Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

В.В. Фомин, В.А. Миклуш

ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ

Учебное пособие

РГГМУ
Санкт-Петербург
2013

УДК 002.52
ББК 32.811

Фомин В.В., Миклуш В.А. Интеллектуальные информационные системы: Учебное пособие. – СПб.: РГГМУ, 2013. – 150 с.

ISBN 978-5-86813-361-9

Рецензент: Л.В. Уткин, д-р техн. наук, проф., зав. каф. управления, автоматизации и системного анализа Санкт-Петербургского государственного лесотехнического университета им. С.М. Кирова

В учебном пособии изложены методологические основы интеллектуальных информационных систем: терминология и определения, основные концепции, подходы, принципы и методы, которые используются в системах искусственного интеллекта. Рассмотрены области применения искусственного интеллекта, классификация интеллектуальных информационных систем, формальные модели представления данных и знаний. Подробно раскрыта проблематика отдельных классов интеллектуальных систем: экспертных систем и систем интеллектуального анализа данных. Представлен инструментарий разработки интеллектуальных информационных систем: языки программирования и оболочки.

Предназначено для студентов, обучающихся по специальности 090302 «Информационная безопасность телекоммуникационных систем».

Fomin V.V., Miklush V.A. Intelligent information system. Textbook. – St. Petersburg: RSHU, 2013. – 150 p.

The manual sets out the methodological foundations of intelligent information systems: terminology and definitions, basic concepts, approaches, principles and techniques that are used in artificial intelligence systems. The scope of application of artificial intelligence, classification of intelligent information systems, formal models of data and knowledge. Disclosed in detail the problems of individual classes of intelligent systems: expert systems and data mining. Presented development tools of intelligent information systems: programming languages and shell.

ISBN 978-5-86813-361-9

Оглавление

Contents

Предисловие

По тематике интеллектуальных информационных систем и технологий опубликованы сотни учебников и монографий, огромное количество информации и учебного материала можно найти в интернете. При формировании данного пособия была проведена целенаправленная выборка материала лекционного характера из разных источников открытого доступа, а также ранних авторских изданий по тематике искусственного интеллекта и интеллектуальных информационных систем.

Процесс обучения специалистов интеллектуальным информационным системам напрямую зависит от целевой аудитории обучающихся. Данное пособие ориентировано на студентов старших курсов технических специальностей и служит не более чем ознакомлением с возможностями выделенных из огромного разнообразия двух больших классов интеллектуальных систем (экспертных систем и систем интеллектуального анализа), знание инструментария которых, по мнению авторов, востребовано на текущий момент на рынке труда.

Ориентируясь на специалистов-эксплуатационников информационных технологий, а не их разработчиков, авторы опускают специальные разделы математики, не затрагивают профессиональный процесс проектирования информационных систем, вводят некоторые упрощения и в качестве дополнений или комментария снабжают различными ссылками на информационные источники.

В первой главе пособия последовательно излагаются основные термины и определения, раскрываются проблематика, отличительные особенности и перспективы развития интеллектуальных систем и технологий. Материал акцентирован на пути эволюции информационных систем от представления и обработки данных до интеллектуальных систем и баз знаний. В конце главы представлены формальные модели описания и обработки данных и знаний. Глава является некоторой фундаментальной базой для дальнейшего изучения интеллектуальных систем и технологий в практике их применения.

Вторая глава полностью посвящена разделу экспертные системы. Экспертные системы – старейший класс интеллектуальных

систем, прочно вошедший в практику применения в различных сферах человеческой деятельности и нашедший широкое применение и распространение. За полвека человечеством накоплена масса описаний, практик, публикаций, учебных курсов и т.д., посвященная экспертным системам. Будучи развитым и ярчайшим представителем класса интеллектуальных систем, экспертные системы наиболее дидактически удобный материал для обучения студентов непрофильных специальностей.

В третьей главе представлен материал по новейшим технологиям интеллектуального анализа данных. В XXI в. главным источником рыночной силы становится интеллект, воплощенный в организационных структурах исследовательских и рыночных организациях, корпораций, создающих метатехнологии и удерживающих контроль за ними. В информационном обществе все больший вес приобретают высококвалифицированные специалисты – «золотые воротнички», владеющие метатехнологиями. Именно таким технологиям посвящена третья глава, раскрывающая и описывающая класс инструментальных средств распознавания – кластеризации и классификации, применение которых кардинально повышает эффективность анализа и качество принимаемых решений, повышает конкурентоспособность организации и остро востребовано на современном рынке образовательных и трудовых навыков специалистов различного профиля.

Четвертая глава знакомит обучающихся с миром формальных языков программирования и инструментальных оболочек, ориентированных на разработку интеллектуальных информационных систем и может послужить толчком к стремлению студента профилироваться в направлении специализации инженера по программному обеспечению и/или программированию.

Предлагаемые материалы ни в коем случае нельзя рассматривать как догму или уповать на полноту представления тех или иных парадигм, методов, теорий. Авторы предполагают, что данное пособие может служить только отправной точкой в области теории, разработки и эксплуатации интеллектуальных систем. Представленные материалы показывают огромное разнообразие существующего знания и должны стать движущей, направляющей силой для изучения сложного мира интеллектуальных информационных систем.

Введение

Без владения научным инструментарием невозможно принимать участие в гонке индустрии программного и информационного обеспечения. Вместе с тем современный инструментарий должен опираться не только на научные достижения, но и инженерно-технические, программно-аппаратные новшества и разработки, и помогать в решении актуальных задач в различных отраслях человеческой деятельности (экономических, технологических, медицинских, социальных и т.д.). В развитии информационных технологий особое место занимают процессы, направленные на моделирование и перенос на компьютерные платформы функций присущих аналитическим способностям человека, его когнитивным и творческим возможностям и задачам, которые обобщаются одним понятием – интеллект, или интеллектуальные системы (ИС). Различные методы, алгоритмы, формальный аппарат ИС позволяют выявлять текущие и перспективные потребности в информатизации различных отраслей производства и хозяйствования, адаптировать науку к конкретным промышленным технологиям, инженерным исследованиям с целью эффективного, сбалансированного их оснащения программным, аппаратным и информационным обеспечением.

