FM – ежемесячная величина запаса денег. Приведённые переменные являются функциями времени. Их значения определяются на основе статистических данных в виде временных рядов. Кроме того, используется множество обобщённых констант действительного типа *R*.

Для обработки переменных предусмотрены следующие операции: сложение (+); вычитание (–); умножение (*); защищённое деление (%), результатом которого является единица при попытке разделить на 0; защищённое логарифмирование (RLOG), дающее 1 при нулевом значении аргумента; вычисление экспоненты (EXP).

Грубая оценка пригодности сгенерированных уравнений вычисляется как сумма квадратов отклонений расчётных значений от фактических в заданных экспериментальных точках:

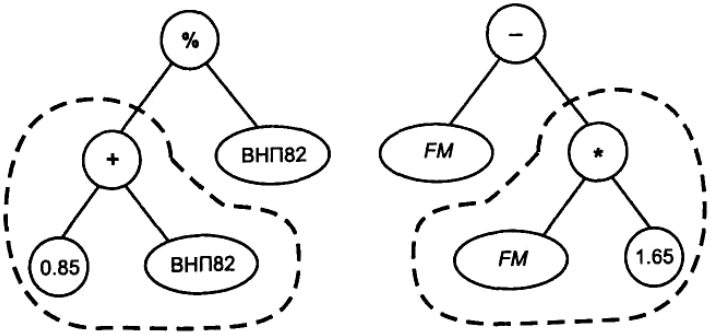
$$R_h(t) = \sum_{j=1}^{N} \left[V_j^h - S_j \right]^2 ,$$

где *Sj* – фактическое значение выходного параметра модели; $j = 1, ..., N$; N – число точек; V_j^h – расчётное значение вычисляемого показателя; h – индекс сгенерированной модели.

Значение $R_h(t)$ масштабируется в целях получения нормированной оценки пригодности $a_h(t) = \dfrac{1}{1 + R_h(t)}$, которая изменяется на интервале [0, 1] и позволяет перейти к задаче максимизации. На основе $a_h(t)$ вычисляется относительная нормированная оценка пригодности $n_h(t) = a_h(t) \Big/ \sum_{m=1}^{M} a_m(t)$, которая имеет более высокие значения для лучших членов популяции и обладает свойством $\sum_{m=1}^{M} n_m(t) = 1$, допускающим вероятностную интерпретацию.

Критерием окончания процесса эволюции является достижение заданного числа генераций (50) или достижение наилучшего значения целевой функции. Численность популяции была принята равной 500. В процессе генерации новых поколений скрещивание проводилось на 90% численности популяции, т.е. из каждого поколения выбиралось 225 пар родителей с вероятностью, равной относительной оценке их пригодности. Кроме того, для каждой новой популяции осуществлялась репродукция 10% лучших представителей поколения.

Генерируемые модели (программы) удобно представить в виде древовидных структур (рис. 3.11).



а) *б*)

Рис. 3.11. Древовидное представление компьютерных моделей, отобранных для скрещивания:
а – родитель 1; *б* – родитель 2

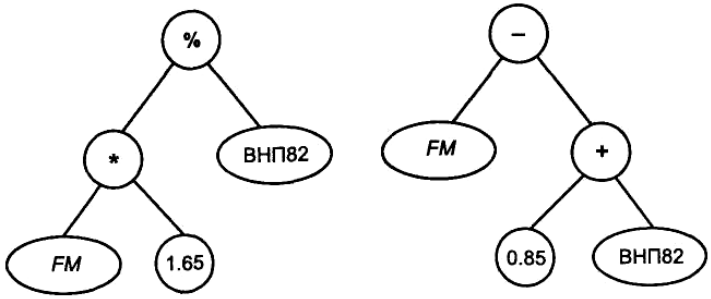


Рис. 3.12. Модели-потомки, полученные в результате скрещивания

Представленные на рис. 3.11 модели соответствуют выражениям V_i = (0,85 + ВНП82)/ВНП82 и V_j = l,65*FM* – *FM*. Операция скрещивания начинается со случайного и независимого выбора точки кроссинговера в каждой из двух моделей-родителей. Отсечённые фрагменты программ-родителей, обозначенные пунктиром, меняются местами, и в результате образуются две модели-потомка (рис. 3.12). Потомкам соответствуют уравнения V_k = 1,65*FM*/ВНП82 и V_l = 0,85 + ВНП82 – *FM*.

Оператор мутации в данном примере выполнялся путём замены функций в узлах деревьев либо путём случайного изменения значений констант.

Используя статистические данные за 30 лет (заметим, что в примере исследовалась экономика США в период с 1959 по 1988 г.), генетический алгоритм за 50 последовательных генераций выдал наилучшее решение GD = 1,656.FM/ВНП82, которое хорошо согласуется с известным эконометрическим уравнением обмена $P = MV/Q$, где P – уровень цен; M – запасы денежной массы; V – скорость обращения денег в экономике; Q – валовой национальный продукт.

Эволюционное программирование. В 1960-х гг. Л. Фогель, А. Оуэне и М. Уолш предложили схему эволюции логических автоматов, решающих задачи предсказания, диагностики, распознавания и классификации образцов, а также задачи управления объектом с неизвестным характером [12]. Исследования, идейно очень близкие к работам Л. Фогеля с сотрудниками, были разносторонне развиты и описаны в работах И.Л. Букатовой [2, 3]. В более поздних работах Л. Фогеля [20, 21] эволюционное программирование используется для решения систем линейных алгебраических уравнений.

Логические (конечные) автоматы – это модели, описывающие средствами формальной логики возможные переходы исследуемой системы из некоторого начального состояния в заключительное. Удобной формой представления конечных автоматов являются ориентированные графы (рис. 3.13), где вершина q_0 – начальное состояние; q_f – заключительное состояние; q_1, q_2 – промежуточные состояния; $\{0,1\}$ – символы входного словаря.

Конечные автоматы используются в задачах распознавания, управления и многих других приложениях. Знаменитая машина Тьюринга является разновидностью конечного автомата.

Эволюционная программа реализует моделирование процессов естественной эволюции моделей-автоматов, причём в каждый момент времени сохраняется тот «организм», который наилучшим образом может справиться с данной задачей. «Родительский» организм оцени-

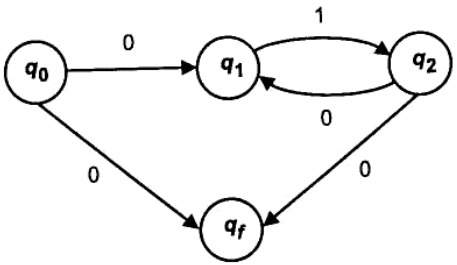


Рис. 3.13. Ориентированный граф, соответствующий конечному автомату

вается в зависимости от способности принимать требуемое решение на основе имеющихся данных. Этот организм подвергается мутации и производит на свет «потомка», которому ставится та же задача и который оценивается таким же образом. Автомат, который демонстрирует наилучшую способность выполнять требуемые функции, сохраняется и поставляет «потомков» в следующее поколение. Таким образом производятся всё лучшие и лучшие модели (программы) для решения поставленной задачи. Процесс завершается, когда получена достаточно хорошая программа или исчерпаны ресурсы времени. Всякий раз, когда поступает новая информация, происходит эволюционный поиск логической структуры, обеспечивающей получение наиболее приемлемого решения.

В эволюционном программировании объектами эволюции являются конечные автоматы, способные реагировать на стимулы, поступающие из внешней среды. Каждый автомат на основе текущей информации предсказывает состояние, соответствующее определённому значению функции ценности. Решение ищется постепенным отбором автоматов-родителей, к которым применяется мутация на следующем шаге эволюции.

В эволюционном программировании используются следующие способы реализации оператора мутации:

- изменение заключительного состояния;
- изменение условия перехода из одного состояния в другое;
- добавление нового состояния;
- удаление состояния;
- изменение начального состояния.

Обобщённый алгоритм эволюционного программирования включает следующие шаги.

1. Формулируется постановка задачи. Формируются входной словарь, множество входных и выходных состояний, набор возможных состояний, условия переходов из состояния в состояние, функция ценности для характеристики генерируемых моделей.

2. Случайным образом генерируется начальная популяция конечных автоматов-родителей.

3. Выполняется тестирование автоматов-родителей путём решения поставленной задачи (на вход модели подаётся заданный образец) и оценка полученных результатов на основе выбранной функции ценности.

4. Отсев неперспективных моделей.

5. На основе случайного применения оператора мутации к автоматам-родителям производятся потомки-члены новой популяции.

6. Тестирование моделей-потомков путём решения поставленной задачи и оценка полученных результатов.

7. Отбор наиболее перспективных потомков.

8. Проверка условий окончания процесса эволюции, в качестве которых могут быть: достижение оптимального значения функции ценности и/или достижение предельных значений, ограничивающих длительность процесса. Если условия завершения эволюции удовлетворены, то переход на шаг 9, в противном случае – возврат на шаг 5, где объекты последней сгенерированной популяции выступают в качестве родителей.

9. Конец алгоритма.

Дальнейшая эволюция автоматов возможна на основе предъявления автоматам более сложных задач.

Эволюционные стратегии. Эволюционные стратегии были предложены в 1970-х гг. [31, 32] в качестве стохастического метода нахождения глобального минимума функций многих переменных $F(X)$, суть которого состоит в следующем. Из случайных векторов решения задачи многокритериальной оптимизации $x = \{x_{ij}\}$, $j = 1, .., P$, P – размерность пространства параметров оптимизации, формируется начальная популяция объектов эволюции, над которыми выполняются следующие действия.

1. Из решений x формируются новые объекты-потомки x_i' путём сложения каждой компоненты $x_{ij} = x_{ij} + \xi_{ij}$ со случайной переменной ξ_{ij}, имеющей нормальный закон распределения с нулевым математическим ожиданием.

2. Вычисляются значения целевой функции $F(x_i)$ и $F(x_i')$ и осуществляется выбор наилучшего (минимального) решения, которое отбирается в новую популяцию.

3. Процесс продолжается до тех пор, пока не будет достигнуто приемлемое решение.

Каждый объект в популяции характеризуется двумя векторами – вектором решения и случайным вектором, модифицирующим это решение. Случайный вектор характеризуется вектором дисперсии, который хранится в процессе поиска, и может быть дополнен корректирующим вектором, ускоряющим сходимость алгоритма. Значение ξ_{ij} моделирует величину шага изменения параметров, выбираемую случайным образом. В общем случае ξ_{ij} может принимать любые значения, однако в схеме моделирования эволюционных механизмов вели-

чина ξ_{ij} отражает интенсивность мутаций «родителя» и поэтому не слишком велика. Совокупность полученных точек составляет очередное поколение решений, которые оцениваются по значениям минимизируемой функции $F(X)$. В результате отбора одни особи гибнут, а другие живут и размножаются. Эту простую схему легко усовершенствовать, вводя по аналогии с естественными закономерностями зависимость числа порождаемых потомков от значений функций ценности «родителей». Соответствующие эволюционные стратегии поиска известны и широко используются на практике. Популяции можно формировать следующими способами:

1) μ родителей порождают λ потомков, все решения борются за выживание, и лучшие μ объектов отбираются в следующую популяцию;

2) время жизни объекта ограничено одной генерацией, т.е. μ родителей, произведя λ потомков, погибают. За место в следующей популяции соревнуются только λ потомков, причём в данном способе должно выполняться условие $\lambda > \mu$ (рекомендуемое соотношение $\lambda / \mu > 7$). Такой подход применим к задачам с изменяющимся оптимумом и с зашумлёнными данными.

В эволюционных стратегиях используется оператор рекомбинации (в эволюционном программировании, в отличие от эволюционных стратегий, рекомбинация не применяется), который аналогичен скрещиванию в генетических алгоритмах. При этом компоненты вектора «потомка» создаются из компонент векторов решений двух «родителей». Это можно сделать разными способами, например:

– компоненты вектора потомка выбираются случайным образом из векторов родителей;

– компоненты вектора потомка получаются как средние арифметические значения компонент обоих родителей, а затем к полученному потомку применяется оператор мутации.

В эволюционных стратегиях иногда применяется глобальная рекомбинация, при которой компоненты вектора каждого потомка случайным образом выбираются из векторов всей популяции родителей.

Следует отметить, что моделирование естественных процессов развития, в том числе и эволюции, было и остаётся одним из самых перспективных научных направлений. Кроме описанных методов эволюционных вычислений, на основе естественных аналогий придуманы нейронные сети, предложены методы эволюционного синтеза систем и методы эволюционного проектирования технических объектов. Осо-

бенностью подходов, базирующихся на эволюционных аналогиях, является контраст между достаточно простым математическим аппаратом (по сравнению с другими методами) и впечатляющими результатами в области решения слабоструктурированных и плохо обусловленных проблем.

Великий Гёте назвал природу «творцом всех творцов», поэтому разработчикам ИИС ещё предстоит очень многому у неё научиться.

3.3. Контрольные вопросы и задания

1. Перечислите основные направления эволюционного моделирования и приведите основные факторы, определяющие неизбежность эволюции.
2. Какие алгоритмы называют генетическими? Сформулируйте основные особенности генетических алгоритмов.
3. Охарактеризуйте простой генетический алгоритм. Приведите пример.
4. Опишите операторы репродукции и кроссинговера в простом генетическом алгоритме. Приведите примеры.
5. Приведите примеры использования простого генетического алгоритма для вычисления функции $f(x) = x^4$ на интервале [0, 1, 2, 3, 4].
6. Составьте примеры, иллюстрирующие работу операторов репродукции, кроссинговера, мутации и инверсии.
7. Дайте характеристику понятию «схема» в простом генетическом алгоритме. Расскажите о назначении и способах использования схем. Приведите примеры.
8. Расскажите о фундаментальной теореме генетического алгоритма.
9. Приведите пример применения фундаментальной теоремы генетического алгоритма.
10. Сформулируйте прикладную экономическую или управленческую оптимизационную задачу и опишите её решение с применением генетического алгоритма.
11. Расскажите о классифицирующих системах Холланда. Приведите пример.
12. Перечислите основные этапы технологии генетического программирования.
13. В чём особенности эволюционного программирования? Приведите основные шаги обобщённого алгоритма эволюционного программирования.

14. Охарактеризуйте метод эволюционных стратегий. В чём его отличие от эволюционного программирования и от генетических алгоритмов?

15. Расскажите о применении эволюционных вычислении в ИИС. Каким образом применяют ГА для обучения нейронных сетей? Приведите небольшой содержательный пример, демонстрирующий применение ГА для формирования продукционных правил интеллектуальной системы.

16. Расскажите об устойчивости и эффективности генетического алгоритма.

17. Расскажите про генетические операторы и порядок их выполнения.

18. Сформулируйте критерии завершения работы генетического алгоритма.

19. Расскажите про обобщённый алгоритм эволюционного программировании, поясните каждый шаг.

20. Приведите пример конечного автомата, изобразите соответствующий ориентированный граф.

21. Расскажите о конечных автоматах.

22. Перечислите популярные программные средства, реализующие технологии оптимизации с применением генетических алгоритмов.

23. Дайте краткую характеристику средств, реализующих технологии оптимизации с применением генетических алгоритмов.

24. Сформулируйте общую задачу оптимизации сети.

25. Изобразите схему обработки правил в классифицирующей системе.

3.4. Список литературы

1. Батищев, Д.И. Генетические алгоритмы решения экстремальных задач : учеб. пособие / Д.И. Батищев. – Воронеж : Изд-во ВГТУ, 1995.

2. Букатова, И.Л. Эволюционное моделирование и его приложения / И.Л. Букатова. – М. : Наука, 1979.

3. Букатова, И.Л. Эвоинформатика. Теория и практика эволюционного моделирования / И.Л. Букатова и др. – М. : Наука, 1991.

4. Гудман, Э.Д. Эволюционные вычисления и генетические алгоритмы / Э.Д. Гудман, А.П. Коваленко // Обозрение прикладной и промышленной математики. – М. : ТВП, 1996. – Т. 3. – Вып. 5.

5. Корнеев, В.В. Базы данных. Интеллектуальная обработка информации / В.В. Корнеев и др. – М. : Нолидж, 2000.

6. Корячко, В.П. Теоретические основы САПР / В.П. Корячко, В.М. Курейчик, И.П. Норенков. – М. : Энергоатомиздат, 1987.

7. Курейчик, В.М. Генетические алгоритмы : монография / В.М. Курейчик. – Таганрог : Изд-во ТРТУ, 1998.

8. Курейчик, В.М. Генетические алгоритмы в проектировании СБИС : учеб. пособие / В.М. Курейчик. – Таганрог : Изд-во ТРТУ, 1997.

9. Курейчик, В.М. Методы генетического поиска : учеб. пособие / В.М. Курейчик. – Таганрог : Изд-во ТРТУ, 1998.

10. Растршин, Л.А. Статистические методы поиска / Л.А. Растршин. – М. : Наука, 1968.

11. Стецюра, Г.Г. Эволюционные методы в задачах управления, выбора и оптимизации / Г.Г. Стецюра // Приборы и системы управления. – 1998. – № 3.

12. Фогель, Л. Искусственный интеллект и эволюционное моделирование / Л. Фогель, А. Оуэне, М. Уолш. – М. : Мир, 1969.

13. Фролов, Ю.В. Интеллектуальные системы и управленческие решения / Ю.В. Фролов. – М. : Изд-во МГПУ, 2000.

14. Холланд, Дж. Генетические алгоритмы / Дж. Холланд // В мире науки. – 1992. – № 9, 10.

15. Цетлин, М.Л. Исследование по теории автоматов и моделирование биологических систем / М.Л. Цетлин. – М. : Наука, 1969.

16. Ackley, D.H. A Connection machine for genetic hillclimbing / D.H. Ackley. – Boston : Kluwer Academic Publishers, USA, 1987.

17. Booker, L.B. Classifier systems and genetic algorithms / L.B. Booker, D.E. Goldberg, J.H. Holland // Ibid. – 1989. – Vol. 40, No. 1–3.

18. Branke, J. A distributed genetic algorithm improving the generalization behavior of neural networks / J. Branke, U. Kohlmorgen, H. Sshmeck // LNAI. – 1995. – Vol. 912.

19. Dzerovski, S. Discovering dynamics with genetic programming / S. Dzerovski, I. Petrovski // LNAI. – Vol. 783.

20. Fogel, D. Evolutionary computatio / D. Fogel. – IEEE press, 1995.

21. Fogel, D.B. Comparing genetic operators with gaussian mutations in simulated evolutionary process using linear systems / D.B. Fogel, J.W. Atmar // Biol. Cybernetics. – 1990. – Vol. 63.

22. Foundation of Genetic Algorithms / Ed. by rawens gregory. – San Mateo : Morgan Kaufman Publishers, California, USA, 1991.

23. Goldberg, David E. Genetic algorithms in search, optimization and machine learning / David E. Goldberg. – Addison–Wesley Publishing Company, Inc. 1989.

24. Gruckles, B.P. Genetic algorithms / B.P. Gruckles, F.E. Party. – Los Alamos : IEEE Computer Society Press, LA, USA, 1992.

25. Handbook of Genetic Algorithms // Ed. by Lawrense Davis, Van Nostrand Reinholds. – New York, 1991.

26. Holland, J.H. Adaptation in natural and artificial systems. An introductory analysis with application to biology, control and artificial intelligence / J.H. Holland. – University of Michigan, 1975.

27. Koza, J. Genetic evolution and coevolution of computer programs / J. Koza // Artificial life 2. Proceedings of the Workshop on Artificial life, 1990.

28. Koza, J. Genetic programming / J. Koza // MIT press. – 1992.

29. Koza, J. GP2: Automatic discovery of reusable programs / J. Koza // MIT press. – 1993.

30. Ono, N. Self-organization of communication in distributed earning classifier systems. Artificial neural nets and genetic algorithms / N. Ono, A. Rahmani // Proceedings of the International Conference. – 1993. – P. 361 – 367.

31. Schwefel, H.P. Numerical optimization of computer models / H.P. Schwefel. – New York : John Willey, 1981.

32. Schwefel, H.P. Evolution and optimum searching / H.P. Schwefel. – New York : John Willey, 1995.

4. ИНТЕЛЛЕКТУАЛЬНЫЕ МУЛЬТИАГЕНТНЫЕ СИСТЕМЫ

Интеллектуальные мультиагентные системы – одно из новых перспективных направлений искусственного интеллекта, которое сформировалось на основе результатов исследований в области распределённых компьютерных систем, сетевых технологий решения проблем и параллельных вычислений. В мультиагентных технологиях заложен принцип автономности отдельных частей программы (агентов), совместно функционирующих в распределённой системе, где одновременно протекает множество взаимосвязанных процессов. Под агентом подразумевают автономный искусственный объект (компьютерную программу), обладающий активным мотивированным поведением и способный к взаимодействию с другими объектами в динамических виртуальных средах. Каждый агент может принимать сообщения, интерпретировать их содержание и формировать новые сообщения, которые либо передаются на «доску объявлений», либо направляются другим агентам.

Агентно-ориентированный подход уже нашёл применение в таких областях, как распределённое решение сложных задач, реинжиниринг предприятий, электронный бизнес и т.п. Важной областью применения мультиагентных технологий является моделирование. В этой области Д.А. Поспелов [9] выделяет два класса задач. К первому классу он относит задачи распределённого управления и задачи планирования достижения целей, где усилия разных агентов направлены на решение общей проблемы, и необходимо обеспечение эффективного способа

кооперации их деятельности. В задачах второго класса агенты самостоятельно решают свои локальные задачи, используя общие, как правило, ограниченные ресурсы.

4.1. Основные понятия теории агентов

Понятие агент соответствует аппаратно или программно реализованной сущности, которая способна действовать в интересах достижения целей, поставленных перед ней владельцем и/или пользователем [3, 12, 23].

В мультиагентных системах (МАС) множество автономных агентов действуют в интересах различных пользователей и взаимодействуют между собой в процессе решения определённых задач. Примерами таких задач являются: управление информационными потоками и сетями, управление воздушным движением, поиск информации в сети Интернет, электронная коммерция, обучение, электронные библиотеки, коллективное принятие многокритериальных управленческих решений и другие.

Идея мультиагентных систем появилась в конце 1950-х гг. в научной школе М.Л. Цетлина, которая занималась исследованиями коллективного поведения автоматов [14]. Агентами (маленькими животными) были названы искусственные существа, обладающие свойством реактивности, т.е. способные воспринимать и интерпретировать сигналы, поступающие из внешней среды, и формировать ответные сигналы. В роли маленьких животных выступали конечные автоматы, которые не имели априорных знаний о свойствах окружающей среды и о наличии в ней других существ. Единственным знанием, которым они обладали, была цель их деятельности и способность оценивать поступающие сигналы относительно достижения этой цели. Оказалось, что даже такие простые структуры, как конечные автоматы (см. главу 3), демонстрируют хорошие способности к адаптации в стационарных вероятностных средах. Одной из главных характеристик агентов–автоматов была рациональность, которая определялась как сумма положительных откликов среды, накопленных агентом за некоторый период его существования. В дальнейших исследованиях структура маленьких животных усложнялась. Сначала появились вероятностные автоматы с переменной структурой, адаптирующейся к характеристикам среды, затем появились агенты, способные изменять свои реакции на основании предыстории и анализа состояния окружения. Серьёзным шагом в развитии мультиагентных технологий стала реализация способности агентов к рассуждениям [7, 12]. Простейшие модели взаимодействия агентов предусматривали их общение через среду. При этом на каждом шаге функционирования агенты совершают вы-

бор возможных для них действий. Множество действий всех агентов обусловливает распределение откликов среды для всех участников, которые могут его использовать либо не использовать при формировании своих ответных реакций.

Новый шаг к современному пониманию агентов был сделан при переходе к коллективной работе в распределённых компьютерных системах. Этот шаг стал началом бурного развития мультиагентных технологий. К настоящему времени в данном направлении накоплен определённый опыт. Предложены разнообразные модели агентов и способы их реализации, решены практические задачи и созданы инструментальные средства для разработки мультиагентных систем, сформулированы различные принципы взаимодействия агентов и т.п. В этой главе мы остановимся на вопросах, связанных с построением и применением интеллектуальных МАС.

Одна из возможных классификаций агентов [3, 19] приведена в табл. 4.1, из которой следует, что для интеллектуальных агентов характерно целесообразное поведение, которое предполагает наличие у агента целей функционирования и способностей использовать знания об окружающей среде, партнёрах и о своих возможностях.

Интеллектуальным агентам присущи следующие основные свойства:

– автономность – способность функционировать без вмешательства со стороны своего владельца и осуществлять контроль собственных действий и внутреннего состояния. Автономность предполагает относительную независимость агента от окружающей среды, т.е. наличие «свободы воли», обусловливающей собственное поведение, которое должно быть обеспечено необходимыми ресурсами;

– активность – способность к организации и реализации действий;

– общительность – взаимодействие и коммуникация с другими агентами;

– реактивность – адекватное восприятие состояния среды и реакция на его изменение;

– целенаправленность, предполагающая наличие собственных источников мотивации;

– наличие базовых знаний о себе, о других агентах и об окружающей среде;

– убеждения – переменная часть базовых знаний, меняющихся во времени;

– желания – стремление к определённым состояниям;

– намерения – действия, которые планируются агентом для выполнения своих обязательств и/или желаний;

– обязательства – задачи, которые выполняет один агент по просьбе и/или поручению других агентов.

4.1. Классификация агентов

Признак	Тип агента			
	простой	смышлёный	ителлектуальный	действительно интеллектуальный
Автономность	+		+	+
Взаимодействие с другими агентами и/или пользователями	+	+	+	+
Реактивность	+	+	+	+
Способность использования абстракции		+	+	+
Адаптивное поведение		+	+	+
Обучение на основе взаимодействия с окружением			+	+
Толерантность к ошибкам и/или неверным входным сигналам			+	
Функционирование в режиме реального времени			+	
Взаимодействие на естественном языке			+	

Иногда к этому списку добавляются другие качества, в том числе:

– правдивость – неспособность к подмене истинной информации заведомо ложной;

– благожелательность – готовность к сотрудничеству с другими агентами в процессе решения собственных задач, что обычно предполагает отсутствие конфликтующих целей, поставленных перед агентами;

– альтруизм – приоритетность общих целей по сравнению с личными;

– мобильность – способность агента мигрировать по сети в поисках необходимой информации.

В работе [12] для классификации агентных программ используются два основных признака: 1) степень развития внутреннего представления о внешнем мире; 2) способ поведения.

По первому признаку выделяются интеллектуальные (когнитивные, рассуждающие) и реактивные агенты. Интеллектуальные агенты обладают хорошо развитой и пополняемой символьной моделью внешнего мира благодаря наличию у них БЗ, механизмов рассуждения и анализа действий. Реактивные агенты не имеют развитого представления о внешней среде. Они не используют рассуждений и могут не иметь собственных ресурсов. Их поведение определяется целью, в соответствии с которой формируются реакции на предъявляемые ситуации. В связи с этим реактивные агенты не имеют внутренних источников мотивации и не способны планировать свои действия (реактивность в чистом виде – это обратная связь без прогноза).

Интеллектуальная мультиагентная система представляет собой множество интеллектуальных агентов, распределённых в сети, которые мигрируют по ней в поисках релевантных данных, знаний, процедур и кооперируются для достижения поставленных перед ними целей.

В зависимости от концепции, принятой при разработке МАС, возможны различные варианты её архитектуры, среди которых выделяют три базовых типа:

1) архитектуры, основанные на методах работы со знаниями;

2) архитектуры, в которых используются поведенческие модели «стимул–реакция»;

3) гибридные архитектуры.

В архитектурах первого типа для представления и обработки знаний используются традиционные модели, методы и средства искусственного интеллекта, а принятие решений осуществляется на основе механизмов формальных рассуждений. В самых первых системах такого типа для представления и обработки знаний использовалась логика предикатов первого порядка. Развитие исследований в этой области привело к появлению специальных расширений логических исчислений, ориентированных на учёт таких свойств агентов, как убеждения, желания, намерения и обязательства [9, 12]. Основной недостаток архитектур первого типа – сложность или принципиальная невозможность построения достаточно полных баз знаний, которые являются необходимой частью создаваемых систем. В частности, интеллектуальный агент может иметь архитектуру типичной продукционной системы, которая способна воспринимать информацию из внешней среды и осуществлять те или иные действия в результате обработки этой информации. Главные отличия агентной программы от обычной продукционной ЭС связаны с наличием механизма формирования целей и

модуля коммуникации, который обеспечивает взаимодействие с другими агентами. Агент с такой архитектурой способен к рассуждениям, но не способен к обучению. Адаптивное поведение агента позволяет реализовать архитектура на основе классифицирующих систем Дж. Холланда. Важнейшими отличиями классифицирующих систем от продукционных являются: 1) возможность формирования новых правил с применением генетического алгоритма; 2) наличие механизма поощрений.

В архитектурах второго типа, которые называют реактивными, не используются традиционные для ИИ символьные модели представления знаний [16]. Модели поведения агентов представлены либо наборами правил, которые позволяют выбрать действие, соответствующее ситуации, либо конечными автоматами, либо другими средствами, обеспечивающими формирование адекватных реакций агента на возникающие в системе стимулы. Системы этого типа, как правило, имеют высокую степень специализации и строгие ограничения на сложность решаемых задач.

Наиболее перспективными считаются гибридные интеллектуальные мультиагентные системы, которые позволяют использовать возможности интеллектуальных и реактивных архитектур. Примером может служить архитектура с иерархической базой знаний, которая содержит структурированную БЗ, рабочую память, модуль управления коммуникацией и человеко-машинный интерфейс. Агент с подобной архитектурой обладает способностью к рассуждениям и к реактивному поведению. Его БЗ содержит три уровня: 1) знания предметной области; 2) знания о взаимодействии, которые позволяют принимать решения в условиях неопределённости; 3) управляющие знания. Интеллектуальное поведение агента обеспечивается способностью принимать решения, а реактивное – системой контроля за содержимым рабочей памяти, которая функционирует по принципу глобальной доски объявлений. Агент взаимодействует с пользователем, используя человеко-машинный интерфейс. В общем случае гибридные архитектуры являются многоуровневыми и отличаются друг от друга структурой и содержанием уровней, которые могут соответствовать различным уровням управления, абстракции либо отдельным функциональным свойствам агента.

Одно из новых направлений – применение нейронных сетей для реализации МАС. Коннекционистские архитектуры (на основе ИНС) позволяют создавать самообучающихся агентов, знания которых формируются в процессе решения практических задач. Хорошие перспективы для реализации самообучающихся агентов имеют сети с обратными связями и нечёткие ИНС [12].

4.2. Коллективное поведение агентов

Главная черта МАС, отличающая их от других интеллектуальных систем, – взаимодействие между агентами. Взаимодействие означает установление двусторонних и многосторонних динамических отношений между субъектами. Оно является не только следствием деятельности агентов, но и необходимым условием формирования виртуальных сообществ. Взаимодействие – не просто связь между сосуществующими агентами, но и предпосылка для взаимных превращений самих агентов и отношений между ними.

Главными характеристиками любого взаимодействия являются направленность, избирательность, интенсивность и динамичность. В контексте МАС эти понятия можно интерпретировать следующим образом:

– направленность – положительная или отрицательная; кооперация или конкуренция; сотрудничество или конфронтация; координация или субординация и т.п.;

– избирательность – взаимодействие происходит между агентами, которые каким-либо образом соответствуют друг другу и поставленной задаче. При этом агенты могут быть связаны в одном отношении и независимы – в другом;

– интенсивность – взаимодействие между агентами не сводится к наличию или отсутствию, а характеризуется определённой силой;

– динамичность – наличие, сила и направленность взаимодействий могут изменяться с течением времени.

Общая проблема анализа взаимодействия между агентами включает следующие задачи [12]:

1) идентификацию ситуации взаимодействия агентов;
2) выделение основных ролей и их распределение между агентами;
3) определение числа и типов взаимодействующих агентов;
4) построение формальной модели взаимодействия;
5) определение набора возможных стратегий поведения агентов;
6) формирование множества коммуникативных действий.

К базовым видам взаимодействия между агентами относятся:

– кооперация (сотрудничество);
– конкуренция (конфронтация, конфликт);
– компромисс (учёт интересов других агентов);
– конформизм (отказ от своих интересов в пользу других);
– уклонение от взаимодействия.

Интеллектуальные агенты сотрудничают с другими агентами «сознательно», преследуя при этом определённые цели. Кооперацию в

сообществе реактивных агентов можно назвать непреднамеренной, поскольку она базируется на естественных реакциях отдельных агентов, направленных на выживание вида. Показатели выживания отражают способность особи или группы сохранять свою целостность при воздействиях факторов, которые могут её разрушить. Кооперация между агентами может возникать на принудительных началах (директивная кооперация) или на основе добровольных отношений (ситуативная кооперация). Эти два вида сотрудничества часто представлены так называемой контрактной формой кооперации, когда взаимодействие агентов регламентируется набором формальных или неформальных соглашений между ними.

Взаимодействие агентов обусловлено целым рядом причин, важнейшими среди которых являются следующие.

Совместимость целей (общая цель). Эта причина обычно порождает взаимодействие по типу кооперации или сотрудничества. При этом следует выяснить, не ведёт ли взаимодействие к снижению жизнеспособности отдельных агентов. Несовместимость целей или убеждений обычно порождает конфликты, позитивная роль которых заключается в стимулировании процессов развития. Известная модель хищник–жертва представляет собой пример одновременного взаимодействия по двум типам кооперация–конфронтация.

Отношение к ресурсам. Ресурсами будем называть любые средства, используемые для достижения агентами своих целей. Задачи распределения долей рынка, затрат и прибылей совместных предприятий можно рассматривать как примеры взаимодействия, обусловленного общими ресурсами. Ограниченность ресурсов, которые используются многими агентами, обычно порождает конфликты. Одним их самых простых и эффективных способов разрешения подобных конфликтов является право сильного – сильный агент отбирает ресурсы у слабых. Более тонкие способы разрешения конфликтов обеспечивают переговоры [13], направленные на достижение компромиссов, в которых учитываются интересы всех агентов.

Необходимость привлечения недостающего опыта. Каждый агент обладает ограниченным набором знаний, необходимых ему для реализации собственных и общих целей. В связи с этим ему приходится взаимодействовать с другими агентами. При этом возможны различные ситуации: а) агент способен выполнить задачу самостоятельно; б) агент может обойтись без посторонней помощи, но кооперация позволит решить задачу более эффективным способом; в) агент не способен решить задачу в одиночку. В зависимости от ситуации агенты выбирают тип взаимодействия и могут проявлять разную степень заинтересованности в сотрудничестве.

Взаимные обязательства. Обязательства являются одним из инструментов, позволяющих упорядочить хаотические взаимодействия агентов. Они позволяют предвидеть поведение других агентов, прогнозировать будущее и планировать собственные действия. Можно выделить следующие группы обязательств: а) обязательства перед другими агентами; б) обязательства агента перед группой; в) обязательства группы перед агентом; г) обязательства агента перед самим собой. Формальное представление целей, обязательств, желаний и намерений, а также всех остальных характеристик составляет основу ментальной модели интеллектуального агента, которая обеспечивает его мотивированное поведение в автономном режиме.

Перечисленные причины в различных сочетаниях могут приводить к разным формам взаимодействия между агентами, например:

– простое сотрудничество, которое предполагает интеграцию опыта отдельных агентов (распределение задач, обмен знаниями и т.п.) без специальных мер по координации их действий;

– координируемое сотрудничество, когда агенты вынуждены согласовывать свои действия (иногда привлекая специального агента-координатора) для того, чтобы эффективно использовать ресурсы и собственный опыт;

– непродуктивное сотрудничество, когда агенты совместно используют ресурсы или решают общую проблему, не обмениваясь опытом и мешая друг другу (как лебедь, рак и щука в басне И.А. Крылова).

Рассматривая проблему моделирования взаимодействия агентов друг с другом и с окружающей средой, Д.А. Поспелов [2] выделил следующие основные признаки естественных систем, которые необходимо учитывать при моделировании виртуальных сред.

1. Конечность времени существования любого агента. Длительность жизни агента зависит от различных обстоятельств, в частности от поставленной перед ним задачи, от величины доступных ресурсов и т.п.

2. Использование механизма биологического отбора в моделях искусственной жизни. Естественный отбор эффективных агентов может осуществляться в адаптивных системах с использованием различных эволюционных механизмов (обучаемых нейронных сетей, генетических алгоритмов, автоматов с перестраиваемой структурой и т.д.).

3. Учёт уровня организации сообщества агентов. Если модель описывает взаимодействие сложных организмов, имеющих социальную организацию, то помимо реактивности, активности и когнитивности (способность к рассуждениям) агенты приобретают ещё одно свойство – социальность. В таких моделях возникает необходимость учёта социального статуса и социальных отношений. Распределение

труда в обществе служит основой для выделения классов агентов, выполняющих специализированные функции, в том числе функции управления искусственной средой. Задача распределения функций приводит к необходимости реализации механизма социального отбора, который принципиально отличается от биологического принципа.

Вопросы организации сообщества искусственных организмов по образу и подобию человеческого общества связывают теорию МАС с системным анализом, теорией организаций, теорией административного управления и т.п. Серьёзной и пока не решённой проблемой является морально-этическая основа организации мультиагентных систем, связанная с формированием понятий об основных ценностях и нормах, принятых в обществе. Ориентация на модели нормативного поведения агентов вызывает дискуссии, так как наряду с нормативным в реальном обществе имеет место и ненормативное поведение [9].

Коллективное поведение агентов в МАС предполагает кооперацию агентов при коллективном решении задач. В процессе работы мультиагентной системы агент может обращаться за помощью к другим агентам, если не в состоянии решить поставленную перед ним задачу самостоятельно. При этом агенты могут строить планы совместных действий, не только полагаясь на свои возможности, но анализируя планы и намерения других членов коллектива. Моделирование коллективного поведения необходимо также в случаях, когда агенты для решения своих задач используют общий ограниченный ресурс. Каждый агент вынужден учитывать наличие других агентов, а выбор стратегии действий одного агента обычно зависит от поведения остальных.

Проблемы коллективного поведения рассматриваются в теории систем, в теории управления и в теории игр. Основной идеей системного анализа является применение декомпозиции исходной задачи на более простые, из решения которых может быть найдено решение задачи в целом. В мультиагентных системах идея декомпозиции воплотилась в принцип распределённого решения подзадач с их координацией для получения стратегии коллективного поведения.

В процессе моделирования коллективной работы агентов возникает множество проблем [4]:

- распознавание необходимости кооперации;
- выбор подходящих партнёров;
- возможность учёта интересов партнёров;
- организация переговоров о совместных действиях;
- формирование планов совместных действий;
- синхронизация совместных действий;

– декомпозиция задач и разделение обязанностей;
– выявление конфликтующих целей;
– конкуренция за совместные ресурсы;
– формирование правил поведения в коллективе.
– обучение поведению в коллективе и т.д.

Особенностью коллективного поведения агентов является то, что их взаимодействие в процессе решения частных задач (или одной общей) порождает новое качество решения этих задач. При этом в моделях координации поведения агентов используются следующие основные идеи [4].

1. Отказ от поиска наилучшего решения в пользу «хорошего», что приводит к переходу от процедуры строгой оптимизации к поиску приемлемого компромисса, реализующего тот или иной принцип координации.

2. Использование самоорганизации в качестве устойчивого механизма формирования коллективного поведения.

3. Применение рандомизации (случайно-вероятностного способа выбора решений) в механизмах координации для разрешения конфликтов.

4. Реализация рефлексивного управления [6], сущность которого заключается в том, чтобы заставить субъекта осознанно подчиняться влиянию извне, т.е. сформировать у него такие желания и намерения (интенции), которые совпадают с требованиями окружения.

Наиболее известными моделями координации поведения агентов являются: теоретико-игровые модели, модели коллективного поведения автоматов, модели планирования коллективного поведения, модели на основе BDI-архитектур (Belief–Desire–Intention), модели координации поведения на основе конкуренции.

Теоретико-игровые модели. Предметом теории игр являются задачи выбора решений в условиях неопределённости и конфликта. Наличие конфликта предполагает существование как минимум двух участников, которых называют игроками. Множество решений, возможных для выбора каждым игроком, называется стратегией. Равновесными точками игры (оптимальными решениями) называют такие состояния, когда ни одному из игроков невыгодно менять свою позицию. Понятие равновесия оказалось весьма полезным в теории МАС, поскольку механизм поиска равновесных ситуаций может использоваться как средство самоорганизации коллективного поведения агентов. Следствием подобной интерпретации является подход, в котором необходимые атрибуты коллективного поведения агентов обеспечиваются путём конструирования правил игры. Кроме того, на основе разви-

тия теории игр в области МАС предпринимаются попытки построения эффективных, устойчивых, полностью распределённых протоколов переговоров, направленных на координацию коллективного поведения агентов.

В работе [24] множество возможных ситуаций выбора поведения пары агентов классифицируется следующим образом.

1. Симметричная кооперация, когда существует непустое множество стратегий (переговорное множество), при использовании которых оба агента достигают своих целей и получают больший эффект, чем в ситуациях, когда они действуют поодиночке.

2. Симметричный компромисс, когда достижение цели в одиночку более выгодно для каждого агента, однако невозможно в присутствии другого агента.

3. Несимметричная кооперация или несимметричный компромисс – один из агентов может самостоятельно достичь своей цели в присутствии другого агента, а другой – только за счёт кооперации с первым.

4. Конфликт – переговорное множество пусто, т.е. не существует стратегий, обеспечивающих достижение целей обоих агентов.

В этой же работе показано, что теоретико-игровые модели позволяют для всех перечисленных случаев сконструировать наборы правил переговоров, следуя которым агенты придут к некоторому соглашению, отвечающему состоянию равновесия. Это достигается за счёт использования множества дополнительных предположений и специальных приёмов. Например, кроме стоимости цели в рассмотрение вводится понятие ценности цели, а в качестве одной из возможных стратегий может выступать стратегия манипулирования информацией о ценности целей (т.е. агенты могут сообщать друг другу заведомо ложные значения). При этом «нечестные» агенты могут либо увеличить свой доход, либо освободиться от части своей работы.

Модели коллективного поведения автоматов. Они основаны на идеях рандомизации, самоорганизации и полной распределённости [1, 14]. Модели этого типа подходят для построения протоколов переговоров в задачах, которые характеризуются большим количеством очень простых взаимодействий с неизвестными характеристиками.

Модели планирования коллективного поведения. Планирование может быть централизованным, частично централизованным или распределённым (децентрализованным). В последнем случае агенты сами принимают решения о выборе своих действий в процессе координации частных планов, в связи с чем возникают вопросы о рациональной децентрализации, о возможности изменения целей при возникновении конфликтов, а также проблемы вычислительной сложности.

Модели на основе BDI-архитектур [12, 17]. В моделях этого класса применяются аксиоматические методы теории игр и логической парадигмы искусственного интеллекта. Для описания агентов используются логические средства, в том числе темпоральные и модальные логики. Акцент делается на описании интенсиональных понятий, таких, как убеждения (belief), желания (desire) и намерения (intention). Задача координации поведения агентов решается путём согласования результатов логического вывода в базах знаний отдельных агентов, полученных для текущего состояния внешней среды, в которой действуют агенты. Логический вывод осуществляется непосредственно в процессе функционирования агентов, что приводит к высокой сложности моделей, вычислительным трудностям и к проблемам, связанным с аксиоматическим описанием нетривиальных ситуаций, например, когда перед агентом возникает выбор между решением собственной задачи и выполнением обязательств по отношению к партнёрам.

Модели на основе конкуренции. В моделях данного класса используется понятие аукцион в качестве механизма координации поведения агентов. Использование механизма аукциона основано на предположении о возможности явной передачи «полезности» от одного агента к другому или к агенту-аукционеру, причём эта полезность обычно имеет смысл денег [4].

Аукционы принято разделять на открытые и закрытые. В первом случае предлагаемые цены объявляются всем участникам. В закрытом аукционе о предлагаемых ценах знает только аукционер. Открытые аукционы различаются по способу проведения. В так называемых английских аукционах обычно задаётся стартовая цена, которая может увеличиваться участниками в ходе торгов. Побеждает тот, кто даст максимальную цену. Голландский аукцион начинается с верхней цены, которая постепенно снижается. Победителем считается тот, кто дал наибольшую текущую цену. Закрытые аукционы разделяют на аукционы первой и второй цены. В аукционах первой цены побеждает тот, кто предложил самую высокую цену, известную только аукционеру. В аукционах второй цены победитель определяется таким же способом, но платит за товар не свою цену, а вторую по величине. Теоретически доказано, что все разновидности аукционов эквивалентны для аукционера, однако практика показывает иное. Например, если участники аукциона не склонны к риску, то аукционер стимулирует повышение цены продажи при проведении голландского аукциона первой цены. Существуют варианты «групповых» аукционов, когда один или несколько участников представляют интересы группы, и в случае выигрыша проводится аукцион внутри группы. При этом на внутреннем

аукционе товар продаётся по более высокой цене по сравнению с ценой внешнего аукциона. Полученная разница делится между участниками группы.

Сам по себе механизм аукциона не затрагивает способов принятия решений участниками. Решения могут приниматься на основе некоторой модели рассуждений, которая может использовать различные типы знаний, доступных агентам, и разнообразные способы их обработки.

Аукцион всегда должен заканчиваться. Для этого в стратегии его проведения должны быть заложены средства для разрешения возможных конфликтов (например, при наличии нескольких победителей). Одним из самых простых способов разрешения конфликтов является рандомизация, когда применяется случайный механизм выбора.

4.3. Примеры мультиагентных систем

Рассмотрим практические примеры организации взаимодействия в мультиагентных системах с использованием различных механизмов координации поведения.

Координация поведения на основе модели аукциона. Электронный магазин. Рассмотрим типичную задачу электронной коммерции [4], в которой участвуют агенты-продавцы и агенты-покупатели (рис. 4.1). Торговля осуществляется в электронном магазине, который представляет собой программу, размещённую на сервере. Её основным назначением является организация взаимодействия агентов, интересы которых совпадают. Агенты действуют по поручению своих персональных пользователей. При этом агенты-продавцы стремятся продать свой товар по максимально возможной цене, а агенты-покупатели стремятся купить нужный товар по минимальной цене. Оба вида аген-

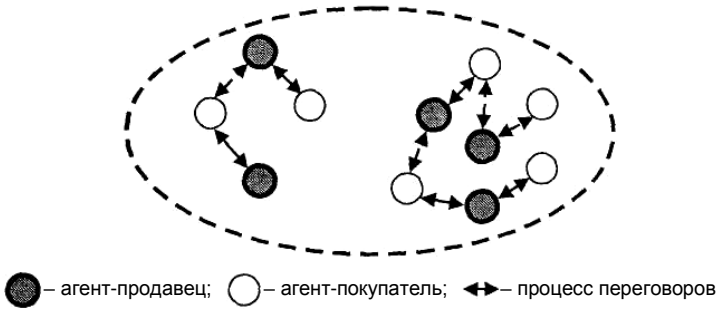


Рис. 4.1. Схема электронного магазина

тов действуют автономно и не имеют целей кооперации. Электронный магазин регистрирует появление и исчезновение агентов и организует контакты между ними, делая их «видимыми» друг для друга.

Поведение агента-продавца характеризуется следующими параметрами:

– желаемая дата, до наступления которой необходимо продать товар;
– желаемая цена, по которой пользователь хочет продать товар;
– самая низкая допустимая цена, ниже которой товар не продаётся;
– функция снижения цены во времени (линейная, квадратичная и др.);
– описание продаваемого товара.

Агент-покупатель имеет «симметричные» параметры:

– крайний срок покупки товара;
– желаемая цена покупки;
– самая высокая приемлемая цена;
– функция роста цены во времени;
– описание покупаемого товара.

Торги ведутся по схеме закрытого аукциона первой цены. Поведение агентов описывается простой моделью, в которой не используются знания и рассуждения. Агент-продавец, получив от электронного магазина информацию о потенциальных покупателях своего товара, последовательно опрашивает их всех с целью принять решение о возможности совершения сделки. Сделка заключается с первым агентом-покупателем, который готов дать за товар запрашиваемую цену. Продавец не может вторично вступить в контакт с любым покупателем до тех пор, пока не опросит всех потенциальных покупателей. При каждом контакте агент-продавец ведёт переговоры, предлагая начальную цену либо снижая её. Агент-покупатель действует аналогичным образом, отыскивая продавцов нужного товара и предлагая им свою цену покупки, которую он может увеличить в процессе переговоров. Любая сделка завершается только в случае её одобрения пользователем агента.

Данная схема переговоров представляет собой простейший случай взаимодействия автономных агентов, действующих реактивно. Тем не менее, итоговое поведение системы вполне адекватно реальности.

Виртуальное предприятие. Создание виртуальных предприятий является одним из современных направлений бизнеса, которое в значительной мере стимулируется быстрым ростом информационных ресурсов и услуг, предоставляемых в сети Интернет. Кроме того, появлению виртуальных предприятий способствует сокращение времени

жизненного цикла создаваемых изделий и повышение уровня их сложности, так как при этом возникает необходимость оперативного объединения производственных, технологических и интеллектуальных ресурсов. Ещё одна немаловажная причина – ужесточение конкуренции на товарных рынках, стимулирующее объединение предприятий в целях выживания.

Виртуальное предприятие можно определить как кооперацию юридически независимых предприятий, организаций и индивидуумов, которые производят продукцию или услуги в общем бизнес-процессе. Во внешнем мире виртуальное предприятие выступает как единая организация, в которой используются методы управления и администрирования, основанные на применении информационных и телекоммуникационных технологий. Целью создания виртуального предприятия является объединение производственных, технологических, интеллектуальных и инвестиционных ресурсов для продвижения на рынок новых товаров и услуг.

Поскольку каждое реальное предприятие в рамках виртуального выполняет только часть работ из общей технологической цепочки, то при его создании решаются две главные задачи. Первая – это декомпозиция общего бизнес-процесса на компоненты (подпроцессы). Вторая задача заключается в выборе рационального состава реальных предприятий-партнёров, которые будут осуществлять технологический процесс. Первая задача решается с применением методов системного анализа, а для решения второй могут применяться средства мультиагентных технологий.

Задача оптимального распределения множества работ (подпроцессов) среди множества работников (реальных предприятий) в исследовании операций формулируется как задача о назначениях [5]. Её решение начинается с формирования множеств подпроцессов и потенциальных предприятий-участников. Затем строятся возможные отображения из множества участников на множество подпроцессов и делается выбор наиболее приемлемого отображения, которое соответствует конкретным назначениям предприятий на бизнес-процессы. Для этого можно использовать механизм аукциона. На рисунке 4.2 приведена схема аукциона по созданию виртуального предприятия, в котором выделены бизнес-процессы A, B, C, D, E и участвуют четыре предприятия: P_1, P_2, P_3, P_4, претендующие на их реализацию. Каждое из предприятий представлено интеллектуальным агентом, при этом одно из них (P_1) выступает в роли инициатора (аукционера).

Перед началом аукциона аукционер (менеджер) формирует базу данных и базу знаний об участниках аукциона. Затем он выставляет на продажу отдельные бизнес-процессы, информация о которых пред-

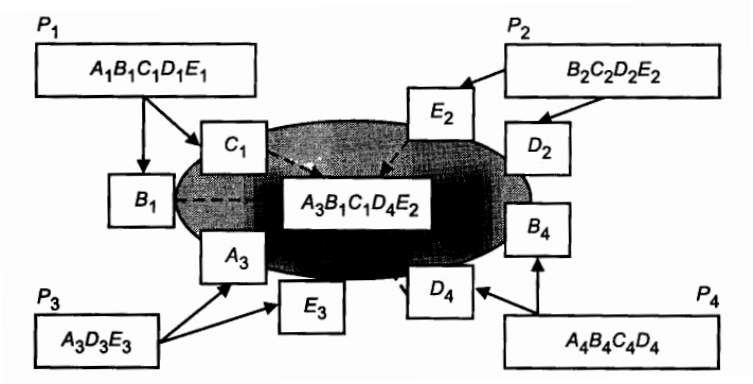


Рис. 4.2. Схема создания виртуального предприятия

ставлена стартовой ценой и требованиями по заданному набору показателей. Каждый претендент выдвигает свои предложения по параметрам, которые он в состоянии обеспечить, и свою цену. Собрав и обработав эти предложения, аукционер с помощью некоторой модели рассуждения упорядочивает потенциальных претендентов с учётом собственной информации о них. После этого он принимает решение о выборе назначений или отвергает их и выдвигает новые предложения.