Возрастание объемов информации, увеличение количества и качества задач, решаемых в процессе управления, усложнение управляющих и подконтрольных систем предъявляют все более жесткие требования к информационным системам как системам поддержки принятия решений. К традиционным условиям по обеспечению высокой эффективности работы, надежности, гибкости, эргономичности и др. на современном этапе развития добавляется требование, которое можно охарактеризовать как способность решать интеллектуальные задачи. Разнородность информации, усложнение функций и расширение возможностей вычислительных систем выводят на передний план несколько задач, в том числе систематизации и классификации данных, исследований задач классификации, называемых задачами распознавания образов, прогнозирования и способов их решения. Таким образом, усложнение и интеллектуализация информационных систем, помимо

всего прочего, приводит к необходимости наделения их развитыми способностями искусственного интеллекта. Это, в свою очередь, делает насущным решение проблем, входящих в сферу разработок теории распознавания образов, прогнозирования, экспертных систем, машинного творчества и т.д.

В последние годы активизировались теоретические исследования в различных математических областях искусственного интеллекта (ИИ): алгебры логики и теории предикатов (доказательств), семантического анализа, моделей знаний и данных, теории формальных языков, диалоговых и экспертных процедур, теории когнитивной графики и распознавания текстов на естественном языке и т.д. В начале 60-х годов прошлого века шли горячие споры, может ли машина «мыслить», «творить», «анализировать» и т.д. Прошедшие десятилетия вполне определенно дали положительный ответ. Современные программные системы, совмещенные с сенсорными системами, образуют робототехнические комплексы высокого уровня, наделенные ИИ, способные выполнять сложнейшие функции на больших глубинах, на других планетах и труднодоступных (опасных) для человека производственных местах.

Системы искусственного интеллекта способны выполнять функции, ранее считавшиеся исключительно прерогативой естественного интеллекта: доказывать математические теоремы, переводить тексты с одного языка на другой, диагностировать болезни, распознавать месторождения полезных ископаемых, умело играть в шахматы и другие интеллектуальные игры.

Широкое и массовое распространение получили интеллектуальные системы, которые классифицируются термином экспертные системы (ЭС), то есть системы, позволяющие на базе компьютеров накапливать, обновлять и корректировать знания из различных предметных областей. В своем развитии и совершенствовании, начиная со второй половины прошлого века, экспертные системы претерпевают качественные изменения со значительным возрастанием своей роли в информационных процессах принятия решений. Экспертные системы помогают не только накапливать и хранить информацию в удобном для пользователя виде, но и получать новые знания в медицине, геологии, геофизике, машиностроении, робототехнике, экономике, экологии и в других предметных областях.

Последние десятилетия эффективным инструментарием проявляет себя постепенно обособившаяся группа информационных систем, основанная на технологиях интеллектуального анализа данных (ИАД). Можно смело утверждать, что владение этой технологией является неотъемлемой составляющей технического специалиста (и не только) с целью успешной конкуренции на рынке труда. Технологии интеллектуального анализа данных – передовой рубеж аналитических систем, развитие методов статистического анализа, методов прогнозирования. Инструментарий ИАД является непосредственным ресурсом повышения производительности, эффективности и оперативности принятия решений на различных предприятиях промышленности, производства, бизнеса и, как следствие, оберегаемыми секретными корпоративными технологиями и системами.

Особенности современных теорий искусственного интеллекта и информатики заключаются в том, что они имеют тесную связь с лингвистикой, психологией и логикой, которые изучают явления, относящиеся к познанию, пониманию и умозаключениям. Эти связи носят взаимный характер: с одной стороны, сегодня лингвисты, психологи, математики, биологи переводят в программы те новые модели, которые они разрабатывают, а с другой – исследователи в области искусственного интеллекта изучают эти модели и пытаются воссоздать на их основе логику эффективных методов решения задач.

Особенно тесные взаимосвязи имеются между искусственным интеллектом, науками о познании и биологией. При этом основной задачей является воссоздание с помощью искусственных устройств (в основном с помощью компьютеров) разумных рассуждений и действий. Следовательно, методы искусственного интеллекта представляют собой экспериментальную, развивающуюся научную дисциплину.

Учебная дисциплина «Интеллектуальные информационные системы» является одной из самых динамичных дисциплин, поскольку непрерывно появляются новейшие разработки в различных разделах информатики. Курс лекций и практик, проводимых в рамках данной тематики, направлен на повышение качества выпускаемых специалистов, в разрезе научного анализа, управления и прогнозирования с применением современного инструментария искусственного интеллекта, основанного на использовании компьютерной техники.

Глава 1.
МЕТОДОЛОГИЯ КОМПЬЮТЕРНЫХ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Смена поколений вычислительной техники приравнивается к очередной научно-технической революции. С появлением новых поколений ЭВМ, средств коммуникации, многопроцессорных систем, развитием Internet-технологий появляется возможность решать не только принципиально новый класс задач во всех отраслях науки и техники, но и существенно расширяются возможности при решении прежних традиционных задач на новом более качественном уровне.

Более высокий качественный уровень в решении задач предполагает, прежде всего, обеспечение необходимой и достаточной интеллектуальной поддержкой. Интеллектуализация информационно-вычислительных систем (ИВС) имеет в виду не только использование нового поколения инструментальных средств, но нового поколения математического, алгоритмического и программного обеспечения.

Информационно-вычислительные системы с интеллектуальной поддержкой, как правило, применяются для решения сложных задач, где логическая (смысловая) обработка информации превалирует над вычислительной. Примерами подобных задач являются: понимание и синтез текстов на естественном языке (ЕЯ), понимание и синтез речи, анализ визуальной информации, управление роботами, анализ ситуаций и принятие решений и т.д.

1.1. Определения и классификации в теории интеллектуальных систем

Прежде всего, уточним, как будем называть системы с элементами искусственного интеллекта и что понимать под искусственным интеллектом и «интеллектуальной системой». По материалам публикаций [6, 12, 14, 40, 51].

Искусственный интеллект (ИИ, англ. *Artificial intelligence*, AI) – наука и технология создания интеллектуальных машин, особенно интеллектуальных компьютерных программ. ИИ связан со

сходной задачей использования компьютеров для понимания человеческого интеллекта, но не обязательно ограничивается биологически правдоподобными методами. Искусственный интеллект— это одно из направлений информатики, целью которого является разработка аппаратно-программных средств, позволяющих пользователю ставить и решать традиционно считающиеся интеллектуальными задачи.

Интеллектуальная система – это информационно-вычислительная система с интеллектуальной поддержкой при решении задач без участия оператора (лица, принимающего решение – ЛПР).

Интеллектуализированная система – это информационно-вычислительная система с интеллектуальной поддержкой при решении задач с участием оператора – ЛПР.

Система с интеллектуальной поддержкой – система способная самостоятельно принимать решения.

Под способностью самостоятельно принимать решение системой необходимо понимать это шире, то есть способность системы получать и анализировать информацию, понимать ее и делать новые выводы (пополняя ее), формулировать заключения, то есть «мыслить», помогая естественному интеллекту-человеку, который, в свою очередь, корректируя, «улучшает» принятое интегрированное решение.

Здесь следует оговорить, что существуют автоматы или просто механическое или электронное реле, которые реагируют на наличие или отсутствие сигнала или при контроле параметров работают по принципу «годен – негоден». Это тоже принятие решения, однако, отнесем их к ИВС с низким уровнем «интеллектуализации».

Под интеллектуализированной, интеллектуальной системой (ИС) будем понимать систему способную принимать решение в условиях:

- анализа большого массива информационной базы данных;
- ограниченной информации;
- неопределенности;
- многомерного пространства;
- необходимости распознавать ситуацию (образы, сцены и т.д.);

– динамических, нестационарных фактов, влияющих на решение задачи;

– формализации и представления знаний;

– экстраполяции и прогнозирования;

– адаптации, самообучения, самоорганизации и т.д.

Таким образом, если ИВС имеет необходимую математическую, алгоритмическую, программную и инструментальную поддержку в принятии решения в перечисленных условиях, то будем считать, что она имеет интеллектуальную поддержку при решении широкого класса разнообразных задач.

Так как терминология в области разработки ИС находится на стадии формирования, определим основные термины следующим образом.

Предметная область – объектно-ориентированным образом выделенная и формально описанная область человеческой деятельности (множество сущностей, описывающих область исследования или экспертизы).

Проблемная область – предметная область плюс совокупность решаемых в ней задач.

Неформализованные задачи – задачи, которые обладают одной или несколькими из следующих характеристик:

– они не могут быть заданы в числовой форме, то есть задаются в качественном виде или в терминах теории нечетных множеств;

– цели не могут быть выражены в терминах точно определенной целевой функции;

– не существует алгоритмического решения задач;

– алгоритмическое решение существует, но его нельзя использовать из-за ограниченности ресурсов (время, память).

Экспертная система (система, основанная на знаниях – СОЗ) – сложный программный комплекс, аккумулирующий в формальном виде знания специалистов в конкретных предметных областях.

Пользователь (конечный пользователь) – лицо, для которого предназначена система.

Инженер по знаниям (когнитолог, инженер-интерпретатор) – специалист по ИИ, выступающий в роли промежуточного буфера между экспертом и базой знаний.

Эксперт – высококвалифицированный специалист, согласившийся поделиться опытом в рассматриваемой предметной области.

Интерфейс пользователя – комплекс программ, реализующих диалог пользователям с ИС на всех стадиях функционирования ИС.

База знаний (БЗ) – ядро ИС, совокупность знаний предметной области, записанная на машинный носитель на языке представления знаний.

Решатель (машина логического вывода, дедуктивная машина, интерпретатор) – программа, моделирующая ход рассуждений эксперта на основании знаний, имеющихся в БЗ.

Подсистема объяснений – программа, позволяющая пользователю получать ответы на вопросы: *как* была получена та или иная рекомендация и *почему* система приняла такое решения?

Технология синтеза ЭС – технология создания на основе знаний экспертов систем, решающих неформализованные задачи в слабоструктурированных предметных областях.

Как показал многолетний опыт интенсивной разработки ИС для различных приложений, существует порядка 8–10 типовых задач (табл. 1.1), для которых использование технологии ИС приносит значительные результаты.