Мультиагентная система для поддержки принятия решений на предприятии. Интеллектуальные мультиагентные системы принятия решений предназначены для оценки качества организационно-технических и экономических решений в процессе деятельности предприятия.

В настоящее время происходит переход от концепции стабильного бизнеса к мобильному, в котором главную роль играют понятия конкурентоспособность и гибкость. Для работы в новых быстроизменяющихся условиях предприятиям необходимо постоянно трансформировать свои производственные структуры и структуры бизнес-процессов. При этом становится неизбежным привлечение сторонних специалистов из области технологий, маркетинга, реинжиниринга и т.д. Оценка предлагаемых решений является сложным и постоянным видом деятельности, требующим участия высококвалифицированных экспертов из разных областей знаний, которые, как правило, территориально удалены друг от друга. Этим обусловлена актуальность распределённой компьютерной поддержки процессов принятия решений на предприятиях, которая может быть реализована с применением мультиагентных систем.

Рассмотрим пример мультиагентной системы принятия решений для многокритериальной оценки инновационной деятельности предприятия [11].

Общая схема принятия решений включает следующие этапы:

1) спецификация требований;
2) генерация решений;
3) оценка альтернатив;
4) выбор эффективного решения.

Оценку решений проводит рабочая группа, которая состоит из руководителя, аналитика и экспертов. Функции между участниками рабочей группы распределяются следующим образом. Руководитель формирует набор показателей (критериев), которые будут использоваться для оценки проектов (решений); подбирает состав группы экспертов; составляет персональный календарь, в соответствии с которым эксперты выполняют свои задания. Каждый эксперт работает по индивидуальному сценарию, предложенному руководителем. Аналитик, функции которого может выполнять руководитель, высказывает своё мнение о результатах проведённой экспертами работы.

Для поддержки группового процесса принятия решений используется программная реализация метода анализа иерархий [10], где реализованы следующие основные процедуры:

– формирование и согласование иерархической структуры показателей;
– оценка и согласование качественных показателей проекта;
– оценка и согласование важности показателей;
– ранжирование альтернативных решений и согласование результатов.

В решении перечисленных задач участвует множество экспертов, поэтому на каждом этапе предусмотрены процедуры согласования их мнений.

Ядром мультиагентной системы «Multi Expert» (рис. 4.3) является менеджер знаний, использующий три внешних компонента:

– информационную модель проблемной области в виде упорядоченного набора показателей качества решений;
– средства технической и программной поддержки;
– множество типов пользователей (руководитель, координатор, эксперт, аналитик).

Для координации работы коллектива экспертов используется двухуровневый механизм согласования. Каждый из экспертов представлен агентом, в задачу которого входит оценка предлагаемых руководителем альтернатив по заданному набору показателей качества.

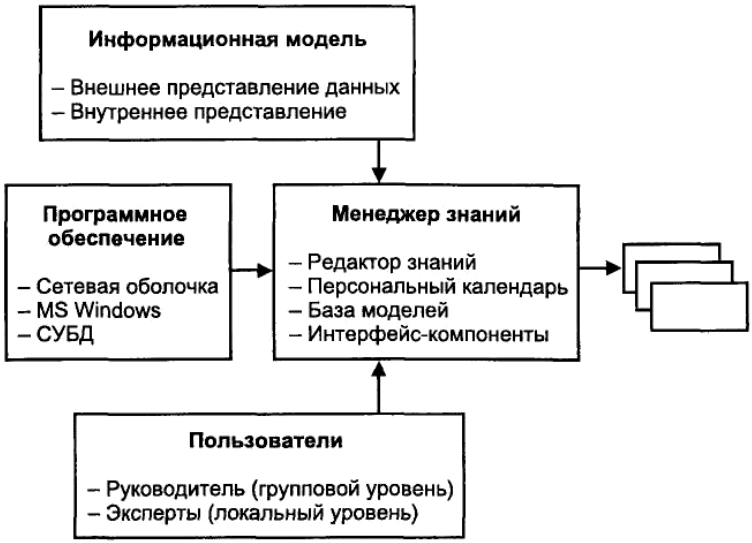


Рис. 4.3. Обобщённая структура мультиагентной системы принятия решений «Multi Expert»

С помощью редактора знаний руководитель формирует задания экспертам и проводит анализ полученной от них информации. Задача координации поведения агентов возложена на агента-координатора. Результатом работы системы являются согласованные экспертные оценки, на основании которых проводится многокритериальное ранжирование альтернатив.

Рассмотрим основные функции агентов в системе «Multi Expert».

Агент-руководитель:

– предоставляет набор процедур для облегчения работы руководителя в распределённой системе;

– вычисляет конечный результат на основании данных, полученных от других агентов;

– отслеживает согласованность решения, вырабатываемого группой;

– предоставляет средства визуализации результатов работы;

– подготавливает сообщения агенту-координатору;

– выполняет почтовые функции в распределённой среде.

Агент-координатор:

– обеспечивает выполнение пошагового алгоритма принятия решения;

– поддерживает целостность баз данных системы на групповом уровне и вносит в них необходимые изменения;

– подготавливает диалоговые формы для информационного обмена через Интернет.

Агент-эксперт:

– поддерживает выполнение текущего шага задания;

– готовит сообщения агенту-координатору;

– поддерживает целостность локальных баз данных;

– выполняет почтовые функции в распределённой среде.

Работа агентов осуществляется следующим образом. Руководитель формирует задания, оперируя справочниками, содержащими знания об экспертах, показателях качества и решениях, требующих рассмотрения. Далее задание в виде входного сообщения M_{inp} поступает агенту-координатору, определяющему состав изменений, которые необходимо сделать в базах данных на локальном уровне. Координатор с помощью предоставленного ему набора функций готовит информацию для всех агентов-экспертов рабочей группы. Агенты-эксперты выполняют задания, предназначенные для своих пользователей, анализируя поступившие от координатора сообщения M_{ij} (j – номер эксперта), и отсылают ему ответные сообщения M_{0j}.

Агент-координатор собирает сообщения о готовности выполненных заданий от всех членов группы. При выполнении всего пакета заданий его состояние изменяется, и посылается сообщение агенту руководителя M_{0ut}.

Руководитель может выполнять проверку согласованности экспертных суждений либо на основе вычислений, либо с помощью логического анализа предоставленной ему информации. Решение руководителя о степени согласованности суждений посылается агенту-координатору, который продвигает задание на следующий шаг или возвращает экспертов на предыдущий этап в целях достижения лучшей согласованности.

4.4. Технологии проектирования мультиагентных систем

Программно реализованные агенты, в том числе и интеллектуальные, относятся к классу программного обеспечения, которое способно действовать самостоятельно от лица пользователя. Созданию программных агентов предшествовал опыт разработки так называемых открытых систем [8], результатом внедрения которых в практику явилось создание архитектуры «клиент–сервер». В настоящее время наибольшее распространение получили две модели такого взаимодействия: «толстый клиент–тонкий сервер» и «тонкий клиент–толстый сер-

вер». В первой модели серверная часть реализует доступ к ресурсам, а приложения находятся на компьютерах клиентов. Во второй модели клиентское приложение обеспечивает только реализацию интерфейса, а сервер объединяет все остальные части программного обеспечения. При создании МАС используются обе модели. При этом может применяться либо статический подход, при котором осуществляется передача только данных, либо динамический подход, обеспечивающий также передачу программного кода.

Динамический подход опирается на парадигму мобильных агентов, которые в отличие от статических могут перемещаться по сети. Они могут покидать клиентский компьютер и перемещаться на удалённый сервер для выполнения своих действий, после чего могут возвращаться обратно. Использование мобильных агентов имеет положительные и отрицательные последствия, поэтому их применение оправдано в тех случаях, когда они обеспечивают следующие возможности [3]:

- уменьшение времени и стоимости передачи данных;
- расширение ограниченных локальных ресурсов;
- облегчение координации;
- выполнение асинхронных вычислений.

При использовании мобильных агентов возникает ряд серьёзных проблем, в том числе: легальность способов перемещения агентов по сети; верификация агентов (например, защита от вирусов); соблюдение прав частной собственности; сохранение конфиденциальности информации; перенаселение сети агентами; совместимость кода агента и программно-аппаратных средств сетевой машины.

Для реализации мультиагентных систем, основанных как на статических, так и на динамических распределённых приложениях, наиболее перспективными на сегодняшний день являются следующие технологии [3, 18]: DCOM (Microsoft Distributed Component Object Model), JawaRMI (Jawa Remote Method Invocation) и CORBA (Common Object Request Broker Architecture).

Главной особенностью объектно-ориентированной технологии DCOM является возможность интеграции приложений, реализованных в разных системах программирования.

В приложениях JawaRMI на сервере создаются объекты и методы их обработки, доступные для вызова удалёнными приложениями, которые размещаются на компьютерах-клиентах.

Технология CORBA – одно из наиболее гибких средств реализации распределённых приложений. Её преимуществом по сравнению с JawaRMI является наличие специального языка описания интерфейсов IDL, унифицирующего средства коммуникации между приложениями и способы взаимодействия с другими приложениями.

Подробную информацию о программных продуктах, предназначенных для разработки мультиагентных систем, можно найти в Интернете по адресу: http:// www.Reticular.com.

Для поддержки процессов проектирования агентов и мультиагентных систем разработаны специальные инструментальные средства. Чтобы получить представление об их возможностях и о технологии создания МАС, рассмотрим в качестве примера систему Agent Builder.

Инструментарий Agent Builder (Reticular Systems, Inc.) предназначен для разработки мультиагентных систем на основе Java-программ, что позволяет исполнять их на любом компьютере, где установлена виртуальная Java-машина (Java Virtua lMachine). Общая схема процесса проектирования и реализации приложений на основе Agent Builder ToolKit представлена на рис. 4.4.

Модель «жизненного цикла» создаваемых агентов включает следующие этапы:

– обработку новых сообщений;
– определение правил поведения;
– выполнение действий;
– обновление ментальной модели в соответствии с заданными правилами;
– планирование действий.

Ментальная модель включает описание намерений, желаний, обязательств и возможностей, а также правил поведения агентов. На основе этой модели осуществляется выбор тех или иных действий интеллектуального агента.

Правила поведения в системе Agent Builder реализуются на специальном объектно-ориентированном языке RADL (Reticular Agent Definition Language) в виде конструкции When–If–Then. Составные части этого правила выполняют следующие функции:

When<...> содержит новые сообщения, полученные от других агентов;

If<...> сравнивает текущую ментальную модель с условиями применимости правила;

Then<...> определяет действия, соответствующие текущим событиям, состоянию ментальной модели и внешнего окружения.

Правила поведения агентов записываются в формате:

Name < *Имя правила* >
When < *Message Conditions* >
If < *Mental Conditions* >
Then < *Private Actions*; *Mental Changes*; *Message Actions* >

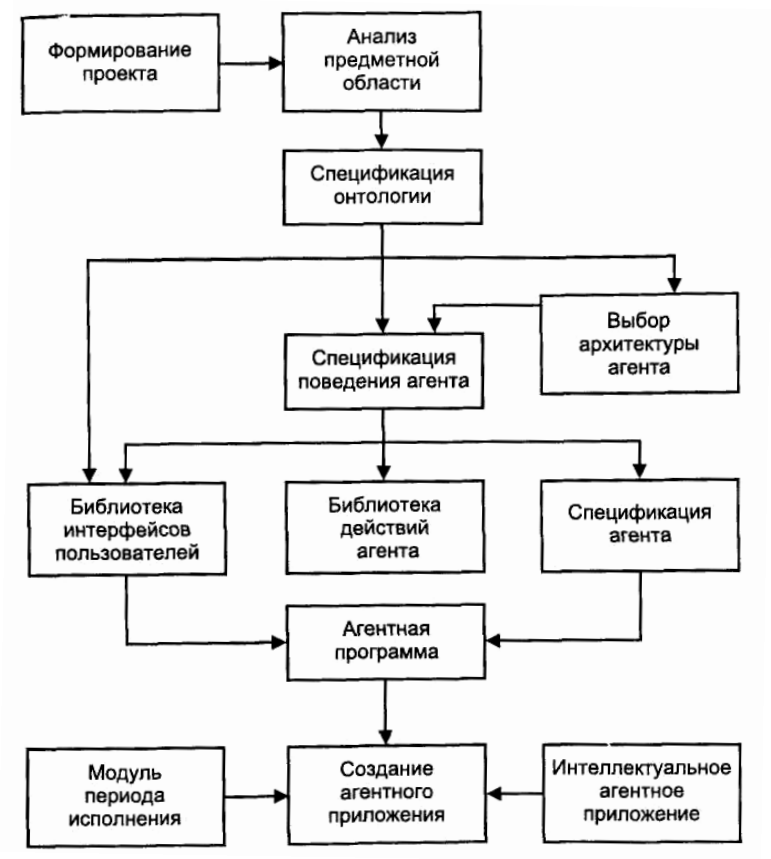


Рис. 4.4. Схема процесса проектирования приложений в системе Agent Builder ToolKit

В языке RADL используются структуры данных, подобные фреймам, а правила представляют собой продукции специального вида. При проектировании приложений необходимо составить спецификации моделей поведения агентов, которые будут применяться совместно с классами и методами из библиотеки действий агентов и библиотеки интерфейсов.

Являясь достаточно мощным средством для представления и обработки знаний, Agent Builder не предусматривает применения средств явного управления логическим выводом, которые могли бы существенно расширить возможности используемого языка.

Мультиагентные системы для поиска информации. В связи с быстрым развитием интернет-технологий возникла необходимость применения средств искусственного интеллекта для поиска и обработки интернет-ресурсов. Применение интеллектуальных МАС для решения задач сбора, поиска и анализа информации в глобальных сетях даёт следующие существенные преимущества перед традиционными средствами обработки информации [3]:

– обеспечение доступа пользователя к сетевым протоколам в сети Интернет;

– параллельное решение нескольких задач;

– выполнение поиска информации после отключения пользователя от сети;

– увеличение скорости и точности поиска, а также уменьшение загрузки сети за счёт поиска информации непосредственно на сервере;

– создание собственных баз информационных ресурсов, постоянно обновляемых и расширяемых;

4.2. Анализ систем интеллектуального поиска и обработки информации

Характеристика	Autonomy	WebCompass
Категория пользователей, на которую ориентирована система	Конечные пользователи	«Продвинутые» пользователи
Подход к описанию предметной области	Технология нейронных сетей и специальные методы распознавания образов и обработки сигналов	Иерархиипонятий, связанных отношениями типа IS-A, PART-OF, HAS-PART, IS-A KIND-OF и т.д.
Средства спецификации запросов	Естественный язык	«Прямое» использование сформированного пользователем описания предметной области
Методы поиска релевантной информации	Нечёткая логика	Поиск по списку ключевых слов одновременно на 35 машинах поиска
Режим обучения поисковых агентов	Есть	Нет

– реализация возможности сотрудничества между агентами, которая позволяет использовать накопленный опыт;

– возможность автоматически корректировать и уточнять запросы, используя контекст и применяя модели пользователей.

В таблице 4.2 приведены отличительные особенности известных в России коммерческих мультиагентных систем Autonomy [15] и Web Compass [22], предназначенных для интеллектуального поиска и обработки информации в сети Интернет.

Недостатком современных систем интеллектуального поиска и обработки информации является их слабая способность к обучению. Поэтому основные усилия по совершенствованию интеллектуальных систем информационного поиска в сети Интернет направлены на развитие моделей представления знаний, механизмов вывода новых знаний, моделей рассуждения и способов обучения агентов [20].

Одним из успешных исследовательских проектов, выполненных в этом направлении, стал проект системы MARRI [21], которая была разработана для поиска Web-страниц, релевантных запросам в определённой предметной области. Для решения поставленной задачи система использует знания, представленные в виде онтологии, под которой в данном случае понимается упорядоченное множество понятий предметной области. Архитектура системы MARRI показана на рис. 4.4.

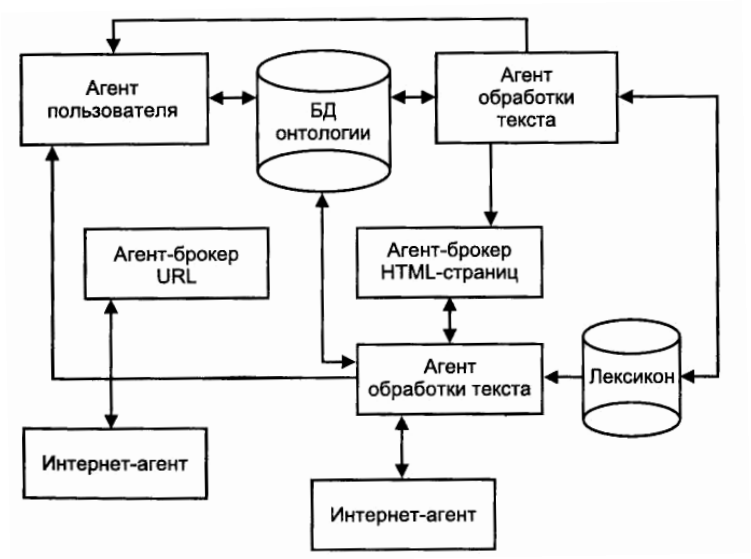


Рис. 4.4. Архитектура системы MARRI

Система MARRI включает следующие типы агентов:

– интерфейсный агент (агент пользователя) обеспечивает интеллектуальное взаимодействие с пользователем. Он поддерживает процесс формулирования запросов и представляет результаты поиска в виде списка URL или Web-страниц;

– агенты-брокеры двух типов: 1) брокер типа URL предназначен для формирования списков интернет-адресов, поставляемых браузером; 2) брокер типа HTML выполняет функции запоминания полученных Web-сграниц и их распределения между агентами обработки текста;

– агент сети (интернет-агент) обеспечивает считывание и анализ заданной страницы URL или Web-страницы (URL – автономная Java – программа с собственным сетевым адресом). Он должен уметь выполнять обработку исключительных ситуаций (например, когда страница недоступна), а также производить анализ текста на считанной странице;

– агент обработки текста сначала преобразует HTML-текст к представлению, с которым работают морфологический и синтаксический анализаторы, а затем проводит семантический анализ Web-страниц для проверки их релевантности запросу на основе соответствующей онтологии. Результат обработки текста представляется в виде синтаксического дерева, которое должно соответствовать какому-нибудь фрагменту используемой онтологии.

Каждый из перечисленных типов агентов наделён специальными знаниями, которые используются для повышения эффективности поиска информации. Агенты способны взаимодействовать друг с другом; обмениваться информацией, контактировать с Web-браузерами, анализаторами естественного языка и онтологическими базами данных.

Отличительной чертой системы MARRI является представление агентов автономными Java-программами, каждая из которых имеет собственный сетевой адрес (URL). Это обеспечивает мобильность агентов, но противоречит политике безопасности, не допускающей запуск подобных программ, если они не сертифицированы на данном сервере.

4.5. Перспективы мультиагентных технологий

В работе [9] сформулированы проблемы, решение которых может существенно продвинуть вперёд технологию мультиагентных систем и исследования в области «искусственной жизни».

1. Применение принципов гомеостатического управления. Предположение о том, что наилучшее взаимодействие агентов в МАС достигается при бесконфликтной кооперации, не всегда справедливо. Это утверждение можно аргументировать примерами биологических систем, в которых эффективной оказывается кооперация противоборст-

вующих сторон (хищник–жертва, взаимодействие симпатической и парасимпатической нервных систем). Противодействующие структуры позволяют поддерживать системы с многокритериальным управлением в границах области «неулучшаемых» решений (область Парето). Поэтому одним из актуальных направлений развития теории МАС является применение принципов гомеостатического управления (гомеостаз – равновесие), основы которого были заложены в работе отечественной научной школы Ю.М. Горского.

2. Создание адекватных механизмов активизации знаний, требующихся при решении конкретных проблем. Опыт создания интеллектуальных систем показывает, что увеличение количества знаний приводит к эффекту «государственной публичной библиотеки». Обладая огромным запасом знаний, библиотека не имеет каких-либо умений и навыков. Поэтому одной из существенных проблем интеллектуальных агентов является повышение их активности, которая связана не с накоплением знаний, а с умением активизировать нужные знания в процессе решения задач. Разработка процедур активизации знаний будет способствовать созданию действительно интеллектуальных агентов.

3. Перспективным направлением является использование идей рефлексивного управления в МАС. Эксперименты с агентами, наделёнными способностью к рефлексивным рассуждениям, показали эффективность данного подхода.

4. Создание конструктивных моделей этических систем и моделей поступков в среде обитания агентов.

5. Исследование влияния внешних факторов на поведение коллектива искусственных агентов и личностных характеристик агентов (психологические типы, оптимизм в оценках достижимости целей, азартность, упорство, конфликтность и т.п.).

4.6. Контрольные вопросы и задания

1. Расскажите о сущности мультиагентных технологий. Что подразумевается под агентом и как он может быть реализован?
2. Какими свойствами обладают «интеллектуальные агенты»?
3. Дайте характеристику архитектурам мультиагентных систем.
4. Сформулируйте основные проблемы, возникающие при моделировании коллективного поведения интеллектуальных агентов.
5. Охарактеризуйте основные модели координации поведения агентов в мультиагентных системах: теоретико-игровые, модели коллективного поведения автоматов, модели планирования коллективного поведения, модели на основе BDI-архитектур, модели координации поведения на основе конкуренции.

6. Сформулируйте постановки задач координации поведения агентов на основе модели аукциона.

7. Проведите сравнительный анализ свойств мобильных и статических агентов.

8. Опишите технологию построения мультиагентных систем.

9. Перечислите основные преимущества интеллектуальных поисковых мультиагентных систем перед традиционными средствами поиска информации.

10. Для каких задач актуально применение мультиагентных технологий? Приведите примеры.

11. Сформулируйте содержательный пример задачи кооперации и покажите возможный способ её решения средствами мультиагентных технологий.

12. Приведите пример задачи координации коллективного поведения, для решения которой актуально применение мультиагентных технологий. Сформулируйте принцип координации и правила нормативного поведения агентов.

13. Спроектируйте виртуальный магазин. Опишите виды агентов, их функции и способы возможной реализации.

14. Спроектируйте структуру мультиагентной системы для реализации конкретного виртуального предприятия. Опишите виды агентов, их функции и способы возможной реализации. Охарактеризуйте механизм координации поведения агентов.

15. Спроектируйте интеллектуальную мультиагентную систему для решения прикладной задачи в области экономики и управления. Реализуйте спроектированную систему на ЭВМ.

16. Расскажите о процессе проектирования приложений в системах разработки мультиагентных систем.

17. Приведите примеры инструментальных средств проектирования мультиагентных систем.

18. Расскажите о возможностях агентного автоматизированного извлечения и обработки информации.

19. Расскажите о свойствах моделей координации поведения агентов.

20. Расскажите о мультиагентных системах для поиска информации.

21. Расскажите о проблемах развития агентных систем.

22. Расскажите о концепциях, применяемых при разработке МАС.

23. Назовите основные признаки естественных систем, которые необходимо учитывать при моделировании виртуальных сред.

24. Расскажите об основных идеях, используемых в моделях координации поведения агентов.

25. Расскажите о классификации множества возможных ситуаций выбора поведения пары агентов.

4.7. Список литературы

1. Варшавский, В.И. Оркестр играет без дирижера / В.И. Варшавский, Д.А. Поспелов. – М. : Наука, 1984.

2. Гаазе-Раппопорт, М.Г. От амебы до робота. Модели поведения / М.Г. Гаазе-Раппопорт, Д.А. Поспелов. – М. : Наука, 1987.

3. Гаврилова, Т.А. Базы знаний интеллектуальных систем / Т.А. Гаврилова, В.Ф. Хорошевский. – СПб. : Питер, 2000.

4. Городецкий, В.И. Многоагентные системы: основные свойства и модели координации поведения / В.И. Городецкий // Информационные технологии и вычислительные системы. – 1998. – № 1.

5. Кудрявцев, Е.М. Исследование операций в задачах, алгоритмах и программах / Е.М. Кудрявцев. – М. : Радио и связь, 1984.

6. Лефевр, В. Конфликтующие структуры / В. Лефевр. – М. : Советское радио, 1973.

7. Моделирование обучения и поведения. – М. : Наука, 1974.

8. Орлик, С. Многоуровневые модели в архитектуре клиент–сервер / С. Орлик. – 1997. – http://www.citforam.ru/database/osbd/glava_94.html.

9. Поспелов, Д.А. Многоагентные системы – настоящее и будущее / Д.А. Поспелов // Информационные технологии и вычислительные системы. – 1998. – № 1.

10. Саати, Т.Л. Принятие решений. Метод анализа иерархий / Т.Л. Саати. – М. : Радио и связь, 1989.

11. Смирнов, А.В. Многоагентные системы поддержки принятия решений для предприятий малого и среднего бизнеса / А.В. Смирнов, М.П. Пашкин, И.О. Рахманова // Информационные технологии и вычислительные системы. – 1988. – № 1.

12. Тарасов, В.Б. От многоагентных систем к интеллектуальным организациям: философия, психология, информатика / В.Б. Тарасов. – М. : Эдиториал УРСС, 2002.

13. Трахтенгерц, Э.А. Взаимодействие агентов в многоагентных системах / Э.А. Трахтенгерц // Автоматика и телемеханика. – 1998. – № 9.

14. Цетлин, М.Л. Исследования по теории автоматов и моделированию биологических систем / М.Л. Цетлин. – М. : Наука, 1969.

15. Autonomy. Autonomy Technology Whitepaper. – 1998. – http:// www. autonomy.com/tech/wp .html

16. Brooks, R.A. Intelligence without Representation / R.A. Brooks // Artificial Intelligence. – 1991. – No. 47.

17. Georgeff, M.P. Social plans: Preliminary report / M.P. Georgeff, S. Rao // Proceedings of 3rd European Workshop on Modeling Autonomous Agents and Multi–Agent Worlds, 1992, North Holland.