Таблица 1.1

Типовые задачи для использования технологий ИС

№ п/п	Тип задачи	Определение (адресуемые задачи)
1	2	3
1	Интерпретация	Процесс определения смысла данных (построение описаний по наблюдаемым данным)
2	Диагностика	Процесс обнаружения неисправностей (в технике и в живых организмах)
3	Слежение (мониторинг)	Непрерывная интерпретация данных в реальном масштабе времени и сигнализация о выходах параметров за допустимые пределы
4	Прогнозирование	Предсказание будущих событий на базе моделей прошлого и настоящего (вывод вероятных следствий из заданных ситуаций)
5	Планирование	Конструирование плана, то есть программы действий
6	Проектирование	Построение спецификаций на создание объектов с заранее определенными свойствами

Окончание табл. 1.1

1	2	3
7	Отладка, ремонт	Выработка рекомендаций по устранению неисправностей
8	Обучение	Диагностика, интерпретация, планирование, проектирование
9	Управление	Интерпретация, прогноз, планирование, моделирование, оптимизация выработанных решений, мониторинг

В общем случае все ИС можно подразделить на решающие задачи *анализа* и на решающие задачи *синтеза*. Примерами задач анализа являются задачи интерпретации данных и диагностики. Примерами задач синтеза являются задачи проектирования и планирования. Комбинированные задачи – задачи обучения, мониторинга, управления.

Класс ИС объединяет сегодня несколько тысяч различных систем, которые можно классифицировать по различным критериям, предлагаемым отдельными авторами [40].

Задачи, решаемые методами ИС [19, 51]:

1. **Классификация** – это отнесение объектов (наблюдений, событий) к одному из заранее известных классов.

2. **Кластеризация** – это группировка объектов (наблюдений, событий) на основе данных (свойств), описывающих сущность этих объектов. Объекты внутри кластера должны быть "похожими" друг на друга и отличаться от объектов, вошедших в другие кластеры. Чем больше похожи объекты внутри кластера и чем больше отличий между кластерами, тем точнее кластеризация.

3. **Регрессия**, в том числе задачи прогнозирования. Установление зависимости непрерывных выходных от входных переменных.

4. **Ассоциация** – выявление закономерностей между связанными событиями. Примером такой закономерности служит правило, указывающее, что из события *X* следует событие *Y*. Такие правила называются ассоциативными. Впервые эта задача была предложена для нахождения типичных шаблонов покупок, совершаемых в супермаркетах, поэтому иногда ее еще называют анализом рыночной корзины (**market basket analysis**).

5. **Последовательные шаблоны** – установление закономерностей между связанными во времени событиями, то есть обнаружение зависимости, что если произойдет событие *X*, то спустя заданное время произойдет событие *Y*.

6. **Анализ отклонений** – выявление наиболее нехарактерных шаблонов.

Существует множество научно-технических направлений и научных школ по искусственному интеллекту и интеллектуальным системам, часть из которых превалирует и пользуется наибольшей известностью и популярностью [11, 19, 40, 44, 57], в том числе:

Экспертные системы (ЭС, англ. *expert system*) – это сложные программные комплексы, аккумулирующие знания специалистов в конкретных предметных областях и тиражирующие этот эмпирический опыт для консультаций менее квалифицированных пользователей. Экспертная система – компьютерная система, способная частично заменить специалиста-эксперта в разрешении проблемной ситуации.

Представление знаний и разработка систем, основанных на знаниях (knowledge-based systems) – направление в области изучения искусственного интеллекта, связанное с разработкой моделей представления знаний, созданием баз знаний, образующих ядро экспертных систем. Включает в себя модели и методы извлечения и структуризации знаний и сливается с инженерией знаний.

Машинный перевод (англ. *natural language processing*) – выполняемое на компьютере действие по преобразованию текста на одном естественном языке (ЕЯ) в эквивалентный по содержанию текст на другом естественном языке, а также результат такого действия. Качество перевода алгоритмов, основанных на морфологическом анализе, достигло своего интенсивного предела и повышается за счет экстенсивных факторов: объема словаря, объема информации, сложность морфологических конструкций, лексических единиц и т.д.

Интеллектуальные роботы (англ. *robotics*) – это технические устройства, предназначенные для автоматизации человеческого труда. Выделяют несколько поколений в истории создания и развития робототехники:

– *роботы с жесткой схемой управления.* Практически все современные промышленные роботы принадлежат к первому поколению. Фактически это программируемые манипуляторы;

– *адаптивные роботы с сенсорными устройствами.* Есть образцы таких роботов, но в промышленности они пока используются мало;

– *самоорганизующиеся или интеллектуальные роботы.* Это – конечная цель развития робототехники.

Распознавание образов (англ. *pattern recognition*) – одно из направлений искусственного интеллекта, берущее начало у самых его истоков и выделившееся в самостоятельную науку. Ее основной подход – описание *классов объектов* через определенные значения значимых признаков. Каждому объекту ставится в соответствие матрица признаков, по которой происходит его распознавание. Процедуры распознавания использует чаще всего специальные математические модели, алгоритмы и функции, разделяющие объекты на классы. Это направление близко к машинному обучению.

Машинное обучение (обучение и самообучение) (англ. *machine learning*) – активно развивающаяся область искусственного интеллекта, которая включает в себя обучение на примерах, традиционные подходы из теории распознавания образов, модели методы и алгоритмы, ориентированные на автоматическое накопление и формирование знаний на основе анализа и обобщения.

Интеллектуальный анализ данных (англ. *data mining, knowledge discovery*). В последние годы стремительно развиваются системы data mining – анализа данных и поиска закономерностей в базах данных. Интеллектуальный анализ данных – это собирательное название, используемое для обозначения совокупности методов обнаружения в данных ранее неизвестных, нетривиальных, практически полезных и доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности.

Программное обеспечение систем ИИ (англ. *software engineering for AI*). В рамках этого направления разрабатываются специальные языки для решения интеллектуальных задач, в которых традиционно упор делается на преобладание логической и символьной обработки над вычислительными процедурами. Помимо этого создаются пакеты прикладных программ, ориентированные на промышленную разработку интеллектуальных систем, или программные инструментарии искусственного интеллекта, Достаточно популярно также создание так называемых пустых экспертных систем или «оболочек», базы знаний которых можно наполнять конкретными знаниями, создавая различные прикладные системы.