18. Gopalan, R.S. A Detailed Comparison of CORBA, DCOM and Java/RNI (with specific code examples) / R.S. Gopalan. – http:// www. execpc. com/~gopalan/ index.html

19. Nwana, H.S. Software Agents: An Overview / H.S. Nwana // Knowledge Engineering Review. – 1996. – Vol. 11, No. 3, Cambridge University Press.

20. Pagina, H. Intelligent Software Agent on the Internet / H. Pagina. – 1996. – http:// www.hermans.org/agents/index.html

21. Villemin, F.Y. Ontologies – based relevant information retrieval / F.Y. Villemin. – 1999. – http:// www.cnam.fr/f–gv

22. WebCompass. WebCompass Page. – 1999. – http:// www.syman–tec.com/techsupp/webcompass/kbase_webcompass.html

23. Wooldridge, M. Inntellidgent Agents: Theory and Practice / M. Wooldridge, N. Jennings // Knowledge Engineering Review. – 1994. – № 10(2).

24. Zlotkin, G. Mechanisms for Automated Negotiation in State Oriented Domain / G. Zlotkin, J.S. Rosenschein // Journal of Artificial Inteeligence Research. – 1996. – № 4.

5. ОСНОВЫ РЕАЛИЗАЦИИ ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ НА ЯЗЫКЕ CLIPS

5.1. Основные теоретические сведения

CLIPS использует продукционную модель представления знаний, которая реализуется следующими основными компонентами языка описания правил:

1) база фактов (fact base);
2) база правил (rule base);
3) блок вывода (машина логического вывода).

На них возлагаются следующие функции: база фактов представляет исходное состояние проблемы, база правил содержит операторы, которые преобразуют состояние проблемы, приводя его к решению. Машина логического вывода CLIPS сопоставляет эти факты и правила и выясняет, какие из правил можно активизировать. Это выполняется циклически, причём цикл состоит из трёх шагов:

1) сопоставление фактов и правил;
2) выбор правила, подлежащего активизации;
3) выполнение действий, предписанных правилом.

Такой трёхшаговый циклический процесс иногда называют «циклом распознавание–действие».

Принципиальным отличием данной системы от аналогов является то, что она полностью реализована на языке С. Причём исходные тексты её программ опубликованы в Интернет.

В CLIPS используется оригинальный LISP-подобный язык программирования, ориентированный на разработку экспертных систем (ЭС). CLIPS поддерживает две парадигмы программирования: объектно-ориентированную и процедурную.

Основными элементами технологии программирования в CLIPS являются три основных элемента: простые типы данных; конструкции для пополнения базы знаний; функции для манипулирования данными [4 – 7].

Для представления информации в CLIPS предусмотрено восемь простых типов данных: float, integer, symbol, string, external-address, fact-address, instance-name, instance-address.

При записи числа используются цифры (0 – 9), десятичная точка (.), знак (– ,+), (e).

Любое число, состоящее только из цифр, перед которыми стоит знак, сохраняется как целое (В CLIPS – **integer**, в С – long integer). Все остальные числа сохраняются как **float** (в С – double float). **Symbol** – последовательность символов кода ASCII, причём как только в этой последовательности встречается символ-разделитель, symbol заканчивается. Символы-разделители: пробел, табуляция, двойные кавычки, (,), &, |, <, >, ~, ;.

String – последовательность символов, заключённых в двойные кавычки ("a and b"). Причём, если внутри строки встречаются двойные кавычки, то перед ними надо поместить символ (\) ("a and \"b").

Язык CLIPS является **чувствительным к регистру.** Например, такие символы рассматриваются в языке CLIPS как различные:

case–sensitive Case–Sensitive CASE–SENSITIVE

Для того чтобы иметь возможность наблюдать за всеми изменениями, происходящими в состоянии CLIPS, предусмотрена команда **Window–>All Above.** Данная команда открывает окна **Facts** (содержит факты из списка фактов) и **Agenda** (содержит все правила из списка активных правил).

Ввести команду в CLIPS можно непосредственно из диалогового окна, появившегося после запуска. Но в этом случае написанные правила после закрытия CLIPS будут потеряны. Поэтому текст программы необходимо сохранить в файле. В CLIPS имеется встроенный редактор.

Для загрузки содержимого файла в базу знаний CLIPS, нужно воспользоваться пунктом **Load** меню **File**.

Для того чтобы CLIPS активизировал начальный факт (initial-fact с идентификатором F-0) необходимо выбрать **Execution→Reset**. Данная команда удаляет существующие факты из списка фактов, включает в список фактов исходный факт (initial-fact), включает в список фактов все факты, описанные в конструкциях (deffacts).

Затем, по команде **Execution→Run** CLIPS начнёт выполнение всех правил программы. Для сохранения протокола работы программы, а также получения ответа в текстовом файле необходимо сразу после запуска CLIPS выполнить команду **File→Turn Dribble On** и в диалоговом окне ввести имя файла, в который будет сохраняться содержимое главного окна CLIPS. После окончания работы программы выполнить **Turn Dribble Off,** по этой команде файл для вывода будет закрыт.

Элементы математической логики. Логика высказываний. При решении логических задач с помощью экспертных систем в CLIPS предусмотрена возможность применения математического аппарата алгебры высказываний, позволяющего представлять факты и правила в виде логических выражений.

Под ***высказыванием*** *p* понимается всякое утвердительное предложение, относительно которого можно сделать заключение, истинно оно или нет. Содержанием высказывания не интересуются, интерес представляет лишь истинность или ложность высказывания. Высказывание считается истинным, если оно равно 1, ложным – если оно равно 0. Над высказываниями можно производить логические операции (для высказываний X и Y):

Отрицание ($\neg X$) – высказывание, которое истинно тогда и только тогда, когда X ложно. В разговорной речи высказыванию $\neg X$ соответствуют фразы: «не X», «неверно, что X».

1. *Конъюнкция* ($X \wedge Y$) – логическое умножение. Высказывание, которое истинно тогда и только тогда, когда истинны оба высказывания. В разговорной речи ей соответствует союз «и» ($X \wedge Y$ – «X и Y»).

2. *Дизъюнкция* ($X \vee Y$) – логическое сложение. Высказывание, которое истинно тогда и только тогда, когда ложны оба высказывания. В разговорной речи ей соответствует союз «или» ($X \vee Y$ – «X или Y»).

3. *Импликация* ($X \rightarrow Y$) – логическое следование. Высказывание, которое ложно тогда только тогда, когда X – истинно, а Y – ложно. В разговорной речи импликации соответствуют следующие высказывания: «X только тогда, когда Y», «из X следует Y», «если X то Y». При этом X – посылка, а Y – заключение.

5.1. Таблица истинности для логических операций

A	B	¬A	¬B	A∧B	A∨B	A→B	A↔B
И	И	Л	Л	И	И	И	И
И	Л	Л	И	Л	И	Л	Л
Л	И	И	Л	Л	И	И	Л
Л	Л	И	И	Л	Л	И	И

4. *Эквиваленция* (*X*↔*Y*) – высказывание, которое истинно тогда и только тогда, когда истинностные значения высказываний *X* и *Y* совпадают. В разговорной речи эквиваленция соответствует высказываниям вида: «*X* эквивалентно *Y*», «*X* тогда и только тогда, когда *Y*», «*X* необходимо и достаточно для *Y*».

Таблицей истинности логических операций называется таблица, в которой отражены результаты операций на всех возможных наборах значений высказываний (табл. 5.1).

При помощи рассмотренных операций возможно создавать комбинации из высказываний.

Для сложных высказываний, можно создавать комбинации, построенные из нескольких исходных высказываний посредством применения логических операций ¬, ∧, ∨, →, ↔. Их называют *формулами алгебры высказываний*.

При вычислении по формуле учитывается приоритет логической операции. Перечисленные выше логические операции расположены в порядке убывания приоритета. Изменить порядок логических высказываний можно с помощью расстановки скобок.

Исходные высказывания могут быть постоянными, то есть иметь определённые значения «истина» или «ложь». Если элементарное высказывание не имеет определённого значения, то это переменное высказывание. Например:

1) *A:=«Джек лает», B:=«Джек любит кости»*

C:=A∧B (Джек лает и любит кости) – это постоянное высказывание

2) *A(X):=«Собака (X) лает», B(X):=«Собака (X) любит кости»*

C(X):=A(X)∧B(X) («Собака с именем X лает и любит кости») – это переменное высказывание.

Пропозициональной формулой (ПФ) называется логическое выражение, содержащее переменные, соответствующие логическим высказываниям, 0, 1, константы и логические операции ¬, ∨, ∧, →, ↔,

называемые пропозициональными связками, скобки (,), используемые для определения приоритета операций. ПФ определяется индуктивно следующим образом:

1. Отдельно взятая переменная (высказывание) и константа (0, 1) – это ПФ.

2. Если A и B, составленные из допустимых символов, – ПФ, то и $\neg A$, $\neg B$, $A \vee B$, $A \wedge B$, $A \rightarrow B$, $A \leftrightarrow B$ – тоже ПФ.

Никаких других ПФ, кроме образованных по правилу 2, нет.

Пример:

$p \vee q \wedge r \rightarrow p$ – пропозициональная формула

$p \vee q \wedge r \rightarrow \square p$ – непропозициональная формула

Таблицей истинности для ПФ является перечень значений данной ПФ при всех возможных значениях входящих в неё переменных.

Пропозициональная формула называется ***тавтологией***, если на всех значениях входящих в неё переменных она равна 1. Обозначение: |=A пропозициональная формула A есть тавтология.

Приведём ряд тавтологий, могущих оказаться полезными при преобразовании высказываний.

Закон двойного отрицания: $\models \neg p \leftrightarrow p$ (1)

Закон исключенного третьего: $\models p \vee \neg p$ (2)

Идемпотентность операций дизъюнкции и конъюнкции: $\models p \wedge p \leftrightarrow p$ (3)

$\models p \vee p \leftrightarrow p$ (4)

p из конъюнкции: $\models p \wedge q \rightarrow p$ (5)

p из дизъюнкции: $\models p \rightarrow p \vee q$ (6)

Коммутативность операций дизъюнкции и конъюнкции:

$\models p \wedge q \leftrightarrow q \wedge p$ (7)

$\models p \vee q \leftrightarrow q \vee p$ (8)

Ассоциативность операций дизъюнкции и конъюнкции:

$\models (p \wedge q) \wedge r \leftrightarrow p \wedge (q \wedge r)$ (9)

$\models (p \vee q) \vee r \leftrightarrow p \vee (q \vee r)$ (10)

Разложение операций дизъюнкции и конъюнкции:

$\models p \vee (q \wedge r) \leftrightarrow (p \vee q) \wedge (p \vee r)$ (11)

$\models p \wedge (q \vee r) \leftrightarrow (p \wedge q) \vee (p \wedge r)$ (12)

Правила де Моргана:

$\models \neg (\wedge p) \leftrightarrow \neg p \vee \neg q$ (13)

$\models \neg (\vee p) \leftrightarrow \neg p \wedge \neg q$ (14)

Закон контрапозиции: $\models \rightarrow p \leftrightarrow \neg p \rightarrow \neg q$ (15)

Транзитивность импликации: $\models (p \rightarrow q) \wedge (q \rightarrow r) \leftrightarrow p \rightarrow r$ (16)

Закон косвенного доказательства: $\models (\neg p \rightarrow q) \wedge (\neg p \rightarrow \neg q) \rightarrow p$ (17)

Закон разбора случаев: $|= (p \vee q) \wedge (p \rightarrow r) \wedge (q \rightarrow r) \rightarrow r$ (18)

Транзитивность эквиваленции: $|= (p \leftrightarrow q) \wedge (q \leftrightarrow r) \rightarrow (p \leftrightarrow r)$ (19)

Закон противоположности: $|= (p \leftrightarrow q) \rightarrow (\neg p \leftrightarrow \neg q)$ (20)

Представление единицы:

$|= 1 \leftrightarrow p \vee \neg p$ (21)

$|= 1 \leftrightarrow p \rightarrow p$ (22)

Представление нуля:

$|= 0 \leftrightarrow p \vee \neg p$ (23)

$|= 0 \leftrightarrow \neg(p \rightarrow p)$ (24)

Представление импликации через дизъюнкцию и отрицание:

$|= (p \rightarrow q) \leftrightarrow (\neg p \vee p)$ (25)

Представление эквиваленции:

$|= (p \leftrightarrow q) \leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$ (26)

$|= (p \leftrightarrow q) \leftrightarrow (\neg p \wedge q) \wedge (p \vee \neg q)$ (27)

$|= (p \leftrightarrow q) \leftrightarrow (p \wedge q) \vee (\neg p \wedge \neg q)$ (28)

Представление конъюнкции: $|= p \wedge q \leftrightarrow \neg(p \rightarrow \neg q)$ (29)

Представление дизъюнкции: $|= p \vee q \leftrightarrow (\neg p \rightarrow q)$ (30)

При проектировании интеллектуальных информационных систем язык CLIPS предоставляет возможность применения эвристических алгоритмов поиска в пространстве состояний. При этом большинство поисковых задач можно сформулировать как задачи поиска впространстве состояний пути от исходного состояния заданной задачи до целевого состояния путём повторения возможных преобразований. При этом для организации поиска в пространстве состояний удобно использовать дерево поиска (или его более общую форму – граф).

Одним из подобных алгоритмов поиска является так называемый алгоритм A^*, где используются априорные оценки стоимости пути до целевого состояния, что обеспечивает высокую эффективность поиска [3].

Основная идея алгоритма состоит в использовании для каждого узла n на графе пространства состояний оценочной функции вида $f(n) = g(n) + h(n)$. Здесь $g(n)$ соответствует расстоянию на графе от узла n до начального состояния, а $h(n)$ – оценка расстояния от узла n до узла, представляющего конечное (целевое) состояние. Чем меньше значение оценочной функции $f(n)$, тем «лучше», т.е. узел n лежит на более коротком пути от исходного состояния к целевому. Идея алгоритма состоит в том, чтобы с помощью $f(n)$ отыскать кратчайший путь на графе от исходного состояния к целевому.

Алгоритм A* [3].

Введём следующие обозначения:

s – узел начального состояния;

g – узел конечного (целевого) состояния;

OPEN – список выбранных, но необработанных узлов;

CLOSED – список обработанных узлов.

Шаги:

1. OPEN = {s}.

2. Если OPEN:={}, то прекратить выполнение. Путь к целевому состоянию на графе не существует.

3. Удалить из списка OPEN узел n, для которого $f(n) \le f(m)$ для любого узла m, уже присутствующего в списке OPEN, и перенести его в список CLOSED.

4. Сформировать список очередных узлов, в который возможен переход из узла n, и удалить из него все узлы, образующие петли; с каждым из оставшихся связать указатель на узел n.

5. Если в сформированном списке очередных узлов присутствует g, то завершить выполнение. Сформировать результат – путь, порождённый прослеживанием указателей от узла g до узла s.

6. В противном случае для каждого очередного узла n', включённого в список выполнить следующую последовательность операций:

6.1. Вычислить $f(n')$.

6.2. Если n' не присутствует ни в списке OPEN, ни в списке CLOSED, добавить его в список, присоединить к нему оценку $f(n')$ и установить обратный указатель на узел n.

6.3. Если n' уже присутствует всписке OPEN или в списке CLOSED, сравнить новое значение $f(n') = new$ с прежним $f(n') = old$.

6.4. Если $old \le new$, прекратить обработку нового узла.

6.5. Если $old > new$, заменить новым узлом прежний всписке, причём, если прежний узел был в списке CLOSED, перенести его в список OPEN.

5.2. 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.

Пример:

(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 <class-name> [<optional-comment>]
(is-a <superclass-name>) <slot-definition>*)

В этом определении терм <superclass-name> 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 <slot-definition> определено следующим образом:

(slot <slot-name> <slot-attribute>*) | (multislot <slot-name> <slot-attribute>*)

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

```
(defclass PERSON "PERSON defclass"
    (is-a USER)
    (slot full-name)
    (slot age)
    (slot eye-color)
    (slot hair-color))
```

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, allowed-instance-names, default и default-dynamic.

Пример применения таких атрибутов:

```
(defclass PERSON "PERSON defclass"
    (is-a USER)
    (slot full-name
        (type STRING))
    (slot age
        (type INTEGER)
        (range 0 120))
    (slot eye-color
        (type SYMBOL)
        (allowed-values brown blue green)
        (default brown))
    (slot hair-color
        (type SYMBOL)
        (allowed-values black brown red blonde)
        (default brown))
    )
```

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 С 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 <name>
(slot-1)
(slot-2)
. . . . . .
(slot-N)),

где <name> – 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.

Пример:

(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]:

(defrule <имя правила>
< необязательный комментарий >
< необязательное объявление >
< предпосылка_1 >
< предпосылка_m>
=>
<действие_1 >
<действие_n>)

Пример:

(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:

(wash car)
(chop wood)

в случае, если правило будет активизировано. Приведённый в тексте правила комментарий "Things todo on Sunday" (Что сделать в воскре-

сенье) 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"
```

```
(today is ?day)
?chore <- (do ?job on ?day)
=>
(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 <-), который мы собираемся исключить из базы. Таким образом, если это правило будет сопоставлено с базой фактов, в которой содержатся

```
(today is Sunday)
(do carwash on Sunday)
```

то при активизации правила из базы будет удалён факт

```
(do carwash on Sunday)
```

Отметим, что факт

```
(do carwash on Sunday)
```

будет сопоставлен с любым из представленных ниже образцов

```
(do ? ? Sunday)
(do ? on ?)
(do ? on ?when)
```

Если за префиксом ? не следует имя переменной, он рассматривается как универсальный символ подстановки, которому может быть сопоставлен любой элемент.

При написании правил в части посылок иногда требуются некоторые логические операции, например, необходимо указать факты, что «сегодня суббота или воскресенье», «цветок не синий», «шар большой и зелёный». Это реализуется специальными логическими операторами: «**ИЛИ**», «**НЕ**», «**И**», которые обозначаются как |, ~, & соответственно. Таким образом, указанные выше факты запишутся следующим образом:

```
(today is Saturday|Sunday)
(flower is ~blue)
(ball is big&green)
```

Использование экземпляров и классов вместо фактов и конструкций deftemplate предоставляет несколько преимуществ. Первым из них является само наследование. Конструкция defclass может наследовать информацию от одного или нескольких различных классов. Это позво-

ляет создавать более структурированные, модульные определения данных. Вторым преимуществом является то, что за объектами можно закрепить относящуюся к ним процедурную информацию с помощью обработчиков сообщений. Третьим преимуществом является то, что сопоставление с шаблонами на основе объектов обеспечивает большую гибкость, чем сопоставление с шаблонами на основе фактов. В объектных шаблонах может использоваться наследование, сопоставление с шаблонами может осуществляться с учётом слотов, принадлежащих нескольким классам, существует возможность исключить повторную активизацию шаблона под действием изменений в незаданных слотах, а также может обеспечиваться поддержание истинности на основе значений слотов.

5.3. Типы функций манипулирования данными

Существует несколько типов функций: *пользовательские* и *системные*. Системные определены внутри среды CLIPS изначально, пользовательские – фрагменты кода, написанные пользователями на CLIPS или C.

Хотя CLIPS не ориентирован на численные вычисления, в нём предусмотрены стандартные математические и арифметические функции: +, –, *, /, ** (возведение в степень), **Abs**, **Sqrt**, **Mod**, **Min**, **Max**.

Пример:

(+ 2 5 8).

Конструкция **deffunction** позволяет пользователю определять новые функции непосредственно в среде CLIPS [4 – 8].

(deffunction <имя функции> (<аргумент> ... < аргумент >)
<выражение>
<выражение>)

Пример:

```
(deffunction hypotenuse (?a ?b)
(sqrt (+ (* ?a ?a) (* ?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 про-

белами. 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)
(reset)
(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> 5.0
```

Пример:

```
(deffunction between(?lb ?value ?ub)
        (or (> ?lb ?value) (> ?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 ?a присваивается значение 4:

```
(bind ?a 4)
```

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

5.4. Особенности решения задач планирования действий системы в заданной предметной области

Задачи планирования – определить последовательность действий модуля решения, например 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 – A и B. В комнате A находится робот, в комнате B – ящик. Задача – вытолкнуть ящик в комнату A.

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 A, второй, что ящик в комнате B, 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 <- (in (object robot) (location ~?Y))
=>
(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 <- (in (object ?X) (location ?Y))
?robot-position <- (in (object robot) (location ?Y))
=>
(modify ?robot-position (location ?Z))
(modify ?object-position (location ?Z))
)
```

В данном случае изменяются два факта, ссылки на которые задаются в переменных ?object-position и ?robot-position: значение слота location меняется на значение переменной ?Z, соответствующей значению, куда необходимо переместить предмет роботом.

Остановка выполнения программы в CLIPS осуществляется с помощью команды (halt). Условием остановки является наличие факта,

что предмет, указанный в цели (слот object) находится в комнате, указанной в слоте to:

```
(defrule stop
(goal (object ?X) (to ?Y))
(in (object ?X) (location ?Y))
=>
(halt))
```

Полный листинг программы представлен в [3].

5.5. 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 age)
(slot eye-color)
(slot hair-color)
(slot job-position)
(slot employer)
(slot salary))
(deftemplate student "Student deftemplate"
(slot full-name)
(slot age)
(slot eye-color)
(slot hair-color)
(slot university)
(slot major)
(slot GPA))
```

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)
[John]
CLIPS> (make-instance [Jack] of EMPLOYEE)
[Jack]
CLIPS> (make-instance [Jill] of STUDENT)
[Jill]
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)
(age nil)
(eye-color nil)
(hair-color nil)
(job-position nil)
(employer nil)
(salary 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)
CLIPS>

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 A
(is-aUSER)
(slot x (default 3))
(slot y)
(slot z (default 4)))

(defclassB
(is-a A)
(slot x)
(slot y (default 5))
(slot z (default 6)))
```

В таком случае создание экземпляров классов A и B приведёт к получению следующих результатов:

```
CLIPS> (make-instance [a] of A)
CLIPS> (make-instance [b] of B)
CLIPS> (send [a] print)
[a] of A (x 3) (y nil) (z 4)
CLIPS> (send [b] print)
[b] ofB (x nil) (y 5) (z 6)
CLIPS>
```

Обратите внимание на то, что слоту x экземпляра b по умолчанию присвоено значение nil вместо 3. Это связано с тем, что при отсутствии заданного по умолчанию значения для слота x класса B полностью перекрывается заданное по умолчанию значение 3, присваиваемое слоту x в классе A. 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 A и B будут объявлены следующим образом:

```
(defclassA
(is-a USER)
(slot x (default 3))
```

```
(slot y)
(slot z (default 4)))
(defclass B (is-a A)
(slot x (source composite))
(slot y (default 5))
(slot z (default 6)))
```

то после создания экземпляров классов A и B будут получены такие результаты:

```
CLIPS> (make-instance [a] of A)
CLIPS> (make-instance [b] of B)
CLIPS> (send [a] print)
[a] of A (x 3) (y nil) (z 4)
CLIPS> (send [b] print)
[b] ofB (x 3) (y 5) (z 6)
CLIPS>
```

Теперь, после того как слот x класса B объявлен с атрибутом source, которому присвоено значение composite, этот слот может наследовать заданный по умолчанию атрибут от класса A, и применяемое по умолчанию результирующее значение для слота x 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 A и B будет определены следующим образом:

```
(defclass A
(is-a USER)
(slot x (propagation no-inherit))
(slot y))
(defclass B
(is-a A)
(slot z))
```

то после создания экземпляров классов A и B будут получены такие результаты:

```
CLIPS> (make-instance [a] of A)
CLIPS> (make-instance [b] of B)
CLIPS> (send [a] print)
a of A (x nil) (y nil)
```

```
CLIPS> (send [b] print)
b of B (y nil) (z nil)
CLIPS>
```

Экземпляр b класса B наследует слот y из класса A, но не наследует слот x из класса A, 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
      (is-a USER)
      (role abstract))
(defclass MAMMAL
      (is-a ANIMAL)
      (role abstract))
(defclass CAT
      (is-a MAMMAL)
      (role concrete))
(defclass DOG
      (is-a MAMMAL)
      (role concrete))
```

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]
CLIPS>
```

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.

Пример 1:

(defclass A (is-a USER))

Класс A является прямым наследником класса USER. Список старшинства классов для A: A USER OBJECT.

Пример 2:

(defclass B (is-a USER))

Класс B является прямым наследником класса USER. Список старшинства классов для B: B USER OBJECT.

Пример 3:

(defclass C (is-a A B))

Класс C является прямым наследником классов A и B. Список старшинства классов для C: C A B USER OBJECT.

Пример 4:

(defclass D (is-a B A))

Класс D является прямым наследником классов A и B. Список старшинства классов для D: D B A USER OBJECT.

Пример 5:

(defclass E (is-a A C))

В соответствии с правилом 2, A должен быть старше C. В нашем случае, C – это потомок A и является более старшим в соответствии с правилом 1. Ошибка.

Пример 6:

defclass E (is-a C A))

Правильное определение класса из примера 5. Список старшинства для E: E C A B 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.

Пример:

```
(defclass A (is-a USER)
(slot fooA)
(slot barA))

(defclass B (is-a A)
(slot fooB)
(slot barB))
```

Список старшинства для A: A USER OBJECT. Экземпляр класса A будет иметь 2 слота: fooA и barA. Список старшинства для B: B A USER OBJECT. Экземпляр класса B 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 (make-instance a of A) – создаётся экземпляр с именем класса A. Другой вариант (создание массива экземпляра классов):

```
(definstances my_inst
(a of A)
(b of A)
(c of A)
)
```

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.

Пример:

```
CLIPS> (clear)
CLIPS>
(defclass A (is-a USER)
(roleconcrete)
(multislot foo (create-accessor read)
(default abc def ghi)))
CLIPS> (make-instance a of A)
[a]
CLIPS> (nth$ 2 (send [a] get-foo))
def
CLIPS>
```

Если при создании слота указывается модификатор для создания методов для записи или чтения по умолчанию ((create-accessor read-write)), то экземпляр класса будет реагировать на сообщения get-имя_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> (clear)
CLIPS> (setgen 1)
1
CLIPS>
(defclass A (is-a USER)
(role concrete)
(slot foo (default-dynamic (gensym))
(create-accessor read)))
CLIPS> (make-instance al of A)
[a1]
CLIPS> (make-instance a2 of A)
[a2]
CLIPS> (send [a1] get-foo)
gen 1
CLIPS> (send [a2] get-foo)
gen2
CLIPS>
```

Фасет Storage. Фасет определяет, будет ли значение слота храниться локально в экземпляре класса (local), либо это значение будет одно для всех экземпляров класса (shared).

Пример:

```
CLIPS> (clear)
CLIPS>
(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)
[a]
CLIPS> (send [a] print)
[a] of A
(foo 1)
(bar 2)
(woz ni1)
```

```
CLIPS> (send [a] put-foo 56)
CLIPS> (send [a] put-bar 104)
104
CLIPS> (make-instance b of A)
[b]
CLIPS> (send [b] print)
[b] of A
(foo 56)
(bar 2)
(woz nil)
CLIPS> (send [b] put-foo 34)
34
CLIPS> (send [b] put-woz 68)
68
CLIPS> (send [a] print)
[a] of A
(foo 34)
(bar 2)
(woz nil)
CLIPS> (send [b] print)
[b] of A
(foo 34)
(bar 2)
(woz 68)
CLIPS>
```

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)
CLIPS>
(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)))
CLIPS>
(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
FALSE
CLIPS> (make-instance a of A (foo 34) (woz 65))
[a]
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
FALSE
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
FALSE
CLIPS> (send [a] print)
[a] of A
(foo 34)
(bar abc)
(woz 65)
CLIPS>

5.6. Обработка сообщений

Изменение значений свойств объектов по правилам объектно-ориентированного программирования производится самими объектами, поэтому в языке CLIPS это реализовано посредством ***обработчиков сообщений*** [4 – 8].

Общий синтаксис команды создания обработчика сообщений:

(defmessage-handler <class-name><message-name>
[<handler-type>] [<comment>]
(<parameter>* [<wildcard-parameter>])
<action>*)

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]:

(send <object-expression> <message-name-expression> <expression>*)

Например, сообщение print отображает информацию о слотах экземпляра:

```
CLIPS> (send [John] print)
[John] of PERSON
(full-name "John Q. Public")
(age 24)
(eye-color blue)
(hair-color black)
CLIPS>
```

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, put-eye-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)
24
CLIPS>
```

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)
nil
CLIPS> (send [Jack] put-age 22)
22
CLIPS> (send [Jack] get-age)
22
CLIPS>(send [Jack] put-age)
nil
CLIPS> (send [Jack] get-age)
nil
CLIPS>
```

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 <- 24
24
CLIPS> (unwatch slots)
CLIPS> (send [Jack] put-age 22)
22
CLIPS>
```

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)
TRUE
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)
22
CLIPS> (send [Jill] delete)
<== instance [Jill] of PERSON TRUE
CLIPS> (unwatchinstances)
CLIPS>
```

Последовательность знаков <== указывает, что экземпляр удаляется, а последовательность знаков ==> свидетельствует о том, что экземпляр создаётся.

Пример:

;;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)))
CLIPS>
(defmessage-handler A print-all-slots ()
(printout t ?self:foo " "?self:bar crlf))
CLIPS> (make-instance a of A)
[a]
CLIPS> (send [a] print-all-slots)
1 2
CLIPS>
```

5.7. Контрольные вопросы и задания

1. Опишите основные элементы языка CLIPS.
2. Расскажите об основных типах данных языка.
3. Расскажите про конструкторы, используемые в языке CLIPS.
4. Абстракции данных.
5. Упорядоченные факты.
6. Неупорядоченные факты.
7. Функции для работы с фактами.
8. Поясните процесс инициализация фактов.

9. Расскажите о процессе использования локальных и глобальных переменных.

10. Создание правил. Конструктор defrule.

11. Поясните основной цикл выполнения правил.

12. Условные логические элементы.

13. Команды и функции для работы с правилами.

14. Функции, конструктор deffunction.

15. Конструктор defclass.

16. Расскажите об особенностях абстрактных и конкретных классов.

17. Расскажите об особенностях активных и неактивных классов.

18. Слоты класса.

19. Конструктор обработчика сообщений.

20. Работа с объектами.

21. Поясните процесс инициализации объектов.

22. Расскажите о функциях для работы с объектами.

23. Преобразуйте сеть общего вида, на которой представлены авиалинии, в ряд фактов, заданных в операторе deffacts. Для описания фактов используйте единственную конструкцию deftemplate.

24. Преобразуйте семантическую сеть, представляющую семью, в ряд фактов, заданных в операторе deffacts. Для описания сформированных фактов используйте несколько конструкций deftemplate.

25. Преобразуйте семантическую сеть классификации летательных аппаратов в ряд фактов, заданных в операторе deffacts. Для описания фактов используйте несколько конструкций deftemplate.

26. Преобразуйте бинарное дерево решений, представляющее информацию о классификации животных в ряд фактов, заданных в операторе deffacts.

27. Преобразуйте семантическую сеть классификации автомобилей в ряд фактов, заданных в операторе deffacts. Для описания фактов используйте несколько конструкций deftemplate.

28. Преобразуйте бинарное дерево решений, представляющее информацию о классификации растений в ряд фактов, заданных в операторе deffacts.

5.8. Список литературы

1. Адрес языка CLIPS в Интернете: http://www.ghg.net/clips/CLIPS.html.

2. Базы данных. Интеллектуальная обработка информации / В.В. Корнеев [и др.]. – М. : Нолидж, 2000.

3. Вахтин, А.А. Лабораторный практикум по программированию на языке CLIPS для курса «Представление знаний в информационных

системах» : учебно-методическое пособие для вузов / А.А. Вахтин, В.В. Гришина. – Издательско-полиграфический центр ВГУ, 2010. – 95 с.

4. Джарантино, Дж. Экспертные системы: принципы разработки и программирования / Дж. Джарантино, Г. Райли. – 4-е изд. ; пер. с англ. – М. : Изд. дом «Вильямс», 2007. – 1152 с.

5. Джексон, П. Введение в экспертные системы / П. Джексон ; пер. с англ. – М. : Изд. дом «Вильямс», 2001. – 622 с.

6. Люгер, Дж. Искусственный интеллект: стратегии и методы решения сложных проблем / Дж. Люгер, С. Рассел, П. Норвиг. – 4-е изд. ; пер. с англ. – М. : Изд. дом «Вильямс», 2003. – 864 с.

7. Рассел, С. Искусственный интеллект: современный подход / С. Рассел, П. Норвиг. – 2-е изд. ; пер. с англ. – М. : Изд. дом «Вильямс», 2006. – 1408 с.

8. Частиков, А.П. Разработка экспертных систем. Среда CLIPS / А.П. Частиков,Т.А. Гаврилов, Д.Л. Белов. – СПб. : БХВ-Петербург, 2003. – 608 с.

6. ПРИМЕРЫ РЕАЛИЗАЦИИ ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

6.1. Пример объектно-ориентированного программирования на языке CLIPS

Использование объектно-ориентированных средств в CLIPS позволяет значительно упростить программирование правил, поскольку для обновления данных можно применять механизм передачи и обработки сообщений методами классов. Мы продемонстрируем, как это делается на примере, который моделирует правила обращения с полуавтоматическим пистолетом.

Первым делом определим класс pistol, в котором будут перечислены свойства, необходимые для моделирования.

```
(defclass pistol
(is-a USER)
(role concrete)
(pattern-match reactive)
(slot safety (type SYMBOL) (create-accessor read-write))
(slot slide (type SYMBOL) (create-accessor read-write))
(slot hammer (type SYMBOL) (create-accessor read-write))
(slot chamber (type INTEGER) (create-accessor read-write))
(slot magazine (type SYMBOL) (create-accessor read-write))
(slot rounds (type INTEGER) (create-accessor read-write)))
```

Первые три слота – системные. Они нужны объектно-ориентированной надстройке CLIPS (COOL – CLIPS object–oriented language). Эти слоты COOL извещают о том, что pistol – это пользовательский класс; pistol является конкретным классом, т.е. возможно создание экземпляров этого класса.

Экземпляры класса pistol могут быть использованы в качестве объектов данных, которые можно сопоставлять с условиями в правилах и использовать в действиях, определённых правилами.

Следующие пять слотов представляют свойства и члены данных класса:

– слот safety (предохранитель) может содержать символ on или off;

– слот slide (затвор) может содержать значение forward или back, т.е. хранит информацию о положении затвора;

– слот hammer (курок) содержит информацию о состоянии курка, back или down;

– слот chamber (патронник) содержит значение 1 или 0, в зависимости от того, есть ли патрон в патроннике;

– слот magazine (обойма) может содержать значение in или out, в зависимости от того, вставлена ли обойма;

– слот rounds (патроны) содержит текущее количество патронов в обойме.

Для того чтобы иметь возможность записывать в слот новое значение или считывать текущее, нужно разрешить формирование соответствующих функций доступа через грань акцессоров create-accessor. Теперь сформируем экземпляр класса pistol с помощью следующего выражения:

```
(definstances pistols (PPK of pistol (safety on)
(slide forward) (hammer down) (chamber 0) (magazine out) (rounds 6)))
```

Этот экземпляр, PPK, правильно уложен – обойма вынута из рукоятки, пистолет установлен на предохранитель, затвор в переднем положении, курок опущен, а патронник пуст. В обойме имеется 6 патронов.

Теперь, имея в программе определение класса и сформировав экземпляр класса, разработаем правила и обработчики сообщений, с помощью которых можно описать отдельные операции обращения с пистолетом и стрельбы из него. Для этого сначала разработаем шаблон задачи. Желательно отслеживать две вещи:

– есть ли патрон в патроннике;
– произведён ли выстрел.

Для этого можно использовать следующий шаблон:

```
(deftemplate range-test
(field check (type SYMBOL) (default no))
(field fired (type SYMBOL) (default no)))
```

Первое правило будет устанавливать в рабочую память программы задачу range-test.

```
(defrule start
(initial-fact) =>
(assert (range-test)))
```

Приактивизации этого правила в рабочую память будет добавлено (range-test (chec kno) (fired no))

Следующие три правила будут проверять, правильно ли снаряжён пистолет.

```
(defrule check
(object (name [PPK]) (safety on) (magazine out))
?T<-(range-test (check no))
=>
(send [PPK] clear)
(modify ?T (check yes)))
```

Правило check заключается в том, что если пистолет стоит на предохранителе (safety on), обойма вынута (magazine out) и пистолет не был проверен, то нужно очистить патронник и проверить, нет ли в нём патрона. Обработчик сообщения clear для класса pistol будет выглядеть следующим образом:

```
(defmessage-handler pistol clear ()
(dynamic-put chamber 0)
(ppinstance))
```

В первой строке объявляется, что clear является обработчиком сообщения для класса pistol, причём этот обработчик не требует передачи аргументов. Оператор во второй строке «очищает» патронник. Присвоение выполняется независимо от того, какое текущее значение имеет слот chamber, – 0 или 1. Оператор в третьей строке требует, чтобы экземпляр распечатал информацию о текущем состоянии своих слотов.

В следующих двух правилах обрабатываются ситуации, когда пистолет снаряжён неправильно, – не установлен на предохранитель

или в него вставлена обойма. Правило correctl устанавливает пистолет на предохранитель, а правило correct2 извлекает из него обойму.

```
(defrule correctl
(object (name [PPK]) (safety off))
(range-test (check no)) =>
(send [PPK] safety on))

(defrule correct2
(object (name [PPK]) (safety on) (magazine in))
(range-test (check no)) =>
(send [PPK] drop))
```

Как и при разработке предыдущего правила, нам понадобятся обработчики сообщений safety и drop.

```
(defmessage-handler pistol safety (?on-off)
(dynamic-put safety ?on-off)
(if (eq ?on-off on) then (dynamic-put hammer down)))
```

Обработчик сообщения safety принимает единственный аргумент, который может иметь только два символических значения on или off. В противном случае нам пришлось бы разработать два обработчика: один для сообщения safety-on, а другой – для сообщения safety-off .

Обработчик сообщения drop просто извлекает обойму из пистолета.

```
(defmessage-handler pistol drop ()
(dynamic-put magazine out)
)
```

Теперь, когда обеспечено правильное исходное снаряжение пистолета, можно приступить к стрельбе. Следующее правило обеспечивает вставку обоймы в пистолет перед стрельбой:

```
(defrulemag-in
(object (name [PPK]) (safety on) (magazine out))
(range-test (fired no) (check yes)) =>
(send [PPK] seat))
```

Обработчик сообщения seat выполняет действия, противоположные тем, которые выполняет обработчик drop.

```
(defmessage-handler pistol seat ()
(dynamic-put magazine in))
```

Следующее правило обеспечивает снаряжение обоймы патронами:

```
(defrule load
(object (name [PPK]) (magazine in) (chamber 0)) =>
(send [PPK] rack))
```

Обработчик сообщения rack.

```
(defmessage-handler pistol rack ()
(bind ?a (dynamic-get rounds))
(if (> ?a 0) then
(dynamic-put chamber 1)
(dynamic-put rounds (- ?a 1))
(dynamic-put slide forward)
else (dynamic-put chamber 0)
(dynamic-put slide back)))
```

В этом обработчике обеспечивается досылка патрона в патронник в том случае, если в обойме имеются патроны. Следующее правило подготавливает пистолет к стрельбе, снимая его с предохранителя. Обратите внимание на то, что в нём повторно используется сообщение safety, но на этот раз с аргументом off.

```
(defrule ready
(object (name [PPK]) (chamber 1))
=>
(send [PPK] safety off))
```

Правило fire выполняет стрельбу.

```
(defrule fire
(object (name [PPK]) (safety off))
?T <- (range-test (fired no)) =>
(if (eq (send [PPK] fire) TRUE)
then (modify ?T (fired yes))))
```

Обратите внимание, что в данном правиле используется обработчик сообщения, которое возвращает значение. Анализируя его, можно выяснить, произведён ли выстрел, т.е. выполнена ли в действительности та операция, которая «закреплена» за этим сообщением. Если в патроннике был патрон и пистолет был снят с предохранителя, то обработчик сообщения вернёт значение TRUE (после того, как выведет на экран BANG!). В противном случае он вернёт FALSE (после того, как выведет на экран click).

```
(defmessage-handler pistol fire ()
(if (and (eq (dynamic-get chamber) 1) (eq (dynamic-get safety) off))
then (printout t "BANG!" crlf) TRUE
else (printout t "click" crlf) FALSE))
```

Пусть вас не смущает, что в обработчике сообщения анализируется условие, которое уже было проанализировано правилом, отославшим сообщение (в данном случае речь идёт об условии safety off). Дело в том, что одно и то же сообщение может отсылаться разными правилами и нет никакой гарантии, что в каждом из них будет проверяться это условие.

Рассмотрим ещё один пример:

```
(defclass RECTANGLE
      (is-a USER)
      (slot height)
      (slot width))
(defclass CIRCLE
      (is-a USER)
      (slot radius))
(defmessage-handler RECTANGLE compute-area ()
      (* (send ?self get-height)
      (send ?self get-width)))
(defmessage-handler CIRCLE compute-area ()
      (* (pi)
      (send ?self get-radius)
      (send ?self get-radius)))
(definstances figures
      (rectangle-1 of RECTANGLE (height 2) (width 4))
      (circle-1 of CIRCLE (radius 3)))
```

В данном примере определяются два класса, RECTANGLE и CIRCLE, с соответствующими слотами. За каждым классом закреплён обработчик сообщений compute-area. Этот обработчик сообщений предназначен для вычисления площади каждого объекта. Для класса RECTANGLE площадь экземпляра RECTANGLE вычисляется путём умножения высоты, заданной в экземпляре, на ширину. А для класса CIRCLE площадь экземпляра CIRCLE представляет собой значение числа π, возвращаемое функцией pi, которое умножается на значение радиуса, заданное в экземпляре, возведённое в квадрат. Обратите внимание на то, что в обоих обработчиках сообщений используется переменная ?self. Это – специальная переменная, автоматически определяемая для каждого обработчика сообщений. При вызове обработчика сообщений переменной ?self присваивается значение адреса того эк-

земпляра, которому передаётся сообщение. Эта переменная может применяться для передачи в экземпляр сообщений, как и было сделано в рассматриваемом примере для выборки значений слотов height, width и radius.

После определения обработчиков сообщений появляется возможность отправлять сообщения compute-area в экземпляры классов RECTANGLE и CIRCLE для получения информации о площади фигуры, заданной этим экземпляром, как в следующем примере:

```
CLIPS> (reset)
CLIPS> (send [circle-1] compute-area)
28.2743338823081
CLIPS>
(send [rectangle-1] compute-area)
8
CLIPS>
```

Здесь заслуживает внимания то, что каждому экземпляру передаётся одно и то же сообщение, но, вычисляя площадь фигуры, заданной с его помощью, каждый экземпляр отвечает по-разному. Такая способность различных экземпляров отвечать на одно и то же сообщение в характерной для него форме называется полиморфизмом.

6.2. Использование семантических сетей для представления знаний на языке CLIPS

Данный пример наглядно демонстрирует работу с фактами и правилами.

Описание структуры. Создадим шаблон для неупорядоченных фактов.

Для описания структуры генеалогического дерева (рис. 6.1) достаточно четыре слота:

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

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

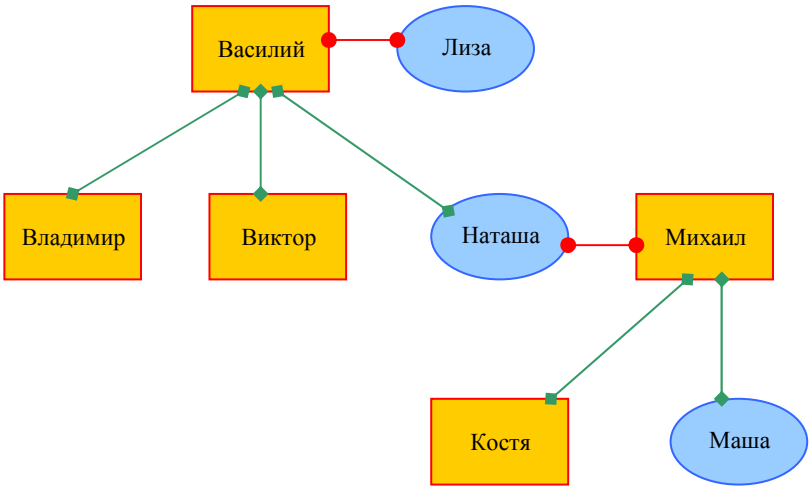


Рис. 6.1. Пример генеалогического дерева

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

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

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

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

Определение отношений. Определим отношение «Мать» (рис. 6.2).

Создадим шаблон:

```
(deftemplate mother
(slot namel)
(slot name2))
```

Создадим правило, описывающее отношение:

```
(defrule Mother
(PERSON (name ?x) (wife ?y))
(PERSON (name ?z) (father ?x))
=>
(printout t ?y " is mother of " ?z crlf)
(assert (mother (namel ?y) (name2 ?z))))
```

Выполним команды:

```
CLIPS> (reset)
CLIPS> (run)
```

Результат:

```
CLIPS>Natasha mother of Masha
Natasha mother of Kostya
Liza mother of Viktor
Liza mother of Natasha
Liza mother of Vladimir
```

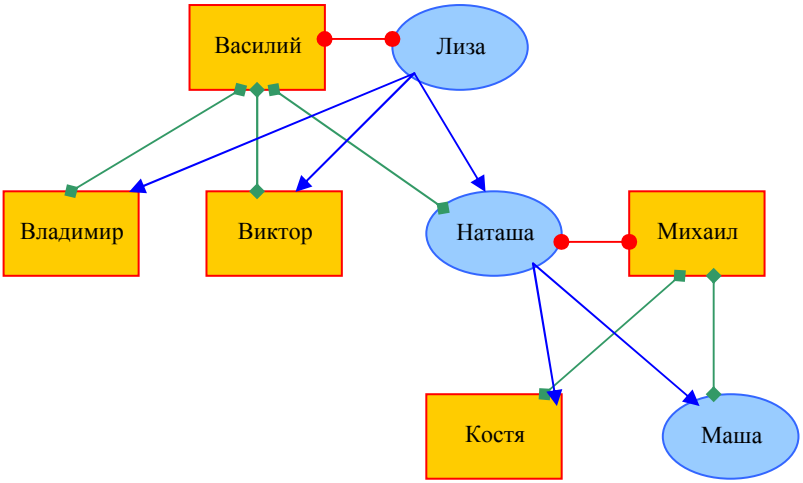


Рис. 6.2. Отображение отношения «Мать»

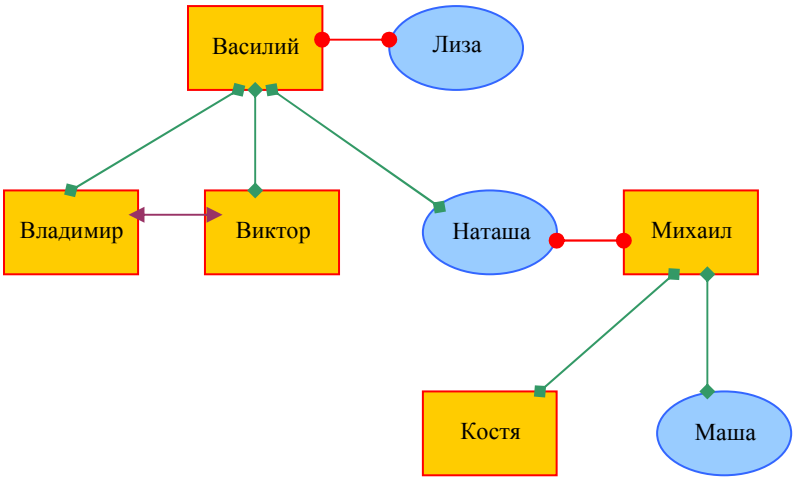


Рис. 6.3. Отображение отношения «Брат»

Определим отношение «Брат» (рис. 6.3).
Создадим шаблон:

```
(deftemplate brother
(slot namel)
(slot name2))
```

Создадим правило, описывающее отношение:

```
(defrule Brother
(PERSON (name ?x) (gender male) (father ?y&~nil))
(PERSON (name ?z&~?x) (gender male) (father ?y&~nil))
(not (brother (namel ?x) (name2 ?z)))
(not (brother (namel ?z) (name2 ?x))) =>
(printout t ?x " brother of " ?z crlf)
(assert (brother (namel ?x) (name2 ?z))))
```

Ограничение ?z&~?x запрещает выводить бессмысленные пары одинаковых имён. Ограничение ?y&~nil запрещает выводить пары, поля «отец» которых не определены (нулевое значение).

Условные элементы:

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

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

Выполним команды:

```
CLIPS> (reset)
CLIPS> (run)
```

Результат:

```
CLIPS>Viktor brother of Vladimir
```

6.3. Пример учёта неопределённости на языке CLIPS

В языке CLIPS непосредственно не предусмотрены какие-либо возможности учёта неопределённости. Тем не менее в программу CLIPS несложно включить средства учёта неопределённости, помещая информацию, касающуюся неопределённости, непосредственно в факты и правила [4]. В качестве примера достаточно указать, что с помощью языка CLIPS может быть эмулирован механизм учёта неопределённости, применяемый в системе MYCIN. Ниже будет показано, как можно перезаписать на языке CLIPS следующее правило MYCIN:

```
IF
The stain of the organism is gramneg and
The morphology of the organism is rod and
The patient is a compromised host
THEN
There is suggestive evidence (0.6) that the identity of the organism is
pseudomonas
```

В системе MYCIN фактическая информация представлена в виде троек «объект–атрибут–значение» (Object–Attribute–Value – OAV). Такие тройки OAV могут быть представлены в языке CLIPS с помощью следующей конструкции deftemplate (эта конструкция будет помещена в собственный модуль в целях создания повторно применимого программного компонента):

```
(defmodule OAV (export deftemplate oav))
(deftemplate OAV::oav
(multislot object (type SYMBOL))
```

```
(multislot attribute (type SYMBOL))
(multislot value))
```

Эта конструкция deftemplate позволяет представить некоторые факты, требуемые для части IF приведённого выше правила MYCIN, следующим образом:

```
(OAV (object organism)
(attribute stain)
(value gramneg))
(OAV (object organism)
(attribute morphology)
(value rod))
(OAV (object patient)
(attribute is a)
(value compromised host))
```

Кроме того, в системе MYCIN с каждым фактом ассоциируется коэффициент достоверности (Certainty Factor – CF), который характеризует степень доверия к факту. Коэффициент достоверности может иметь значение от –1 до 1; значение –1 показывает, что факт является заведомо ложным, значение 0 говорит о том, что какая-либо информация об этом факте отсутствует (налицо полная неопределённость), а значение 1 свидетельствует, что факт является заведомо истинным.