Игры и машинное творчество (англ. *art machines*) – методы ИИ, традиционно включающие в себя игровые интеллектуальные задачи: шахматы, шашки, го. В современных играх к ним относят методы удержания игрока и адаптации под него игрового пространства. Методы анимации позволяют синтезировать компьютерные мультфильмы, широко используемые в компьютерных **играх**, научных исследованиях и в системе когнитивной графики. Также алгоритмы «машинного творчества» направлены на создание компьютером литературных произведений (сочинений, стихов), компьютерной музыки.

1.2. Проблемная область искусственного интеллекта

В настоящее время в области искусственного интеллекта выделено шесть основных проблем (направлений развития) [14, 28, 30].

1. Представление знаний. В рамках этой проблемы решаются задачи, связанные с формализацией и предоставлением знаний в памяти ИС. Для этого разрабатываются специальные модели представления знаний и языки для описания знаний, выделяются различные типы знаний. Изучаются источники, из которых ИС может черпать знания, и создаются процедуры и приемы, с помощью которых возможно приобретение знаний для ИС. Проблема представления знаний для ИС чрезвычайно актуальна, так как ИС – это система, функционирование которой опирается на знания о проблемной области.

2. Манипулирование знаниями. Для того чтобы знаниями можно было пользоваться при решении задач, ИС должна уметь:

1) оперировать знаниями;

2) пополнять знания (с помощью разрабатываемых способов на основе неполного описания знаний);

3) классифицировать хранящиеся в системе знания;

4) обобщать по тем или иным разработанным процедурам знания;

5) формировать на основе знаний абстрактные понятия;

6) осуществлять достоверный и правдоподобный вывод на основе имеющихся знаний с помощью создаваемых методов;

7) пользоваться моделями рассуждений, имитирующих особенности человеческих рассуждений.

Манипулирование знаниями и представление знаний – эти два

направления тесно связаны друг с другом. Создающаяся в настоящее время теория баз знаний включает исследования, относящиеся как к первому, так и ко второму направлениям.

3. Общение. В круг задач этого направления входят:

– проблема понимания связных текстов;

– понимание речи и синтез речи;

– теория моделей коммуникации между человеком и ИС;

– задачи формирования объяснений действий ИС, которые она должна уметь порождать по просьбе человека;

– комплекс задач, связанных с интеграцией в единый внутренний образ сообщений различной модальности (речевых, текстовых, зрительных, и т.п.), полученных в процессе коммуникации.

На основе исследований в этом направлении формируются методы построения лингвистических процессоров, человеко-машинного интерфейса, диалоговых систем (ДС), визуализации данных и т.д., целью которых является обеспечение комфортных условий для общения человека с ИС.

4. Восприятие. Это направление включает:

– проблемы анализа трехмерных сцен;

– разработку методов представления информации о зрительных образах в базе знаний;

– создание методов перехода от зрительских сцен к их текстовому описанию и методов обработки перехода;

– разработку процедур когнитивной графики (КГ);

– создание средств для порождения зрительских сцен на основе внутренних представлений в ИС.

Существуют большие возможности в повышении уровня интеллектуальности ИС за счет обработки зрительной (образной) информации и соотнесения ее с обработкой символьной (текстовой) информации.

5. Обучение. Основная черта ИС – это способность к обучению, то есть решение задач, с которыми они ранее не встречались. Для этого необходимо (перечислим основное):

1) создать методы формализации условий задачи по описанию проблемной ситуации или по наблюдению за этой ситуацией;

2) научиться переходу от известного решения частных задач (примеров) к решению общей задачи (синтез);

3) создать приемы декомпозиции исходной для ИС задачи на более мелкие так, чтобы они для ИС оказались известными (анализ);

4) разработать нормативные процедуры самого процесса обучения;

5) создать теорию подражательного поведения.

6. Поведение. Так как ИС должны действовать в некоторой окружающей среде, то необходимо разработать специальные поведенческие процедуры (бихевиористические модели), которые позволили бы им адекватно взаимодействовать с окружающей средой, другими ИС и людьми. Для достижения такого взаимодействия необходимо вести исследования в ряде направлений и создать: модели целесообразного поведения, нормативного поведения, ситуационного поведения, специальные методы многоуровневого планирования и коррекции планов в динамических ситуациях.

1.3. Данные и знания

Данные – это отдельные факты, характеризующие объекты, процессы и явления предметной области, а также их свойства.

При обработке на ЭВМ данные трансформируются условно на пять классов:

1) данные как результат измерений и наблюдений;
2) данные на материальных носителях информации справочники);
3) модели (структуры) данных в виде визуальных образов (диаграмм, графиков и т.д.);
4) данные в компьютере на языке описания данных;
5) базы данных на машинных носителях информации.

Знания, основанные на данных, полученных эмпирическим путем, представляют собой результат мыслительной деятельности человека, направленной на обобщение его опыта, полученного в результате практической деятельности.

1.3.1. Данные, базы данных, системы управления базами данных

Параллельно с развитием структуры ЭВМ происходило развитие информационных структур для представления данных. Появились способы описания данных в виде:

- векторов;
- матриц;

– списочных структур;

– иерархических структур;

– структур, создаваемых программистом (абстрактных типов данных).

В настоящее время в языках программирования высокого уровня используются абстрактные типы данных, структура которых создается программистом. Появление баз данных (БД) знаменовало собой еще один шаг по пути организации работы с декларативной информацией [26, 27, 32].

По мере развития исследований в области ИС возникла *концепция знаний*, которая объединила в себе многие черты процедурной и декларативной информации.