В системе CLIPS коэффициенты достоверности не учитываются автоматически, поэтому необходимо обеспечить сопровождение и данной информации. В этих целях в каждом факте будет использоваться дополнительный слот, представляющий коэффициент достоверности. После этого конструкция deftemplate с именем OAV для каждого факта принимает такой вид:

```
(deftemplate OAV::oav
(multislot object (type SYMBOL))
(multislot attribute (type SYMBOL))
(multislot value)
(slot CF (type FLOAT) (range -1.0 +1.0)))
```

В качестве примеров фактов можно привести следующее:

```
(OAV (object organism)
(attribute stain)
(value gramneg)
(CF 0.3))
(oav (object organism)
(attribute morphology)
```

(value rod) (CF 0.7))
(OAV (object patient)
(attribute is a)
(valuecompromisedhost)
(CF 0.8))

Для того чтобы факты OAV функционировали должным образом, в программу на языке CLIPS необходимо внести ещё одну модификацию. Система MYCIN позволяет осуществить логический вывод одних и тех же троек OAV с помощью отдельных правил. Затем эти тройки OAV комбинируются для получения единственной тройки OAV, в которой комбинируются коэффициенты достоверности исходных троек OAV. Применяемая в настоящее время конструкция deftemplate с именем OAV позволяет вносить в список фактов две идентичные тройки OAV только в том случае, если в них имеются различные коэффициенты достоверности (поскольку система CLIPS в обычных условиях не позволяет вносить в список фактов два дублирующихся факта). Для того чтобы обеспечить возможность внесения в список фактов идентичных троек OAV, имеющих одинаковые коэффициенты достоверности, можно использовать команду set-fact-duplication для отмены применяемого в системе CLIPS принципа работы, согласно которому предотвращается внесение дублирующихся фактов в список фактов. Указанный принцип действия отменяется с помощью команды, имеющей следующий синтаксис: (set-fact-duplication TRUE)

Аналогичным образом, команда, имеющая следующую форму, исключает возможность внесения в список фактов дублирующихся фактов:

(set-fact-duplication FALSE)

Как уже было сказано, в системе MYCIN две идентичные тройки OAV комбинируются в одну тройку OAV, имеющую комбинированное значение коэффициента достоверности. Для вычисления нового коэффициента достоверности в системе MYCIN используется следующая s-норма, если оба коэффициента достоверности двух фактов (обозначенные как CF1 и CF2) больше или равны нулю:

New Certainty = (CF1 + CF2) – (CF1 * CF2)

Например, предположим, что в списке фактов имеются следующие факты:

(OAV (object organism)
(attribute morphology)

```
(value rod)
(CF 0.7))
(oav (object organism)
(attribute morphology)
(value rod)
(CF 0.5))
```

Допустим, что CF1 обозначает коэффициент достоверности первого факта, равный 0.7, а CF2 – коэффициент достоверности второго факта, равный 0.5; в таком случае новый коэффициент достоверности для комбинации этих двух фактов вычисляется таким образом:

New Certainty = (0.7 + 0.5) – (0.7 * 0.5) = 1.2 – 0.35 = 0.85,

а новый факт, заменяющий два первоначальных факта, принимает следующий вид:

```
(OAV (object organism)
(attribute morphology) (value rod) (CF 0.85))
```

Как уже было сказано, система CLIPS не обрабатывает автоматически коэффициенты достоверности, относящиеся к фактам. Из этого следует, что CLIPS также не комбинирует автоматически две тройки OAV, полученные с помощью разных правил. Но комбинирование троек OAV можно легко обеспечить с помощью правила, которое осуществляет поиск в списке фактов идентичных троек OAV, подлежащих комбинированию. Ниже показано правило и описан метод, которые демонстрируют, как осуществляются указанные действия применительно к таким попарно обрабатываемым тройкам OAV, в которых коэффициенты достоверности больше или равны нулю.

```
(defmethod OAV::combine-certainties
((?C1 NUMBER (> ?C1 0))(?C2 NUMBER (> ?C2 0)))
(- (+ ?C1 ?C2)(* ?C1 ?C2)))
(defrule OAV::combine-certainties (declare (auto-focus TRUE))
?factl <- (oav (object $?o)
(attribute $?a) (value $?v)
(CF ?C1))
?fact2 <- (oav (object $?o)
(attribute $?a) (value $?v) (CF ?C2))
(test (neq ?factl ?fact2))
=>
(retract ?factl) (modify ?fact2 (CF (combine-certainties ?C1 ?C2))))
```

Обратите внимание на то, что идентификаторы фактов ?fact1 и ?fact2 сравниваются друг с другом в условном элементе test. Такая операция применяется для получения гарантий того, что правило не согласовано с фактом с использованием точно такого же факта для первых двух шаблонов. Адреса фактов позволяют сравнить функции eq и neq. Кроме того, следует отметить, что для данного правила разрешён атрибут auto-focus. Это позволяет гарантировать, что две тройки OAV будут скомбинированы, прежде чем будет разрешён запуск других правил, шаблонам которых соответствуют обе эти тройки.

Следующим шагом на пути к внедрению средств поддержки коэффициентов достоверности в систему CLIPS является связывание коэффициентов достоверности фактов, согласующихся с левой частью правила, с коэффициентами достоверности фактов, внесённых в список фактов с помощью правой части правила. В системе MYCIN логический вывод коэффициента достоверности, ассоциирующегося с левой частью правила, осуществляется с использованием следующих формул:

CF(P1 or P2) = max{CF(P1),CF(P2)}
CF(P1 and P2) = min{CF(P1),CF(P2)}
CF(notP) = -CF(P)

В этих формулах P, P1 и P2 обозначают шаблоны из левой части правила. Кроме того, если коэффициент достоверности в левой части правила меньше 0.2, то правило рассматривается как неприменимое и запуск его не происходит.

Логический вывод значения коэффициента достоверности факта, внесённого в список фактов под действием правой части правила, осуществляется путём умножения коэффициента достоверности вносимого факта на коэффициент достоверности, заданный в левой части правила. Ниже приведён результат преобразования правила MYCIN. Это правило показывает, как вычисляются коэффициенты достоверности в левой и правой частях правила. Правило помещается в модуль IDENTIFY, который импортирует конструкцию deftemplate с именем OAV из модуля OAV.

```
(defmodule IDENTIFY (import OAV deftemplate oav))
(defrule IDENTIFY::MYCIN-to-CLIPS-translation
(OAV (object organism) attribute stain) value gramneg) CF ?C1))
(OAV (object organism) attribute morphology) value rod) CF ?C2))
(OAV (object patient) attribute is a) value compromised host) CF ?C3))
(test (> (min ?C1 ?C2 ?C3) 0.2))
=>
```

(bind ?C4 (* (min ?C1 ?C2 ?C3) 0.6))
(assert (OAV (object organism) (attribute identity) (value pseudomonas)
(CF ?C4))))

6.4. Примеры экспертных систем, написанных на языке CLIPS

У каждого специалиста, занимающегося диагностикой и устранением неисправностей принтеров, накоплен уникальный опыт, но он не является исчерпывающим. Возникает необходимость объединения подобного опыта для качественного улучшения диагностирования и устранения неисправностей принтеров. В Приложении А приведён код экспертной системы диагностики неисправностей принтеров. Используемый язык для создания внешнего интерфейса – Python. Разработанная экспертная система позволяет объединить различные алгоритмы диагностирования принтеров. Данную экспертную систему могут использовать мастерские, занимающиеся ремонтом принтеров. Система позволит специалистам ускорить процесс поиска неисправности устройства. Содержащиеся в экспертной системе знания помогут начинающим специалистам получить опыт в диагностике и устранении неисправностей принтеров.

В Приложении Б приведён код экспертной системы, которая помогает пользователю с выбором вакансии. Пользователю предлагается заполнить анкету – ему задаются вопросы, ответы на которые система посылает в виде сообщений объекту, представляющего пользователя. После опроса в системе накапливаются знания об объекте-пользователе, по которым определяется список подходящих ему вакансий. В итоге пользователю выдаётся результат в виде списка вакансий, на которые он может быть устроен, либо отказ в свободной вакансии. Для создания внешнего интерфейса используются языки HTML, CSS, PHP.

6.5. Контрольные вопросы и задания

1. В режиме командной строки вычислите значения выражений:

а) $(4^2 – 5) * (3 + 4)$.

б) $\sin 1 + 1/(\cos (1 – 2))$.

в) $\min (\max (4^3, 6^2), \min (2^5,5^2))$.

г) $(7 + 9) * \tan 5$.

д) $(5 * (5 + 6 + 7)) – ((3 * 4/9 + 2) / 9)$.

2. Создайте функцию для вычисления длины отрезка по заданным координатам его концов (a_1, a_2) и (b_1, b_2), $D = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$.

3. Создайте функцию для вычисления площади треугольника по длинам его сторон, $S = \sqrt{p(p-a)(p-b)(p-c)}$ (использовать отдельную функцию для вычисления полупериметра).

4. Напишите программу CLIPS, которая складывает два двоичных числа без использования каких-либо арифметических функций. Используйте для представления двоичных чисел следующую конструкцию deftemplate.

5. Напишите программу CLIPS, которая запрашивает у пользователя значения цветов, а затем выводит список всех государств, флаги которых содержат все указанные цвета.

6. Напишите программу, которая будет считывать файл данных, содержащий список имён людей с указанием возрастов, и создавать новый файл, в котором содержится тот же список, отсортированный в порядке увеличения возрастов.

7. Напишите программу, которая после получения значений координат двух точек на плоскости определяет наклон прямой, проходящей через эти две точки. Программа должна выполнять проверку для определения того, что координаты точек заданы числами и что одна и та же точка не указана дважды. Линии, направленные перпендикулярно горизонтальной оси, следует рассматривать как имеющие бесконечный наклон.

8. Напишите программу для поиска решения задачи с ханойскими башнями, в которой необходимо переместить ряд колец, имеющих разный наружный диаметр и одинаковый внутренний диаметр, с одного колышка на другой колышек, ни разу не насаживая на колышек кольцо с большим наружным диаметром поверх кольца с меньшим наружным диаметром.

9. Напишите программу, позволяющую определить цифровые значения букв, после подстановки которых следующая задача решается правильно. Каждой из букв H, O, C, U, S, P, R, E и T соответствует уникальная цифра от 0 до 9.

HOCUS
+ POCUS
= PRESTO

10. Напишите программу, позволяющую определить простые множители числа. Например, простыми множителями числа 15 являются 3 и 5.

11. Напишите программу для преобразования сообщения, заданного в виде азбуки Морзе, в эквивалентный этому сообщению ряд знаков алфавита.

12. Напишите программу для ведения игры Жизнь.

13. Пусть множество E = {1, 2, …, 100} определяет возраст человека. Подмножество A «Молодой» можно задать функцией принадлежности:

$$\mu(A) = \begin{cases} 1, & x \le 25 \\ \dfrac{1}{1+\left(\dfrac{x-25}{5}\right)^2}, & x > 25. \end{cases}$$

Напишите программу, которая по возрасту человека определяет, к какой категории он относится:

молодой

$$\mu(A) = 1 ;$$

среднего возраста

$$0{,}03 \le \mu(A) < 1 ;$$

старый

$$\mu(A) < 0{,}03 .$$

14. Напишите программу, определяющую является ли число *n* (номер варианта) простым.

15. Постройте генеалогическое дерево своей семьи для трёх поколений. Определить следующие отношения: мать, брат, сестра, дедушка, бабушка, тёща, шурин (брат жены), свояченица (сестра жены), свояк (муж свояченицы), свёкор (отец мужа), золовка (сестра мужа), деверь (брат мужа), сноха (жена сына для его матери), невестка (жена сына для его отца).

16. Предположим, что дана шахматная доска размерами $N \times N$, где N – целое число. Напишите программу, которая расставляет N ферзей на шахматной доске таким образом, что ни один ферзь не может напасть на другого.

17. Напишите конструкцию deffunction, которая определяет все простые числа от 1 до указанного целого числа и возвращает эти простые числа в виде многозначного значения.

18. Напишите конструкцию deffunction, которая определяет количество вхождений одной строки в другой строке.

19. Напишите конструкцию deffunction, которая построчно сравнивает два файла и выводит информацию об обнаруженных различиях в файл, указанный логическим именем.

20. Напишите конструкцию deffunction, которая принимает от нуля и больше параметров и возвращает многозначное значение, содержащее значения параметров в обратном порядке.

21. Напишите конструкцию deffunction, в которой не используется рекурсия для вычисления факториала целого числа N.

22. Напишите конструкцию deffunction, которая преобразовывает двоичную строку, состоящую из нулей и единиц, в десятичное число.

23. Без использования функций if или switch напишите ряд методов, предназначенных для преобразования данных, представленных с помощью таких единиц, как дюймы (inches), футы (feet) и ярды (yards).

24. Разработайте экспертную систему для выбора университета, факультета и специальности в соответствии с интересами абитуриента.

25. Разработайте экспертную систему, используя принципы объектно-ориентированного программирования.

6.6. Список литературы

1. Адрес языка CLIPS в Интернете: http://www.ghg.net/clips/CLIPS.html.

2. Базы данных. Интеллектуальная обработка информации / В.В. Корнеев [и др.]. – М. : Нолидж, 2000.

3. Вахтин, А.А. Лабораторный практикум по программированию на языке CLIPS для курса «Представление знаний в информационных системах» : учебно-методическое пособие для вузов / А.А. Вахтин, В.В. Гришина. – Издательско-полиграфический центр ВГУ, 2010. – 95 с.

4. Джарантино, Дж. Экспертные системы: принципы разработки и программирования / Дж. Джарантино, Г. Райли. – 4-е изд. ; пер. с англ. – М. : Изд. дом «Вильямс», 2007. – 1152 с.

5. Джексон, П. Введение в экспертные системы / П. Джексон ; пер. с англ. – М. : Изд. дом «Вильямс», 2001. – 622 с.

6. Люгер, Дж. Искусственный интеллект: стратегии и методы решения сложных проблем / Дж. Люгер, С. Рассел, П. Норвиг. – 4-е изд. ; пер. с англ. – М. : Изд. дом «Вильямс», 2003. – 864 с.

7. Рассел, С. Искусственный интеллект: современный подход / С. Рассел, П. Норвиг. – 2-е изд. ; пер. с англ. – М. : Изд. дом «Вильямс», 2006. – 1408 с.

8. Частиков, А.П. Разработка экспертных систем. Среда CLIPS / А.П. Частиков, Т.А. Гаврилов, Д.Л. Белов. – СПб. : БХВ-Петербург, 2003. – 608 с.

ЗАКЛЮЧЕНИЕ

Поскольку в сферу приложений искусственного интеллекта вошли практически все направления современной информатики, мы не ставили перед собой цели «объять необъятное», а включили в учебник разделы, содержащие описание традиционных моделей и технологий создания интеллектуальных систем, а также новых перспективных подходов к решению проблем, возникающих в области искусственного интеллекта.

При написании книги мы видели свою цель в том, чтобы познакомить читателя с принципами создания и функционирования интеллектуальных информационных систем. Практический опыт показывает, что люди, не знакомые с этими принципами, испытывают большие трудности, выступая в роли пользователей интеллектуального программного обеспечения.

При всём многообразии видов интеллектуального программного обеспечения, имеющегося в настоящее время, мы достаточно подробно рассмотрели среду CLIPS. Это объясняется её доступностью и тем, что язык и среда CLIPS предоставляют пользователям возможность быстро создавать эффективные, компактные и легко управляемые экспертные системы. Несмотря на то, что CLIPS распространяется бесплатно, он весьма успешно конкурирует даже с самыми известными коммерческими проектами.

Научиться программировать можно только программируя, решая конкретные задачи. В приложения учебника мы целенаправленно включили сравнительно большие по объёму практические примеры экспертных систем на языке CLIPS. Поэтому разбирайте примеры, ставьте себе задачи и программируйте.

Авторы надеются, что знания, полученные при изучении данной книги, и приведённые практические примеры помогут читателю в самостоятельном освоении языка CLIPS, а также позволят расширить кругозор в области интеллектуальных информационных систем и технологий.