База данных как естественнонаучное понятие характеризуется двумя основными аспектами – информационным и манипуляционным. Первый аспект отражает такую структуризацию данных, которая является наиболее подходящей для обеспечения информационных потребностей, возникающих в предметной области (ПО). С каждой ПО ассоциируется совокупность «информационных объектов», связей между ними. Манипуляционный аспект БД касается смысла тех действий над структурами данных, с помощью которых осуществляются выборка из них различных компонентов, добавление новых, удаление и обновление устаревших компонентов структур данных, а также их преобразования.

Под системой управления БД (СУБД) понимается комплекс средств (языковых, программных и, возможно, аппаратных), поддерживающих определенный тип БД. Главное назначение СУБД, с точки зрения пользователей, состоит в обеспечении их инструментарием, позволяющим оперировать данными в абстрактных терминах (именах и/или характеристиках информационных объектов).

На практике в каждом рассматриваемом случае пути решения этих задач выбираются исходя из специфики ПО, функциональных возможностей доступных СУБД и вычислительных систем, допустимых затрат на создание ИС и др.

Основная масса работ по теоретическим и теоретико-прикладным исследованиям в области ИС за весь период ее развития (начиная с конца 60-х годов) была связана с моделированием БД, разработкой архитектур СУБД и их языковых средств. Рассмотрим кратко основные направления исследований в области БД.

На первом этапе (примерно до 1975 г.) развития этой области наиболее характерными являются исследования способов структуризации данных, не зависящих от специфики организации среды хранения (памяти ЭВМ). Разработанные способы структуризации данных получили название моделей данных. Каждая такая модель характеризуется определенными средствами и методами структурирования данных. Наиболее известны, так называемая, иерархическая, сетевая и реляционная модели данных.

На втором этапе (примерно 1975–1980 гг.) развития теории БД акценты заметно смещаются в сторону анализа прагматико-смысловых интерпретаций данных. Все это привело к формированию нового направления исследований, ориентированного на вскрытие структур и закономерностей ПО. Для этого периода характерно построение моделей данных путем расширения иерархической, сетевой и реляционной моделей за счет включения элементов «смысловой интерпретации» информационных объектов ПО, связей и зависимостей между ними.

В первой половине 80-х годов внимание исследователей сконцентрировалось на построении самостоятельных, автономных моделей ПО и выработке методов отображения их в модели БД. Что же касается структур данных и средств манипулирования ими, то они в данном случае полностью или частично выпали из поля зрения исследователей. Ввиду большой сложности проблемы моделирования ПО предложенные языковые средства моделирования представляют собой, как правило, набор синтаксических структур без строгой семантической их интерпретации. Поэтому, как и следовало ожидать, исследования по моделированию ПО не привели к появлению глубоких результатов в теории БД, хотя и обеспечили богатый фактологический материал, касающийся этой проблематики.

В целом проблематика БД и ИС развивалась по экстенсиональному пути. Это означает, что основное внимание уделялось широте охвата вопросов, не придавая особого значения расстановке акцентов на них, не отделяя главные, носящие принципиальный характер вопросы от второстепенных. Большое количество публикаций в этой области, разобщенность взглядов авторов, интуитивная трактовка понятий породили своего рода «терминологический Вавилон». Все это отрицательно отразилось на интенсиональном

характере развития этой проблематики. Вопросы построения и исследования основополагающих концепций БД, познания их глубинной природы затрагивались слабо. Только в рамках реляционного подхода был сделан определенный (хотя и довольно локальный) шаг в сторону интенсионализации проблематики БД. Следует отметить, что такое положение в области БД не делает ее каким-то исключением. Любая достаточно емкая и комплексно трудная научная область в изначальном своем развитии характеризуется тем, что экстенсиональный этап будет предшествовать интенсиональному.

Эти исследования, как нам представляется, позволят теснее связать проблематику БД, языков манипулирования данными, с одной стороны, и проблематику современных языков программирования и поддерживающих их систем – с другой [15, 50]. Заметно стала сближаться область баз данных (знаний) с областью искусственного интеллекта. Это очень важно, так как одним из основных требований, предъявляемых к современным ИС (и особенно к ИС типа экспертных систем), является высокий уровень их «интеллектуальности» во взаимодействии с пользователем.

Однако успехи указанных исследований, глубина получаемых результатов, полезность разрабатываемых проектов ИС во многом зависят от того, насколько удачными и точными будут разработаны сами основания теории ИС. Это, прежде всего, система таких понятий (построенных на теоретико-множественной основе), на которых возможно развитие самой теории БД как вглубь, так и вширь. Выработка оснований теории даст возможность по-новому переосмыслить накопленные теоретические результаты, богатый фактологический материал, опыт построения и использования реальных СУБД, ИС и на основе всего этого глубже взглянуть на природу баз данных, баз знаний, построить высокопродуктивные информационные технологии.

После краткого ознакомления с основными направлениями исследований в области БД перейдем к более подробному обсуждению вопросов, касающихся концепций структур данных, средств манипулирования ими и базирующихся на них моделей БД. При этом основное внимание будет уделено семантическому (как определяющему) аспекту структур данных и средств их обработки.

1.3.2. Знания, информационные единицы, информационные системы управления базами знаний

Рассмотрим общую совокупность качественных свойств для знаний (специфических признаков знаний) и перечислим ряд особенностей, присущих этой форме представления информации в ЭВМ и позволяющих охарактеризовать сам термин «знаний» [14, 15, 26].

Прежде всего, знания имеют более сложную структуру, чем данные (метаданные). При этом знания задаются как экстенсеонально (то есть через набор конкретных фактов, соответствующих данному понятию и касающихся предметной области), так и интенсионально (то есть через свойства, соответствующие данному понятию и схеме связей между атрибутами). С учетом сказанного представим свойства:

1. Внутренняя интерпретируемость знаний. Каждая информационная единица (и.е.) должна иметь уникальное имя, по которому ИС находит ее, а также отвечает на запросы, в которых это имя упомянуто. Когда данные, хранящиеся в памяти, были лишены имен, то отсутствовала возможность их идентификации системой. Данные могла идентифицировать лишь программа.

При переходе к знаниям в память ЭВМ вводится информация о некоторой *протоструктуре информационных единиц.* В рассматриваемом примере она представляет собой специальное машинное слово, в котором указано, в каких разрядах хранятся сведения о фамилиях, годах рождения, специальностях и стаже. При этом должны быть заданы специальные словари, в которых перечислены имеющиеся в памяти системы фамилии, года рождения, название специальностей и курса. Все эти *атрибуты* могут играть роль имен для тех машинных слов, которые соответствуют строчкам таблицы. По ним можно осуществлять поиск нужной информации. Каждая строка таблицы будет экземпляром протоструктуры. В настоящее время СУБД обеспечивают реализацию внутренней интерпретируемости всех и.е., хранимых в базе данных.

2. Структурированность (рекурсивная структурированность) знаний. И.е. должны обладать гибкой структурой. Для них должен выполняться «принцип матрешки», то есть рекурсивная вложимость одних и.е. в другие. Каждая и.е. может быть включена

в состав любой другой, и из каждой и.е. можно выделить некоторые составляющие ее и.е. Другими словами, должна существовать возможность произвольного установления между отдельными и.е. отношений типа «часть–целое», «род–вид» или «элемент–класс».

3. Связность (взаимосвязь единиц знаний). В информационной базе между и.е. должна быть предусмотрена возможность установления связей различного типа. Прежде всего, эти связи могут характеризовать отношения между и.е. Семантика отношений может носить декларативный или процедурный характер. Например, две или более и.е. могут быть связаны отношением «одновременно», две и.е. – отношением «причина–следствие» или отношением «аргумент–функция». Приведенные отношения характеризуют декларативные знания. Различают *отношения структуризации, функциональные отношения, каузальные отношения и семантические отношения.* С помощью первых задаются иерархии и.е., вторые несут процедурную информацию, позволяющую находить (вычислять) одни и.е. через другие, третьи задают причинно-следственные связи, четвертые соответствуют всем остальным отношениям.

Между и.е. могут устанавливаться и иные связи, например, определяющие порядок выбора и.е. из памяти или указывающие на то, что две и.е. несовместимы друг с другом в одном описании.

Перечисленные три особенности знаний позволяют ввести общую модель представления знаний, которую можно назвать *семантической сетью,* представляющей собой иерархическую сеть, в вершинах которой находятся и.е. Эти единицы снабжены индивидуальными именами. Дуги семантической сети соответствуют различным связям между и.е. При этом иерархические связи определяются отношениями структуризации, а иерархические связи – отношениями иных типов.

4. Семантическая метрика (семантическое пространство знаний с метрикой). На множестве и.е. в некоторых случаях полезно задавать отношение, характеризующее ситуационную близость и.е., то есть ассоциативной связи между и.е. Его можно было бы назвать отношением релевантности для и.е. Такое отношение дает возможность выделять в информационной базе некоторые типовые ситуации (например, «покупка», «регулирование движе-

ния на перекрестке»). Отношение релевантности при работе с и.е. позволяет находить знания, близкие к уже найденным.

5. Активность знаний. С момента появления ЭВМ и разделения используемых в ней и.е. на данные и команды создалась ситуация, при которой данные пассивны, а команды активны. Все процессы, протекающие в ЭВМ, инициируются командами, а данные используются этими командами лишь в случае необходимости. Для ИС эта ситуация неприемлема. Как и у человека, в ИС актуализации тех или иных действий способствуют знания, имеющиеся в системе. Таким образом, выполнение программ в ИС должно инициироваться текущим состоянием информационной базы. Появление в базе фактов или описаний событий, установление связей может стать источником активности системы.

Следует упомянуть о функциональной целостности знаний, то есть возможности выбора желаемого результата, времени и средств получения результата, средств анализа достаточности полученного результата.

Перечисленные пять особенностей и.е. определяют ту грань, за которой данные превращаются в знания, а базы данных перерастают в базы знаний (БЗ). Совокупность средств, обеспечивающих работу со знаниями, образует *систему управления базой знаний* (СУБЗ). К БЗ, в которых в полной мере была бы реализована внутренняя интерпретируемость, структуризация, связность, введена семантическая мера и обеспечена активность знаний, еще необходимо проделать определенный путь.

Все приведенные выше качественные свойства знаний касаются в основном концептуально-логического уровня представления знаний и связаны со сложной природой знания, изучение которой происходит на междисциплинарном стыке таких наук, как кибернетика, лингвистика, психология и т.д.

1.4. Формальные модели данных и знаний

Выделим три основных раздела исследований в области представления знаний, тесно связанных с вопросами создания комплексных автоматизированных информационных систем:

Первый раздел касается воспроизведения на ЭВМ творческих для человека процедур, например, создание программ, имитирую-

щих способность человека к обучению, обработки языков программирования, общения.

Второй раздел связан с новой информационной технологией решения задач на ЭВМ путем разработки интеллектуального интерфейса, который смог бы обеспечить простоту процесса взаимодействия пользователя с ЭВМ и, который позволил бы исключить посредников-программистов и необходимость регулярного сопровождения программ. Процесс внедрения средств интеллектуального интерфейса опирается на методы работы со знаниями: их представления, хранение, преобразование и т.п.

Третий раздел включает исследования, связанные с созданием новых принципов обработки информации и решения задач на основе операций со знаниями.

Исходя из вышеизложенного, остановимся на следующем определении «знания» [15, 41, 50].