ПРИЛОЖЕНИЯ

Приложение А

Файл kurs.clp

```
;;;;;;;;;;;;;;;;;;;;
;Определение классов;
;;;;;;;;;;;;;;;;;;;;

;;Определение абстрактного класса принтера;;
(defclass PRINTER "Printer defclass"
        (is-a USER)
        (role abstract)
    (slot power
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot podkl_el
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot shnur
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot napr
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot print_doc
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
;;Определение конкретного класса лазерного принтера;;
(defclass PRINTER_LASER "Laser printer defclass"
        (is-a PRINTER)
        (role concrete)
        (pattern-match reactive)
    (slot zastr_cart
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
```

```
    (slot zastr_vyhod
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot zaderzh
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot shum
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot kachestvo
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot install_skrip
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot opred_cart
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot opred_bum
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))
    (slot zahvat_bum
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))

;;Определение конкретного класса струйного принтера;;
(defclass PRINTER_INK "Inkjet printer defclass"
    (is-a PRINTER)
    (role concrete)
    (pattern-match reactive)
```

```
(slot otkl_val
    (type SYMBOL)
    (create-accessor read-write)
    (storage local)
    (default none))
(slot skrip
    (type SYMBOL)
    (create-accessor read-write)
    (storage local)
    (default none))
(slot bum_perekos
    (type SYMBOL)
    (create-accessor read-write)
    (storage local)
    (default none))
(slot nesk_listov
    (type SYMBOL)
    (create-accessor read-write)
    (storage local)
    (default none))
(slot zastr_raboch
    (type SYMBOL)
    (create-accessor read-write)
    (storage local)
    (default none))
(slot kachestvo
    (type SYMBOL)
    (create-accessor read-write)
    (storage local)
    (default none))
(slot caret_upor
    (type SYMBOL)
    (create-accessor read-write)
    (storage local)
    (default none))
(slot coord_lenta
    (type SYMBOL)
    (create-accessor read-write)
    (storage local)
    (default none))
(slot opred_cart
    (type SYMBOL)
    (create-accessor read-write)
    (storage local)
    (default none))
```

```
    (slot pod_bum
        (type SYMBOL)
        (create-accessor read-write)
        (storage local)
        (default none))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Определение шаблона, описывающего текущее состояние системы;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(deftemplate UI-state
    (slot cur_quest (type STRING))
    (multislot ans (type STRING))
    (multislot sys_ans (type SYMBOL))
    (slot user_eval (type STRING))
    (slot prev_q (type SYMBOL) (default no))
    (slot prev_ans (type STRING))
    (slot state (type SYMBOL)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Определение функции, сообщающей системе о нажатии кнопки перехода к предыдущему вопросу;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(deffunction prev_q_yes (?f)
    (modify ?f (prev_q yes)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Определение правил вывода вопросов системы;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule start
    (initial-fact)
    =>
    (assert (UI-state (cur_quest "Добро пожаловать в экспертную систему диагностики неисправности принтера! Для продолжения нажмите кнопку \"Далее\".")
                      (ans)
                      (sys_ans)
                      (user_eval "")
                      (prev_ans "")
                      (state start)))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Общие вопросы для диагностики неисправности принтера;
(defrule q_type_printer
    ?F<-(UI-state (state start))
    =>
```

```
    (modify ?F (cur_quest "Укажите тип принтера:")
               (ans "лазерный" "струйный")
               (sys_ans PRINTER_LASER PRINTER_INK)
               (user_eval  "make-instance  [cur_printer]
of ")
               (prev_ans "")
               (state diag))
    (halt))
(defrule q_type_printer_prev
    ?F<-(UI-state  (cur_quest  "Укажите  тип  принтера:")
(prev_q yes))
    =>
    (modify ?F (cur_quest "Добро пожаловать в экспертную
систему диагностики неисправности принтера! Для продол-
жения нажмите кнопку \"Далее\".")
                       (ans)
                       (sys_ans)
                       (user_eval "")
                       (prev_q no)
                       (prev_ans "")
                       (state start))
    (halt))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_power
    (object (is-a ?x) (name [cur_printer]) (power none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Включается ли принтер?")
               (ans "Нет" "Да")
               (sys_ans no yes)
              (user_eval "send [cur_printer] put-power ")
               (prev_ans "")
               (state diag))
    else
    (if (eq ?x PRINTER_LASER)
     then
    (modify ?F (cur_quest "Укажите тип принтера:")
               (ans "лазерный" "струйный")
               (sys_ans PRINTER_LASER PRINTER_INK)
               (user_eval  "make-instance  [cur_printer]
of ")
               (prev_q no)
               (prev_ans "лазерный")
               (state diag))
```

```
    (if (eq ?x PRINTER_INK)
     then
    (modify ?F (cur_quest "Укажите тип принтера:")
               (ans "лазерный" "струйный")
               (sys_ans PRINTER_LASER PRINTER_INK)
               (user_eval  "make-instance  [cur_printer]
of ")
               (prev_q no)
               (prev_ans "струйный")
               (state diag)))
    (send [cur_printer] delete))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_podkl_el
    (object  (name  [cur_printer])  (power  no)  (podkl_el
none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Подключён ли принтер к элек-
трической сети?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval   "send   [cur_printer]   put-
podkl_el ")
               (prev_ans "")
               (state diag))
    else
    (modify ?F (cur_quest "Включается ли принтер?")
               (ans "Нет" "Да")
               (sys_ans no yes)
              (user_eval "send [cur_printer] put-power ")
               (prev_q no)
               (prev_ans "Нет")
               (state diag))
    (send [cur_printer] put-power none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_shnur
    (object  (name  [cur_printer])  (power  no)  (podkl_el
yes) (shnur none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
```

```
    then
    (modify ?F (cur_quest "Исправен ли шнур питания
принтера?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-shnur ")
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "Подключён ли принтер к элек-
трической сети?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
podkl_el ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-podkl_el none))
    (halt))

(defrule r_podkl_el
    (object (name [cur_printer]) (power no) (podkl_el
no))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Для устранения неисправности
система предлагает Вам подключить принтер к электросети.
Если не все проблемы устранены, попробуйте выполнить ди-
агностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Подключён ли принтер к элек-
трической сети?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
podkl_el ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
```

```
    (send [cur_printer] put-podkl_el none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_napr
    (object (name [cur_printer]) (power no) (podkl_el
yes) (shnur yes) (napr none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Есть ли напряжение в электри-
ческой сети?")
                (ans "Нет" "Да")
                (sys_ans no yes)
               (user_eval "send [cur_printer] put-napr ")
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "Исправен ли шнур питания
принтера?")
                (ans "Нет" "Да")
                (sys_ans no yes)
               (user_eval "send [cur_printer] put-shnur ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-shnur none))
    (halt))

(defrule r_shnur
    (object (name [cur_printer]) (power no) (podkl_el
yes) (shnur no))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Для устранения неисправности
система предлагает заменить сетевой шнур. Если не все
проблемы устранены, попробуйте выполнить диагностику за-
ново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
```

```
    (modify ?F (cur_quest "Исправен ли шнур питания
принтера?")
                (ans "Нет" "Да")
                (sys_ans no yes)
               (user_eval "send [cur_printer] put-shnur ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-shnur none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule r_napr_yes
    (object (name [cur_printer]) (power no) (podkl_el
yes) (shnur yes) (napr yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Неисправен блок питания прин-
тера. Для устранения неисправности система предлагает
заменить его. Если не все проблемы устранены, попробуйте
выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Есть ли напряжение в электри-
ческой сети?")
                (ans "Нет" "Да")
                (sys_ans no yes)
               (user_eval "send [cur_printer] put-napr ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-napr none))
    (halt))

(defrule r_napr_no
    (object (name [cur_printer]) (power no) (podkl_el
yes) (shnur yes) (napr no))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
```

```
    (modify ?F (cur_quest "Для устранения неисправности
система предлагает подать напряжение в электрическую
сеть. Если не все проблемы устранены, попробуйте выпол-
нить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Есть ли напряжение в электри-
ческой сети?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-napr ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-napr none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Вопросы для диагностики неисправности лазерного прин-
тера;;;;;;;
(defrule q_print_doc
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Выполняется ли печать доку-
мента?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-
print_doc ")
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "Включается ли принтер?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-power ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
```

```
    (send [cur_printer] put-power none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_zastr_cart
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc yes) (zastr_cart none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "После прохождения картриджа
листы бумаги застревают?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval   "send   [cur_printer]   put-
zastr_cart ")
               (prev_ans "")
               (state diag))
    else
    (modify ?F (cur_quest "Выполняется ли печать доку-
мента?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval   "send   [cur_printer]   put-
print_doc ")
               (prev_q no)
               (prev_ans "Да")
               (state diag))
    (send [cur_printer] put-print_doc none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_zastr_vyhod
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Застревают ли листы бумаги на
выходе из принтера?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval   "send   [cur_printer]   put-
zastr_vyhod ")
```

```
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "После прохождения картриджа
листы бумаги застревают?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
zastr_cart ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-zastr_cart none))
    (halt))

(defrule r_zastr_cart
    (object  (is-a  PRINTER_LASER)  (name  [cur_printer])
(power yes) (print_doc yes) (zastr_cart yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Дефект вызван тем, что за-
грязнились резиновые ремни подачи бумаги в фьюзер. Для
устранения неисправности система предлагает снять ремни
с приводных роликов и очистить. Если не все проблемы
устранены, попробуйте выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "После прохождения картриджа
листы бумаги застревают?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
zastr_cart ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-zastr_cart none))
    (halt))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_zaderzh
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Принтер печатает с большой
задержкой?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval "send [cur_printer] put-
zaderzh ")
               (prev_ans "")
               (state diag))
    else
    (modify ?F (cur_quest "Застревают ли листы бумаги на
выходе из принтера?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval "send [cur_printer] put-
zastr_vyhod ")
               (prev_q no)
               (prev_ans "Нет")
               (state diag))
    (send [cur_printer] put-zastr_vyhod none))
    (halt))

(defrule r_zastr_vyhod
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Причина подобного дефекта вы-
звана загрязнением или ослаблением прижима роликов выхо-
да бумаги. Для устранения неисправности система предла-
гает очистить резиновую поверхность ролика и проверить
пружины на ребристых пластмассовых роликах. Если не все
проблемы устранены, попробуйте выполнить диагностику за-
ново.")
               (ans)
               (sys_ans)
```

```
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Застревают ли листы бумаги на
выходе из принтера?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
zastr_vyhod ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-zastr_vyhod none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_shum
    (object  (is-a  PRINTER_LASER)  (name  [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh no) (shum none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Слышен ли повышенный шум при
работе принтера?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-shum ")
                (prev_ans "")
                (state diag))
    else
    (modify  ?F  (cur_quest  "Принтер  печатает  с  большой
задержкой?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
zaderzh ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-zaderzh none))
    (halt))
```

```
(defrule r_zaderzh
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Подобный дефект вызван отка-
зом интерфейсной платы. Для устранения неисправности
система предлагает заменить её. Если не все проблемы
устранены, попробуйте выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Принтер печатает с большой
задержкой?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
zaderzh ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-zaderzh none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_kachestvo_laser
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh no) (shum no) (kachestvo none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "на изображе-
нии вертикальные белые полосы" "неравномерная контраст-
ность изображения" "на обратной стороне отпечатка видны
посторонние изображения" "не закрепляется часть изобра-
жения" "нормальное" "вариант отсутствует в списке")
                (sys_ans bled vert_pol contr obrat zakrep
norm drugoi_otv)
```

```
                (user_eval   "send   [cur_printer]   put-
kachestvo ")
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "Слышен ли повышенный шум при
работе принтера?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-shum ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-shum none))
    (halt))

(defrule r_shum
    (object  (is-a  PRINTER_LASER)  (name  [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh no) (shum yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Дефект вызван тем, что за-
грязнились шестерни главного привода принтера. Для уст-
ранения неисправности система предлагает снять редуктор,
очистить все шестерни от пыли и тонера, а затем смазать
их. Если не все проблемы устранены, попробуйте выполнить
диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Слышен ли повышенный шум при
работе принтера?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-shum ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-shum none))
    (halt))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule r_bled_laser
    (object  (is-a  PRINTER_LASER)  (name  [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh no) (shum no) (kachestvo bled))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Бледная печать появляется в
результате выхода из строя магнитного вала картриджа.
Для устранения неисправности система предлагает заменить
магнитный вал. Если не все проблемы устранены, попробуй-
те выполнить диагностику заново.")
               (ans)
               (sys_ans)
               (user_eval "")
               (prev_ans "")
               (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
               (ans "бледное изображение" "на изображе-
нии вертикальные белые полосы" "неравномерная контраст-
ность изображения" "на обратной стороне отпечатка видны
посторонние изображения" "не закрепляется часть изобра-
жения" "нормальное" "вариант отсутствует в списке")
               (sys_ans bled vert_pol contr obrat zakrep
norm drugoi_otv)
               (user_eval   "send   [cur_printer]   put-
kachestvo ")
               (prev_q no)
               (prev_ans "бледное изображение")
               (state diag))
    (send [cur_printer] put-kachestvo none))
    (halt))

(defrule r_vert_pol_laser
    (object  (is-a  PRINTER_LASER)  (name  [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh no) (shum no) (kachestvo vert_pol))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Причина дефекта связана с за-
грязнением оптической системы лазер-сканера. Для устра-
нения неисправности система предлагает очистить её. Если
```

```
не все проблемы устранены, попробуйте выполнить диагно-
стику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "на изображе-
нии вертикальные белые полосы" "неравномерная контраст-
ность изображения" "на обратной стороне отпечатка видны
посторонние изображения" "не закрепляется часть изобра-
жения" "нормальное" "вариант отсутствует в списке")
                (sys_ans bled vert_pol contr obrat zakrep
norm drugoi_otv)
                (user_eval   "send   [cur_printer]   put-
kachestvo ")
                (prev_q no)
                (prev_ans  "на изображении вертикальные
белые полосы")
                (state diag))
    (send [cur_printer] put-kachestvo none))
    (halt))

(defrule r_contr_laser
    (object  (is-a  PRINTER_LASER)  (name  [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh no) (shum no) (kachestvo contr))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Подобный дефект вызван тем,
что сильно загрязнился вал переноса изображения. Для
устранения неисправности система предлагает очистить вал
от пыли и тонера. Если не все проблемы устранены, попро-
буйте выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "на изображе-
нии вертикальные белые полосы" "неравномерная контраст-
```

```
ность изображения" "на обратной стороне отпечатка видны посторонние изображения" "не закрепляется часть изображения" "нормальное" "вариант отсутствует в списке")
                (sys_ans bled vert_pol contr obrat zakrep norm drugoi_otv)
                (user_eval "send [cur_printer] put-kachestvo ")
                (prev_q no)
                (prev_ans "неравномерная контрастность изображения")
                (state diag))
    (send [cur_printer] put-kachestvo none))
    (halt))

(defrule r_obrat_laser
    (object (is-a PRINTER_LASER) (name [cur_printer]) (power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod no) (zaderzh no) (shum no) (kachestvo obrat))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Дефект вызван износом краёв силиконового вала. К нему прилипает тонер, который переносится на обратную сторону бумаги. Для устранения неисправности система предлагает заменить силиконовый вал. Если не все проблемы устранены, попробуйте выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "на изображении вертикальные белые полосы" "неравномерная контрастность изображения" "на обратной стороне отпечатка видны посторонние изображения" "не закрепляется часть изображения" "нормальное" "вариант отсутствует в списке")
                (sys_ans bled vert_pol contr obrat zakrep norm drugoi_otv)
                (user_eval "send [cur_printer] put-kachestvo ")
                (prev_q no)
```

```
                (prev_ans "на обратной стороне отпечатка
видны посторонние изображения")
                (state diag))
    (send [cur_printer] put-kachestvo none))
    (halt))

(defrule r_zakrep_laser
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh no) (shum no) (kachestvo zakrep))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Подобный дефект возникает из-
за разрыва термоплёнки. Причин может быть много: попада-
ние вместе с бумагой скрепок, высыхание или отсутствие
смазки в месте контакта термоплёнки и нагревательного
элемента, неравномерный прижим роллера к резиновому ва-
лу, брак самой термоплёнки, повреждение термопленки ост-
рыми предметами при попытке извлечения застрявшей бумаги
и т.д. Для устранения неисправности система предлагает
заменить термоплёнку. Если не все проблемы устранены,
попробуйте выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "на изображе-
нии вертикальные белые полосы" "неравномерная контраст-
ность изображения" "на обратной стороне отпечатка видны
посторонние изображения" "не закрепляется часть изобра-
жения" "нормальное" "вариант отсутствует в списке")
                (sys_ans bled vert_pol contr obrat zakrep
norm drugoi_otv)
                (user_eval   "send   [cur_printer]   put-
kachestvo ")
                (prev_q no)
                (prev_ans "не закрепляется часть изобра-
жения")
                (state diag))
    (send [cur_printer] put-kachestvo none))
    (halt))
```

```
(defrule r_norm_laser
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh no) (shum no) (kachestvo norm))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Данная система не выявила не-
исправность в принтере.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "на изображе-
нии вертикальные белые полосы" "неравномерная контраст-
ность изображения" "на обратной стороне отпечатка видны
посторонние изображения" "не закрепляется часть изобра-
жения" "нормальное" "вариант отсутствует в списке")
                (sys_ans bled vert_pol contr obrat zakrep
norm drugoi_otv)
                (user_eval   "send   [cur_printer]   put-
kachestvo ")
                (prev_q no)
                (prev_ans "нормальное")
                (state diag))
    (send [cur_printer] put-kachestvo none))
    (halt))

(defrule r_drugoi_otv_laser
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc yes) (zastr_cart no) (zastr_vyhod
no) (zaderzh no) (shum no) (kachestvo drugoi_otv))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Данная система не может по-
мочь.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
```

```
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
               (ans "бледное изображение" "на изображе-
нии вертикальные белые полосы" "неравномерная контраст-
ность изображения" "на обратной стороне отпечатка видны
посторонние изображения" "не закрепляется часть изобра-
жения" "нормальное" "вариант отсутствует в списке")
               (sys_ans bled vert_pol contr obrat zakrep
norm drugoi_otv)
               (user_eval   "send   [cur_printer]   put-
kachestvo ")
               (prev_q no)
               (prev_ans "вариант отсутствует в списке")
               (state diag))
    (send [cur_printer] put-kachestvo none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_install_skrip
    (object  (is-a  PRINTER_LASER)  (name  [cur_printer])
(power yes) (print_doc no) (install_skrip none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "При инсталляции принтера слы-
шен скрип слева?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval   "send   [cur_printer]   put-
install_skrip ")
               (prev_ans "")
               (state diag))
    else
    (modify ?F (cur_quest "Выполняется ли печать доку-
мента?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval   "send   [cur_printer]   put-
print_doc ")
               (prev_q no)
               (prev_ans "Нет")
               (state diag))
    (send [cur_printer] put-print_doc none))
    (halt))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_opred_cart_laser
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc no) (install_skrip no)
(opred_cart none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Принтер определяет наличие
картриджа?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval "send [cur_printer] put-
opred_cart ")
               (prev_ans "")
               (state diag))
    else
    (modify ?F (cur_quest "При инсталляции принтера слы-
шен скрип слева?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval "send [cur_printer] put-
install_skrip ")
               (prev_q no)
               (prev_ans "Нет")
               (state diag))
    (send [cur_printer] put-install_skrip none))
    (halt))

(defrule r_install_skrip
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc no) (install_skrip yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Лопнула шестерня основного
привода. Для устранения неисправности система предлагает
заменить шестерню, удалить старую смазку из привода и
нанести новую. Если не все проблемы устранены, попробуй-
те выполнить диагностику заново.")
               (ans)
               (sys_ans)
               (user_eval "")
               (prev_ans "")
               (state recomend))
```

```
    else
    (modify ?F (cur_quest "При инсталляции принтера слы-
шен скрип слева?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
install_skrip ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-install_skrip none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_opred_bum_laser
    (object  (is-a  PRINTER_LASER)  (name  [cur_printer])
(power   yes)   (print_doc   no)   (install_skrip   no)
(opred_cart yes) (opred_bum none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify  ?F  (cur_quest  "Принтер  определяет  наличие
бумаги в лотке?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
opred_bum ")
                (prev_ans "")
                (state diag))
    else
    (modify  ?F  (cur_quest  "Принтер  определяет  наличие
картриджа?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
opred_cart ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-opred_cart none))
    (halt))

(defrule r_opred_cart_laser
    (object  (is-a  PRINTER_LASER)  (name  [cur_printer])
(power   yes)   (print_doc   no)   (install_skrip   no)
(opred_cart no))
```

```
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "После закрытия верхней крышки
принтера к картриджу с помощью тяги подводится приводная
шестерня. В её центре есть подпружиненный штифт-контакт,
который подключает к общему проводу вал фоторецептора.
При попадании на штифт тонера цепь размыкается. Для
устранения неисправности система предлагает отвернуть
винт планки прижима штифта и очистить направляющее
отверстие, в котором ходит штифт контакт. Если не все
проблемы устранены, попробуйте выполнить диагностику за-
ново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Принтер определяет наличие
картриджа?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
opred_cart ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-opred_cart none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_zahvat_bum
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power   yes)   (print_doc   no)   (install_skrip   no)
(opred_cart yes) (opred_bum yes) (zahvat_bum none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Принтер захватывает бумагу из
приёмного лотка?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
zahvat_bum ")
```

```
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "Принтер определяет наличие
бумаги в лотке?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-
opred_bum ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-opred_bum none))
    (halt))

(defrule r_opred_bum_laser
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc no) (install_skrip no)
(opred_cart yes) (opred_bum no))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Засорился оптический датчик
или застрял его активатор. Для устранения неисправности
система предлагает очистить оптический датчик от загряз-
нений и проверить возврат в исходное положение его акти-
ватора. Если не все проблемы устранены, попробуйте вы-
полнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Принтер определяет наличие
бумаги в лотке?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-
opred_bum ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-opred_bum none))
    (halt))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule r_zahvat_bum_yes
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc no) (install_skrip no)
(opred_cart yes) (opred_bum yes) (zahvat_bum yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Данная система не может по-
мочь.")
               (ans)
               (sys_ans)
               (user_eval "")
               (prev_ans "")
               (state recomend))
    else
    (modify ?F (cur_quest "Принтер захватывает бумагу из
приёмного лотка?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval "send [cur_printer] put-
zahvat_bum ")
               (prev_q no)
               (prev_ans "Да")
               (state diag))
    (send [cur_printer] put-zahvat_bum none))
    (halt))

(defrule r_zahvat_bum_no
    (object (is-a PRINTER_LASER) (name [cur_printer])
(power yes) (print_doc no) (install_skrip no)
(opred_cart yes) (opred_bum yes) (zahvat_bum no))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Неисправен или сильно загряз-
нён подающий ролик принтера. Для устранения неисправно-
сти система предлагает произвести восстановление свойств
ролика жидкостью для профилактики резиновых поверхностей
или заменить его. Если не все проблемы устранены, попро-
буйте выполнить диагностику заново.")
               (ans)
               (sys_ans)
               (user_eval "")
               (prev_ans "")
               (state recomend))
```

```
    else
    (modify ?F (cur_quest "Принтер захватывает бумагу из
приёмного лотка?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval   "send   [cur_printer]   put-
zahvat_bum ")
               (prev_q no)
               (prev_ans "Нет")
               (state diag))
    (send [cur_printer] put-zahvat_bum none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Вопросы для диагностики неисправности струйного принте-
ра;;;;;;;
(defrule q_otkl_val
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Принтер делает попытку про-
вернуть основной вал и выключается?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval   "send   [cur_printer]   put-
otkl_val ")
               (prev_ans "")
               (state diag))
    else
    (modify ?F (cur_quest "Включается ли принтер?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval "send [cur_printer] put-power ")
               (prev_q no)
               (prev_ans "Да")
               (state diag))
    (send [cur_printer] put-power none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_print_doc_ink
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc none))
    ?F<-(UI-state (prev_q ?pr))
```

```
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Выполняется ли печать документа?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-print_doc ")
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "Принтер делает попытку провернуть основной вал и выключается?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-otkl_val ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-otkl_val none))
    (halt))

(defrule r_otkl_val
    (object  (is-a  PRINTER_INK)  (name  [cur_printer]) (power yes) (otkl_val yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Этот дефект связан с перегрузкой главного привода. Для устранения неисправности система предлагает вначале осмотреть визуально состояние элементов механизма подачи и транспорта бумаги, затем провернуть шестерню двигателя. Если шестерня вращается туго, разобрать механизм и проверить втулки вращения валов. Обычно этой процедуры достаточно для восстановления работоспособности принтера. Но если механизм исправен, а принтер отключается при отсоединённом двигателе, то неисправен сам двигатель. В этом случае его необходимо заменить. Если не все проблемы устранены, попробуйте выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
```

```
    else
    (modify ?F (cur_quest "Принтер делает попытку про-
вернуть основной вал и выключается?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
otkl_val ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-otkl_val none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_skrip
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Слышен ли скрип при движении
каретки?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-skrip ")
                (prev_ans  "")
                (state diag))
    else
    (modify ?F (cur_quest "Выполняется ли печать доку-
мента?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
print_doc ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-print_doc none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_bum_perekos_ink
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power  yes)  (otkl_val  no)  (print_doc  yes)  (skrip  no)
(bum_perekos none))
    ?F<-(UI-state (prev_q ?pr))
```

```
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Лист бумаги подаётся с пере-
косом?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
bum_perekos ")
                (prev_ans  "")
                (state diag))
    else
    (modify ?F (cur_quest "Слышен ли скрип при движении
каретки?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-skrip ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-skrip none))
    (halt))

(defrule r_skrip
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Высохла смазка на направляю-
щей оси каретки. Для устранения неисправности система
предлагает нанести на направляющую ось каретки смазку,
предназначенную для пластмассовых деталей. Если не все
проблемы устранены, попробуйте выполнить диагностику за-
ново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Слышен ли скрип при движении
каретки?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-skrip ")
```

```
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-skrip none))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_nesk_listov
    (object (is-a PRINTER_INK) (name [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
(bum_perekos no) (nesk_listov none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Принтер одновременно захваты-
вает несколько листов?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-
nesk_listov ")
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "Лист бумаги подаётся с пере-
косом?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-
bum_perekos ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-bum_perekos))
    (halt))

(defrule r_bum_perekos_ink
    (object (is-a PRINTER_INK) (name [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
(bum_perekos yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Для устранения неисправности
система предлагает разобрать весь механизм и очистить
все валы и поверхности, по которым движется бумага. Если
```

```
не все проблемы устранены, попробуйте выполнить диагно-
стику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Лист бумаги подаётся с пере-
косом?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
bum_perekos ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-bum_perekos))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_zastr_raboch
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
(bum_perekos no) (nesk_listov no) (zastr_raboch none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Застревает ли лист бумаги в
рабочей полости принтера?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
zastr_raboch ")
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "Принтер одновременно захваты-
вает несколько листов?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
nesk_listov ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
```

```
    (send [cur_printer] put-nesk_listov))
    (halt))

(defrule r_nesk_listov
    (object (is-a PRINTER_INK) (name [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
(bum_perekos no) (nesk_listov yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Загрязнилась или стерлась
тормозная площадка. Для устранения неисправности система
предлагает очистить её или заменить. Если не все пробле-
мы устранены, попробуйте выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Принтер одновременно захваты-
вает несколько листов?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval "send [cur_printer] put-
nesk_listov ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-nesk_listov))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_kachestvo_ink
    (object (is-a PRINTER_INK) (name [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
(bum_perekos no) (nesk_listov no) (zastr_raboch no) (ka-
chestvo none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "изображение
смазано" "на изображении дополнительные горизонтальные
```

```
полосы" "на изображении \"наезжает\" строка на строку"
"нормальное" "вариант отсутствует в списке")
               (sys_ans bled smaz horiz_pol str_na_str
norm drugoi_otv)
               (user_eval "send [cur_printer] put-
kachestvo ")
               (prev_ans "")
               (state diag))
    else
    (modify ?F (cur_quest "Застревает ли лист бумаги в
рабочей полости принтера?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval "send [cur_printer] put-
zastr_raboch ")
               (prev_q no)
               (prev_ans "Нет")
               (state diag))
    (send [cur_printer] put-zastr_raboch))
    (halt))

(defrule r_zastr_raboch
    (object (is-a PRINTER_INK) (name [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
(bum_perekos no) (nesk_listov no) (zastr_raboch yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Подобная неисправность возни-
кает из-за некорректного вытаскивания замятой бумаги.
Для устранения неисправности система предлагает визуаль-
но осмотреть механизм. Если обнаружится отсутствие одно-
го или нескольких металлических роликов прижима к рези-
новым роликам основного вала и отломанные ушки, в кото-
рых вращается вал, то необходимо заменить всю пружину.
Если не все проблемы устранены, попробуйте выполнить ди-
агностику заново.")
               (ans)
               (sys_ans)
               (user_eval "")
               (prev_ans "")
               (state recomend))
    else
    (modify ?F (cur_quest "Застревает ли лист бумаги в
рабочей полости принтера?")
               (ans "Нет" "Да")
```

```
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
zastr_raboch ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-zastr_raboch))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule r_bled_ink
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
(bum_perekos no) (nesk_listov no) (zastr_raboch no) (ka-
chestvo bled))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Картридж израсходован. Для
устранения неисправности система предлагает заменить
его. Если не все проблемы устранены, попробуйте выпол-
нить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "изображение
смазано" "на изображении дополнительные горизонтальные
полосы" "на изображении \"наезжает\" строка на строку"
"нормальное" "вариант отсутствует в списке")
                (sys_ans bled smaz horiz_pol str_na_str
norm drugoi_otv)
                (user_eval   "send   [cur_printer]   put-
kachestvo ")
                (prev_q no)
                (prev_ans "бледное изображение")
                (state diag))
    (send [cur_printer] put-kachestvo))
    (halt))

(defrule r_smaz_ink
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
```

```
(bum_perekos no) (nesk_listov no) (zastr_raboch no) (ka-
chestvo smaz))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Сильно загрязнились резиновые
ножи очистки. Для устранения неисправности система пред-
лагает промыть их. Если не все проблемы устранены, по-
пробуйте выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "изображение
смазано" "на изображении дополнительные горизонтальные
полосы" "на изображении \"наезжает\" строка на строку"
"нормальное" "вариант отсутствует в списке")
                (sys_ans bled smaz horiz_pol str_na_str
norm drugoi_otv)
                (user_eval   "send   [cur_printer]   put-
kachestvo ")
                (prev_q no)
                (prev_ans "изображение смазано")
                (state diag))
    (send [cur_printer] put-kachestvo))
    (halt))

(defrule r_horiz_pol_ink
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
(bum_perekos no) (nesk_listov no) (zastr_raboch no) (ka-
chestvo horiz_pol))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Для устранения неисправности
система предлагает снять картриджи и проверить, есть ли
на дюзах признаки наличия чернил. Если они есть, необхо-
димо проверить и очистить ракеля очистки. Также влажным
материалом очистить поверхность дюз картриджей. Если не
все проблемы устранены, попробуйте выполнить диагностику
заново.")
```

```
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "изображение
смазано" "на изображении дополнительные горизонтальные
полосы" "на изображении \"наезжает\" строка на строку"
"нормальное" "вариант отсутствует в списке")
                (sys_ans bled smaz horiz_pol str_na_str
norm drugoi_otv)
                (user_eval   "send   [cur_printer]   put-
kachestvo ")
                (prev_q no)
                (prev_ans "на изображении дополнительные
горизонтальные полосы")
                (state diag))
    (send [cur_printer] put-kachestvo))
    (halt))

(defrule r_str_na_str_ink
    (object   (is-a   PRINTER_INK)   (name   [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
(bum_perekos no) (nesk_listov no) (zastr_raboch no) (ka-
chestvo str_na_str))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Загрязнился основной вал, по-
этому нет хорошего сцепления с листом бумаги. Для устра-
нения неисправности система предлагает очистить основной
вал. Если не все проблемы устранены, попробуйте выпол-
нить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "изображение
смазано" "на изображении дополнительные горизонтальные
полосы" "на изображении \"наезжает\" строка на строку"
"нормальное" "вариант отсутствует в списке")
```

```
                (sys_ans bled smaz horiz_pol str_na_str
norm drugoi_otv)
                (user_eval "send [cur_printer] put-
kachestvo ")
                (prev_q no)
                (prev_ans "на изображении \"наезжает\"
строка на строку")
                (state diag))
    (send [cur_printer] put-kachestvo))
    (halt))

(defrule r_norm_ink
    (object (is-a PRINTER_INK) (name [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
(bum_perekos no) (nesk_listov no) (zastr_raboch no) (ka-
chestvo norm))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Данная система не выявила не-
исправность в принтере.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
                (ans "бледное изображение" "изображение
смазано" "на изображении дополнительные горизонтальные
полосы" "на изображении \"наезжает\" строка на строку"
"нормальное" "вариант отсутствует в списке")
                (sys_ans bled smaz horiz_pol str_na_str
norm drugoi_otv)
                (user_eval "send [cur_printer] put-
kachestvo ")
                (prev_q no)
                (prev_ans "нормальное")
                (state diag))
    (send [cur_printer] put-kachestvo))
    (halt))

(defrule r_drugoi_otv_ink
    (object (is-a PRINTER_INK) (name [cur_printer])
(power yes) (otkl_val no) (print_doc yes) (skrip no)
```

```
(bum_perekos no) (nesk_listov no) (zastr_raboch no) (ka-
chestvo drugoi_otv))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Данная система не может по-
мочь.")
               (ans)
               (sys_ans)
               (user_eval "")
               (prev_ans "")
               (state recomend))
    else
    (modify ?F (cur_quest "Укажите качество отпечатка:")
               (ans "бледное изображение" "изображение
смазано" "на изображении дополнительные горизонтальные
полосы" "на изображении \"наезжает\" строка на строку"
"нормальное" "вариант отсутствует в списке")
               (sys_ans bled smaz horiz_pol str_na_str
norm drugoi_otv)
               (user_eval "send [cur_printer] put-
kachestvo ")
               (prev_q no)
               (prev_ans "вариант отсутствует в списке")
               (state diag))
    (send [cur_printer] put-kachestvo))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_caret_upor
    (object (is-a PRINTER_INK) (name [cur_printer])
(power yes) (otkl_val no) (print_doc no) (caret_upor
none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "После включения принтера ка-
ретка перемещается до упора в крайнее положение?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval "send [cur_printer] put-
caret_upor ")
               (prev_ans "")
               (state diag))
    else
```

```
    (modify ?F (cur_quest "Выполняется ли печать документа?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-print_doc ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-print_doc))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule q_coord_lenta
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])  (power  yes)  (otkl_val  no)  (print_doc  no)  (caret_upor yes) (coord_lenta none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify  ?F  (cur_quest  "Координатная  лента  на месте?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-coord_lenta ")
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "После включения принтера каретка перемещается до упора в крайнее положение?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-caret_upor ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-caret_upor))
    (halt))

(defrule q_opred_cart_ink
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])  (power  yes)  (otkl_val  no)  (print_doc  no)  (caret_upor  no)  (opred_cart none))
    ?F<-(UI-state (prev_q ?pr))
```

```
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Принтер определяет наличие
картриджей?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
opred_cart ")
                (prev_ans "")
                (state diag))
    else
    (modify ?F (cur_quest "После включения принтера ка-
ретка перемещается до упора в крайнее положение?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
caret_upor ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-caret_upor))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule r_coord_lenta_yes
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power  yes)  (otkl_val  no)  (print_doc  no)  (caret_upor
yes) (coord_lenta yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Загрязнился оптический датчик
определения положения каретки. Для устранения неисправ-
ности система предлагает промыть оптический датчик спир-
том. Если не все проблемы устранены, попробуйте выпол-
нить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify   ?F   (cur_quest   "Координатная   лента   на
месте?")
                (ans "Нет" "Да")
```

```
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
coord_lenta ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-coord_lenta))
    (halt))

(defrule r_coord_lenta_no
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power  yes)  (otkl_val  no)  (print_doc  no)  (caret_upor
yes) (coord_lenta no))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Для устранения неисправности
система предлагает возвратить координатную ленту в нор-
мальное положение. Если не все проблемы устранены, по-
пробуйте выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify  ?F  (cur_quest  "Координатная  лента  на
месте?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
coord_lenta ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-coord_lenta))
    (halt))

(defrule q_pod_bum
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc no) (caret_upor no)
(opred_cart yes) (pod_bum none))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
```

```
    (modify ?F (cur_quest "Осуществляется ли подача бу-
маги?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval   "send   [cur_printer]   put-
pod_bum ")
               (prev_ans "")
               (state diag))
    else
    (modify ?F (cur_quest "Принтер определяет наличие
картриджей?")
               (ans "Нет" "Да")
               (sys_ans no yes)
               (user_eval   "send   [cur_printer]   put-
opred_cart ")
               (prev_q no)
               (prev_ans "Да")
               (state diag))
    (send [cur_printer] put-opred_cart))
    (halt))

(defrule r_opred_cart_ink
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc no) (caret_upor no)
(opred_cart no))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Для устранения неисправности
система предлагает вначале убедиться, что картриджи дей-
ствительно исправны и их контактные площадки чистые, за-
тем вынуть их и очистить контактные группы в корзине ка-
ретки. Если это не помогает, то необходимо заменить пла-
ту электроники каретки. Если не все проблемы устранены,
попробуйте выполнить диагностику заново.")
               (ans)
               (sys_ans)
               (user_eval "")
               (prev_ans "")
               (state recomend))
    else
    (modify ?F (cur_quest "Принтер определяет наличие
картриджей?")
               (ans "Нет" "Да")
               (sys_ans no yes)
```

```
                (user_eval   "send   [cur_printer]   put-
opred_cart ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-opred_cart))
    (halt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule r_pod_bum_yes
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc no) (caret_upor no)
(opred_cart yes) (pod_bum yes))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Данная система не может Вам
помочь.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Осуществляется ли подача бу-
маги?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-
pod_bum ")
                (prev_q no)
                (prev_ans "Да")
                (state diag))
    (send [cur_printer] put-pod_bum))
    (halt))

(defrule r_pod_bum_no
    (object  (is-a  PRINTER_INK)  (name  [cur_printer])
(power yes) (otkl_val no) (print_doc no) (caret_upor no)
(opred_cart yes) (pod_bum no))
    ?F<-(UI-state (prev_q ?pr))
    =>
    (if (eq ?pr no)
    then
    (modify ?F (cur_quest "Для устранения неисправности
система предлагает протереть ролик подачи бумаги тканью,
```

```
смоченной жидкостью для восстановления резиновых роликов. Если это не помогает, необходимо заменить ролик. Если не все проблемы устранены, попробуйте выполнить диагностику заново.")
                (ans)
                (sys_ans)
                (user_eval "")
                (prev_ans "")
                (state recomend))
    else
    (modify ?F (cur_quest "Осуществляется ли подача бумаги?")
                (ans "Нет" "Да")
                (sys_ans no yes)
                (user_eval   "send   [cur_printer]   put-pod_bum ")
                (prev_q no)
                (prev_ans "Нет")
                (state diag))
    (send [cur_printer] put-pod_bum))
    (halt))
```

Файл kurs.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import sys
import clips
import wx
import wx.lib.buttons
import wx.html

# Класс главного окна
class MainFrame(wx.Frame):
    def __init__(self):

        ###################################
        #Создание графического интерфейса#
        ###################################
        #Создание окна приложения
        wx.Frame.__init__(self, parent = None, id = -1,
title = u'Диагностика неисправности принтера',
            size=(550, 450), style=wx.CAPTION |
wx.SYSTEM_MENU | wx.CLIP_CHILDREN | wx.CLOSE_BOX)
        self.Center(wx.BOTH)

self.SetIcon(wx.Icon(u'./img/printer.ico',wx.BITMAP_TYPE
_ICO))

        #Добавление поля для вывода приветствия,
вопросов и рекомендаций
        self.questText = wx.html.HtmlWindow(id=-1,
parent=self, pos=wx.Point(10, 16),
            size=wx.Size(530, 119),
style=wx.html.HW_SCROLLBAR_NEVER)

        #Добавление панели для размещения на ней
вариантов ответов
        self.varPanel = wx.Panel(id=-1, parent=self,
pos=wx.Point(42, 134), size=wx.Size(520, 214),
            style=wx.TAB_TRAVERSAL)

        #Добавление панели для размещения на ней кнопок
управления
        self.btnPanel = wx.Panel(id=-1, parent=self,
pos=wx.Point(30, 360), size=wx.Size(484, 88),
            style=wx.TAB_TRAVERSAL)

        #Добавление кнопки "Назад" на форму
```

```
        self.backBtn =
wx.lib.buttons.GenBitmapTextButton(bitmap=wx.Bitmap(u'./
img/back.png',
            wx.BITMAP_TYPE_PNG), label=u'Назад',
parent=self.btnPanel, pos=wx.Point(21, 26),
            size=wx.Size(128, 30), style=0)
        self.backBtn.Bind(wx.EVT_BUTTON,
self.handleEvent)

        #Добавление кнопки "Далее" на форму
        self.nextBtn =
wx.lib.buttons.GenBitmapTextButton(bitmap=wx.Bitmap(u'./
img/next.png',
            wx.BITMAP_TYPE_PNG), label=u'Далее',
parent=self.btnPanel, pos=wx.Point(336, 26),
            size=wx.Size(128, 30), style=0)
        self.nextBtn.Bind(wx.EVT_BUTTON,
self.handleEvent)

        #Добавление кнопки "Заново" на форму
        self.renewBtn =
wx.lib.buttons.GenBitmapTextButton(bitmap=wx.Bitmap(u'./
img/renew.png',
            wx.BITMAP_TYPE_PNG), label=u'Заново',
parent=self.btnPanel, pos=wx.Point(179, 10),
            size=wx.Size(128, 30), style=0)
        self.renewBtn.Bind(wx.EVT_BUTTON,
self.handleEvent)

        #Добавление кнопки "Выход" на форму
        self.exitBtn =
wx.lib.buttons.GenBitmapTextButton(bitmap=wx.Bitmap(u'./
img/exit.png',
            wx.BITMAP_TYPE_PNG), label=u'Выход',
parent=self.btnPanel, pos=wx.Point(178, 45),
            size=wx.Size(128, 30), style=0)
        self.exitBtn.Bind(wx.EVT_BUTTON, self.OnQuit)

        #Добавление разделительной линии
        self.sashWindow1 = wx.SashWindow(parent=self,
pos=wx.Point(50, 130),
            size=wx.Size(450, 3), style=wx.CLIP_CHILDREN
| wx.SW_3D)

        #Добавление разделительной линии
        self.sashWindow2 = wx.SashWindow(parent=self,
pos=wx.Point(50, 360),
```

```
            size=wx.Size(450, 3), style=wx.CLIP_CHILDREN
| wx.SW_3D)

        ###################################
        ###################################

        #Загрузка и запуск экспертной системы
        clips.Load("kurs.clp")
        clips.Reset()
        clips.Run()

        self.nextState()

    #Обновление графического интерфейса в соответствии с
текущим состоянием экспертной системы
    def nextState(self):
        factlist = clips.FactList()

        fState=factlist[1].Slots["state"]
        fPrevAns=factlist[1].Slots["prev_ans"]
        fAns=factlist[1].Slots["ans"]
        fCurQuest=factlist[1].Slots["cur_quest"]

        #Переопределение параметров кнопок, панелей и
разделителей в соответствии с текущим состоянием
экспертной системы
        if fState == "start":
            self.varPanel.Hide()
            self.sashWindow1.Hide()
            self.backBtn.Enable(False)
            self.nextBtn.Enable(True)
            self.renewBtn.Enable(False)
        elif fState == "diag":
            self.questText.SetSize(wx.Size(530, 119))
            self.varPanel.Show()
            self.sashWindow1.Show()
            self.backBtn.Enable(True)
            self.nextBtn.Enable(True)
            self.renewBtn.Enable(True)
        else:
            self.questText.SetSize(wx.Size(530, 300))
            self.varPanel.Hide()
            self.sashWindow1.Hide()
            self.backBtn.Enable(True)
            self.nextBtn.Enable(False)
            self.renewBtn.Enable(True)
```

```
        #Переопределение вариантов ответов
        self.varPanel.DestroyChildren()
        sizer = wx.BoxSizer(wx.VERTICAL)
        self.varPanel._buttons = []
        for answer in fAns:
            r = wx.RadioButton(self.varPanel, -1,
answer)
            if answer == fPrevAns:
                r.SetValue(True)
            sizer.Add(r, flag = wx.ALL, border = 5)
            self.varPanel._buttons.append(r)
        self.varPanel.SetSizer(sizer)
        self.varPanel.Fit()

        #Переопределение вопроса
        self.questText.SetPage("<body
bgcolor=#F0EBE2><center><h5>%s</h5></center></body>" %
fCurQuest)

        self.Layout()
        self.Refresh()

    #Обработка нажатий кнопок в форме
    def handleEvent(self, event):
        factlist = clips.FactList()

        fState=factlist[1].Slots["state"]
        fUserEval=factlist[1].Slots["user_eval"]
        fSysAns=factlist[1].Slots["sys_ans"]
        fAns=factlist[1].Slots["ans"]

        #Обработка нажатия кнопоки "Далее"
        if event.GetEventObject().GetLabel() ==
u'Далее':
            if fState == "diag":
                j=0;
                for chkAns in self.varPanel._buttons:
                    if chkAns.GetValue():
                        break
                    j=j+1

                if j>len(fAns):
                    return

                clips.Eval("(%s%s)" % (fUserEval,
fSysAns[j]))
```

```
        #Обработка нажатия кнопоки "Заново"
        elif event.GetEventObject().GetLabel() ==
u'Заново':
            clips.Reset()

        #Обработка нажатия кнопоки "Назад"
        elif event.GetEventObject().GetLabel() ==
u'Назад':
            clips.Eval("(prev_q_yes %s)" %
(factlist[1].Index))

        clips.Run()

        self.nextState()

    #Обработка нажатия кнопки "Выход"
    def OnQuit(self, event):
        self.Close() # завершение работы приложения

# Класс приложения
class PrintDiagApp(wx.App):
    def OnInit(self):
        # создание главного окна
        frame = MainFrame()
        # отображение главного окна
        frame.Show()
        return True

if __name__ == '__main__':
    app = PrintDiagApp()
    app.MainLoop()
```

Приложение Б

Файл job.clp

```
(defclass human
(is-a USER) ;; пользовательский класс
(role concrete)
(pattern-match reactive)
(slot education (type SYMBOL)(create-accessor  read-
write))
(slot sphere (type SYMBOL)(create-accessor  read-write))
(slot lang (type SYMBOL)(create-accessor  read-write))
(slot comp (type SYMBOL)(create-accessor  read-write))
(slot managament (type SYMBOL)(create-accessor  read-
write))
(slot experience (type SYMBOL)(create-accessor  read-
write))
(slot experiencenegotiations (type SYMBOL)(create-
accessor  read-write))
(slot communicating (type SYMBOL)(create-accessor  read-
write))
(slot organized (type SYMBOL)(create-accessor  read-
write))
(slot creative (type SYMBOL)(create-accessor  read-
write))
(slot paul (type SYMBOL)(create-accessor  read-write))
(slot age (type SYMBOL)(create-accessor  read-write))
(slot driverscertificate (type SYMBOL)(create-accessor
read-write))
(slot graph (type SYMBOL)(create-accessor  read-write) ) )

(definstances humans (PPK of human
(education 0)
(sphere 0)
(lang 0)
(comp 0)
(managament 0)
(experience 0)
(experiencenegotiations 0)
(communicating 0)
(organized 0)
(creative 0)
(paul 0)
(age 0)
(driverscertificate 0)
(graph 0) ) )
```

```
(make-instance PPK of human)

(deffunction check (?question ?answer)
(switch ?question
       (case 1 then
           (switch ?answer
              (case 1 then (send [PPK] put-education
higher))
              (case 2 then (send [PPK] put-education
average-professional))
              (case 3 then (send [PPK] put-education
average))
              (default 0) ))
       (case 2 then
           (switch ?answer
              (case 1 then (send [PPK] put-sphere
economic))
              (case 2 then (send [PPK] put-sphere
humanitarian))
              (case 3 then (send [PPK] put-sphere
technical))
              (case 4 then (send [PPK] put-sphere other))
              (default 0) ))
       (case 3 then
           (switch ?answer
              (case 1 then (send [PPK] put-lang in-
perfection))
              (case 2 then (send [PPK] put-lang talking))
              (case 3 then (send [PPK] put-lang reading))
              (case 4 then (send [PPK] put-lang no))
              (default 0) ))
       (case 4 then
           (switch ?answer
              (case 1 then (send [PPK] put-comp
professional))
              (case 2 then (send [PPK] put-comp user))
              (case 3 then (send [PPK] put-comp no))
              (default 0) ))
       (case 5 then
           (switch ?answer
              (case 1 then (send [PPK] put-managament yes))
              (case 2 then (send [PPK] put-managament no))
              (default 0) ))
       (case 6 then
           (switch ?answer
              (case 1 then (send [PPK] put-experience yes))
              (case 2 then (send [PPK] put-experience no))
              (default 0) ))
```

```
       (case 7 then
           (switch ?answer
              (case 1 then (send [PPK] put-communicating
yes))
              (case 2 then (send [PPK] put-communicating
no) )
              (default 0) ))
       (case 8 then
           (switch ?answer
              (case 1 then (send [PPK] put-organized yes))
              (case 2 then (send [PPK] put-organized no))
              (default 0) ) )
       (case 9 then
           (switch ?answer
              (case 1 then (send [PPK] put-creative yes))
              (case 2 then (send [PPK] put-creative no))
              (default 0) ))
       (case 10 then
           (switch ?answer
              (case 1 then (send [PPK] put-paul male))
              (case 2 then (send [PPK] put-paul female))
              (default 0) ))
       (case 11 then
           (send [PPK] put-age ?answer) )
       (case 12 then
           (switch ?answer
              (case 1 then (send [PPK] put-
driverscertificate yes))
              (case 2 then (send [PPK] put-
driverscertificate no))
              (default 0) ))
       (case 13 then
           (switch ?answer
              (case 1 then (send [PPK] put-graph full))
              (case 2 then (send [PPK] put-graph free))
              (case 3 then (send [PPK] put-graph
unnormalized))
              (case 4 then (send [PPK] put-graph short))
              (default 0) ))
       (case 14 then
           (switch ?answer
              (case 1 then (send [PPK] put-
experiencenegotiations yes))
              (case 2 then (send [PPK] put-
experiencenegotiations no))
              (default 0) )) ) )
```

```
;Правила для определения вакансии
; -- менеджер
(defrule rule-manager
      (declare (salience 50))
      (object (name [PPK]) (education higher))
      (object (name [PPK]) (sphere economic))
      (object (name [PPK]) (communicating yes))
      (object (name [PPK]) (organized yes))
      (object (name [PPK]) (lang in-perfection))
      (object (name [PPK]) (comp professional | user))
      (object (name [PPK]) (paul male))
      (object (name [PPK]) (creative yes))
      (object (name [PPK]) (managament yes))
      (object (name [PPK]) (experience yes))
      (object (name [PPK]) (experiencenegotiations yes))
      (object (name [PPK]) (graph full))
      =>
      (assert (job-title менеджер) ) )

      ; -- маркетолог
      (defrule rule-market
      (declare (salience 50))
      (object (name [PPK])      (education higher))
      (object (name [PPK])      (sphere economic))
      (object (name [PPK])      (communicating yes))
      (object (name [PPK])      (organized yes))
      (object (name [PPK])      (lang in-perfection))
      (object (name [PPK])      (comp professional |
user))
      (object (name [PPK])      (paul male))
      (test (< (send [PPK] get-age) 35))
      (object (name [PPK])      (creative yes))
      (object (name [PPK])      (experiencenegotiations
yes))(object (name [PPK]) (graph full))
      =>
      (assert (job-title маркетолог) ) )

      ; -- бухгалтер
      (defrule rule-accountant
      (declare (salience 50))
      (object (name [PPK])      (education higher))
      (object (name [PPK])      (sphere economic))
      (object (name [PPK])      (organized yes))
      (object (name [PPK])      (comp professional))
      (object (name [PPK])      (paul female))
      (object (name [PPK])      (creative yes))
      (object (name [PPK])      (graph unnormalized |
```

```
full))
       =>
       (assert (job-title бухгалтер) ) )

       ; -- специалист по связям с общественностью
       (defrule rule-public-relation
       (declare (salience 50))
       (object (name [PPK])      (education higher))
       (object (name [PPK])      (sphere humanitarian))
       (object (name [PPK])      (communicating
yes))(object (name [PPK]) (organized yes))
       (object (name [PPK])      (lang in-perfection))
       (object (name [PPK])      (comp professional |
user))
       (object (name [PPK])      (creative yes))
       (object (name [PPK])      (experiencenegotiations
yes))
       (object (name [PPK])      (graph full))
       =>
       (assert (job-title специалист по связям с
общественностью) ) )

       ; -- системный администратор
       (defrule rule-sys-admin
       (declare (salience 50))
       (object (name [PPK]) (education average | average-
professional))
       (object (name [PPK]) (sphere technical))
       (object (name [PPK]) (lang in-perfection | talking
| reading))
       (object (name [PPK]) (comp professional))
       (object (name [PPK]) (paul male))
       (object (name [PPK]) (graph full))
       =>
       (assert (job-title системный администратор) ) )

       ; -- разработчик приложений
       (defrule rule-developer
       (declare (salience 50))
       (object (name [PPK]) (education higher))
       (object (name [PPK]) (sphere technical))
       (object (name [PPK]) (lang in-perfection | talking
| reading))
       (object (name [PPK]) (comp professional))
 (object (name [PPK]) (graph full | short | free))
 =>
 (assert (job-title разработчик приложений) ) )
```

```
 ; -- web-программист
 (defrule rule-web
 (declare (salience 50))
 (job-title разработчик приложений)
 (object (name [PPK]) (creative yes))
 =>
 (assert (job-title web-программист) ) )

 ; -- водитель
 (defrule rule-driver
 (declare (salience 45))
 (object (name [PPK]) (paul male))
 (object (name [PPK]) (graph full | short))
 (object (name [PPK]) (driverscertificate yes))
 =>
 (assert (job-title водитель) ) )

 ; -- торговый представитель
 (defrule rule-representative
 (declare (salience 45))
 (object (name [PPK]) (communicating yes))
 (object (name [PPK]) (organized yes))
 (object (name [PPK]) (graph full | free))
 =>
 (assert (job-title торговый представитель) ) )

 ; -- секретарь
 (defrule rule-secretary
 (declare (salience 45))
 (object (name [PPK]) (education average-
professional))
 (object (name [PPK]) (communicating yes))
 (object (name [PPK]) (organized yes))
 (object (name [PPK]) (lang in-perfection))
 (object (name [PPK]) (comp professional))
 (object (name [PPK]) (paul female))
 (object (name [PPK]) (graph full))
 =>
 (assert (job-title секретарь) ) )

 ; -- специалист по связям с общественностью
 (defrule rule-public-relation
 (declare (salience 50))
 (object (name [PPK])     (education higher))
 (object (name [PPK])     (sphere humanitarian))
 (object (name [PPK])     (communicating yes))
 (object (name [PPK])     (organized yes))
```

```
 (object (name [PPK])     (lang in-perfection))
 (object (name [PPK])     (comp professional | user))
 (object (name [PPK])     (creative yes))
 (object (name [PPK])     (experiencenegotiations
yes))
 (object (name [PPK])     (graph full))
 =>
 (assert (job-title специалист по связям с
общественностью) ) )

 ; -- системный администратор
 (defrule rule-sys-admin
 (declare (salience 50))
 (object (name [PPK]) (education average | average-
professional))
 (object (name [PPK]) (sphere technical))
 (object (name [PPK]) (lang in-perfection | talking |
reading))
 (object (name [PPK]) (comp professional))
 (object (name [PPK]) (paul male))
 (object (name [PPK]) (graph full))
 =>
 (assert (job-title системный администратор) ) )

 ; -- предложение вакансий
 (defrule rule-list-vacancies
 (declare (salience 40))
 (job-title $?item)
 =>
 (bind ?str (implode$ ?item))
 (printout t crlf)
 (printout t "Вакансия: ")
 (format t "%s%n" ?str))

 ; -- нет вакансий
 (defrule rule-no-vacancies
 (declare (salience 40))
 (not(job-title ?))
 =>
 (assert (job-title К сожалению, в данный момент у
нас нет вакансий))
```

Файл index.php

```
<?php
    header('Content-Type: text/html;
charset=windows-1251');
    clips_clear();
    clips_batch('job.clp');
    clips_reset();
    if (isset($_POST['obraz'])) {
 clips_function_call('check',"1 $_POST[obraz]");
    }
    if (isset($_POST['sfera'])) {
 clips_function_call('check',"2 $_POST[sfera]");
    }
    if (isset($_POST['jazik'])) {
 clips_function_call('check',"3 $_POST[jazik]");
    }
    if (isset($_POST['comp'])) {
 clips_function_call('check',"4 $_POST[comp]");
    }
    if (isset($_POST['managament'])) {
 clips_function_call('check',"5 $_POST[managament]");
    }
    if (isset($_POST['experience'])) {
 clips_function_call('check',"6 $_POST[experience]");
    }
    if (isset($_POST['communicating'])) {
 clips_function_call('check',"7
$_POST[communicating]");
    }
    if (isset($_POST['organized'])) {
 clips_function_call('check',"8 $_POST[organized]");
    }
    if (isset($_POST['creative'])) {
 clips_function_call('check',"9 $_POST[creative]");
    }
    if (isset($_POST['paul'])) {
 clips_function_call('check',"10 $_POST[paul]");
    }
    if (isset($_POST['age'])) {
 clips_function_call('check',"11 $_POST[age]");
    }
    if (isset($_POST['driverscertificate'])) {
 clips_function_call('check',"12
$_POST[driverscertificate]");
    }
    if (isset($_POST['graph'])) {
 clips_function_call('check',"13 $_POST[graph]");
    }
```

```
     if (isset($_POST['experiencenegotiations'])) {
 clips_function_call('check',"14
$_POST[experiencenegotiations]");
     }
     if (isset($_POST['result'])) {
      $job_title = clips_get_fact_lits('job-title');
 }

 ?>
 <html>
       <head>
       <title>Экспертная система для выбора
вакансии</title>
       <style>
              .question {
                  display:none;
              }
              #submit {
                  display:none;
              }
              html, body {
                  margin:auto;
                  padding:10px;
              }
              #main {
                  margin:auto;
                  width:400px;
                  padding:5px;
                  text-align:left;
                  border:1px solid black;
              }
       </style>
       <script lanuage="javascript"
src="jquery.js"></script>
       <script lanuage="javascript">
              var currQ = 1;
              $(document).ready(function() {
                  $('#question1').show();
                  $('#next').click(function() {
                     if (currQ < 14)
                     {
                         $('#question' +
currQ).hide();
                         currQ++;
                         $('#question' +
currQ).show();
                     }else {
                         $('#submit').show();
                     }
```

```
                    return false;
                })
            })
        </script>
        </head>
 <body>
 <div id="main">
 <?php if (isset($job_title)) {
        echo "<div id='result'>Мы может предложить вам
следующие вакансии:";
        foreach ($job_title as $value) {
            echo '<br>'.$value;
        }
        echo "</div>";
      }else {?>
 <form action="index.php" method="POST">
 <div id="question1" class="question">
 Какое у Вас образование?<br />
 <input type="radio" name="obraz" value="1"
/>Высшее<br />
        <input type="radio" name="obraz" value="2"
/>Профессиональное<br />
 <input type="radio" name="obraz" value="3" />Среднее
 </div>
 <div id="question2" class="question">
 Ваше образование?<br />
 <input type="radio" name="sfera" value="1"
/>Экономическое<br />
        <input type="radio" name="sfera" value="2"
/>Гуманитарное<br />
        <input type="radio" name="sfera" value="3"
/>Техническое<br />
 <input type="radio" name="sfera" value="4" />Другое
        </div>
 <div id="question3" class="question">
 Степень владения иностранным языком?<br />
 <input type="radio" name="jazik" value="1" />В
совершенстве<br />
        <input type="radio" name="jazik" value="2"
/>Разговорный<br />
        <input type="radio" name="jazik" value="3"
/>Чтение<br />
 <input type="radio" name="jazik" value="4"
/>Не владею
        </div>
 <div id="question4" class="question">
```

```
 Владение компьютером в своей области?<br />
 <input type="radio" name="comp" value="1"
/>Профессионал<br />
       <input type="radio" name="comp" value="2"
/>Пользователь<br />
 <input type="radio" name="comp" value="3" />Не владею
       </div>
 <div id="question5" class="question">
 Вы умеете руководить людьми?<br />
 <input type="radio" name="managament" value="1"
/>Да<br />
       <input type="radio" name="managament"
value="2" />Нет
       </div>
 <div id="question6" class="question">
 У Вас есть опыт руководства людьми?<br />
 <input type="radio" name="experience" value="1"
/>Да<br />
       <input type="radio" name="experience"
value="2" />Нет
       </div>
 <div id="question7" class="question">
 Вы коммуникабельны?<br />
 <input type="radio" name="communicating" value="1"
/>Да<br />
       <input type="radio" name="communicating"
value="2" />Нет
       </div>
 <div id="question8" class="question">
 Вы организованы?<br />
 <input type="radio" name="organized" value="1"
/>Да<br />
       <input type="radio" name="organized" value="2"
/>Нет
       </div>
 <div id="question9" class="question">
 Вы креативны?<br />
 <input type="radio" name="creative" value="1"
/>Да<br />
       <input type="radio" name="creative" value="2"
/>Нет
       </div>
 <div id="question10" class="question">
 Ваш пол?<br />
 <input type="radio" name="paul" value="1"
/>Мужской<br />
       <input type="radio" name="paul" value="2"
```

```
/>Женский
      </div>
 <div id="question11" class="question">
 Ваш возраст?<br />
 <input type="input" name="age" />
      </div>
 <div id="question12" class="question">
 У Вас есть водительское удостоверение?<br />
 <input type="radio" name="driverscertificate"
value="1" />Да<br />
      <input type="radio" name="driverscertificate"
value="2" />Нет
      </div>
 <div id="question13" class="question">
 Какой график работы Вас устраивает?<br />
 <input type="radio" name="graph" value="1"
/>Полный рабочий день<br />
      <input type="radio" name="graph" value="2"
/>Гибкий<br />
      <input type="radio" name="graph" value="3"
/>Ненормированный<br />
 <input type="radio" name="graph" value="4"
/>Сокращённый
      </div>
 <div id="question14" class="question">
 У Вас есть опыт ведения переговоров?<br />
 <input type="radio" name="experiencenegotiations"
value="1" />Да<br />
      <input type="radio"
name="experiencenegotiations" value="2" />Нет
      </div>
      <input type="hidden" name="result" value="1"
/>
      <input id='submit' type='submit'
value='Вывести результат'>
 </form>
 <a id='next' href=''>Далее</a>
 <?php }?>
 </div>
 </body>
</html>
```

ОГЛАВЛЕНИЕ

Учебное издание

ГРОМОВ Юрий Юрьевич,
ИВАНОВА Ольга Геннадьевна,
АЛЕКСЕЕВ Владимир Витальевич,
БЕЛЯЕВ Максим Павлович,
ШВЕЦ Дмитрий Петрович,
ЕЛИСЕЕВ Алексей Игоревич

ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

Учебное пособие

Редактор Ю.В. С а л ы к и н а
Инженер по компьютерному макетированию М.Н. Р ы ж к о в а

Подписано в печать 15.03.2013
Формат 60 × 84/16. 14,18 усл. печ. л. Тираж 100 экз. Заказ № 110

Издательско-полиграфический центр ФГБОУ ВПО «ТГТУ»
392000, г. Тамбов, ул. Советская, д. 106, к. 14