Знание – информация, выраженная, зафиксированная и функционирующая в особых символических знаковых системах: языках. Традиционная гносеология установила, что знание возникает в результате интерференции и суперпозиции ряда сложных процессов. Изучение этих процессов привело к возникновению ряда дисциплин – нейрофизиологии, психологии, психолингвистики, логики и т.д. Существует проблема установления взаимоотношений между биологическими (мозг) и механическими (компьютер) устройствами, с точки зрения их способности вырабатывать и представлять знания.

Укажем три типа отношений знаков и знаковых конструкций, фиксирующих и выражающих знания.

1. Семантические отношения – правила установления смысла и значения знака или знаковой конструкции. Знак и знаковая конструкция обозначают некую вещь, процесс, предмет, ситуацию, действие или систему действий, каждый знак или знаковая конструкция должны иметь смысл и значение, чтобы расцениваться как средство выражения знаний.

2. Синтаксические отношения – правила синтаксиса между знаковыми конструкциями. Они указывают, как из одних осмысленных выражений получить другие, тоже осмысленные, с более или менее определенным значением.

3. Прагматические отношения - правила описания наборов и последовательностей действий, которые должны или могут осуществляться на базе тех или иных знаний. Все системы действий, выполняемых человеком, опираются на знания. Они образуют информационную базу деятельности.

В информационных технологиях, связанных с разработкой ИС, представление знаний осуществляется с использованием формальных моделей, которые делятся на декларативные и процедурные.

1.4.1. Декларативные модели представления данных и знаний

Одной из проблем ИС является обеспечение пользователей компетентными заключениями, касающимися определенных предметных областей. В основу решения данной проблемы положим исследования в области моделей представления знаний о предметной области или моделей информационного фонда. *Декларативное* описание моделей с позиций разработки программных систем будем осуществлять с точки зрения парадигмы «структуры предметной области». Выделим следующие виды моделей [27, 50].

1. Иерархическая структура. Предметная область представляется в виде частей Q_i, каждая из которых описывается в виде дерева с ветвями *r*-го ранга и *j*-м количеством элементов на r-м ранге.

$$Q_i = \{Q_{r,j}\},\ P = \{Q_1, \ldots, Q_n\}.$$

2. Сетевая структура. Вся предметная область рассматривается как совокупность частей, связанных между собой бинарными отношениями. Отдельные части, поддающиеся разбиению, имеют такую же структуру.

Объект идентифицируется своим именем x_i. Бинарные отношения описываются с помощью символов $R\kappa$. Тогда описание предметной области примет вид:

$$C_i = (x_j, R_t, x_g);$$
$$P = \{C_1, \ldots, C_n\}.$$

3. Релятивная структура. Предметная область рассматривается как совокупность нескольких множеств, между элементами которых существуют отношения.

4. Объектно-характеристическая структура. Предметная область представляется множеством объектов. Каждому объекту

приписываются определенные характеристики. Объект идентифицируется по своему наименованию, а характеристики по своему наименованию и значению, свойственному данному объекту.

$$P = \{X_1 \ldots X_k\},\ X_i = (x, \{(y_1, z_1), \ldots (y_j, z_j)\}),\quad i = 1, ..., k,$$

где P – предметная область; X_i – объект описания; x – имя объекта; y_j – название характеристики; z_j – значение характеристики.

Описание каждого объекта можно представлять (с избыточностью) как набор троек: $(x_i, y_j, z_{i,j})$.

5. Триадная структура. Предположим, что предметная область может быть описана с помощью предиката(-ов) $F(X_1, \ldots, X_k)$, где X_i – предметные переменные. Совокупность названия F предиката и номера i-й позиции, занимаемой предметной переменной X_i, называется ролью R_i этой переменной и ее значения: $R_i = (F, i)$.

Чтобы освободиться от необходимости перечисления значений предметных переменных в том порядке, в котором они представлены в предикате, каждому из них предпосылают название его роли (R, X). Чтобы не группировать пары (R, X) в строки в виде ключевого символа S вводят название строки и представляют описание в виде троек:

$$S_i = (S, R, X),\ P = \{S_1, \ldots, S_n\},$$

где S – имя отношения; R – роль; X – исполнитель роли.

6. Фреймовая структура. Стремление разработать описание структуры предметной области, соединяющее и обобщающее достоинства различных моделей, привело к возникновению фреймового представления.

Предметная область представляется множеством объектов, между которыми существуют отношения. Каждый объект X_i является собой множество слотов Y_i. Структура слота – есть пара (y_i, s_i), где y_i – имя слота, s_i – значение слота. В качестве значения слота s_i могут выступать не только пассивные значения данных $Z_i \subseteq Y_i$ (как в объектно-характеристической структуре), но и активные – программы $A_i \subseteq Y_i$, а также ссылки $S_i \subseteq Y_i$ на другие объекты. Последние позволяют организовывать сложные взаимосвязи (отношения Rs) между объектами:

$$P = \{X_1 \ldots X_k\},\ X_i = (x, \{(y_1, s_1), \ldots (y_j, s_j)\}),\ \ \exists S_i \mid s_j \in S_i\colon (x_g\ Rs_j\ x_m),$$

$$i = 1 \ldots k.$$

где X_i – объект описания; x_i – имя объекта; $Y_i = \{(y_1, s_1), \ldots (y_j, s_j\}$ – множество слотов описания объекта.

Описатель объекта по составу и типу слотов и есть декларативное понятие фрейма.

$$F_i = (fi, [(y_1, t_1), \ldots, (y_k, t_k)]^n),$$

где f_i – описатель фрейма (ключ, комментарий и пр.); y_i – имя слота; t_i – тип слота; n – список экземпляров (при $n > 1$ получаем релятивную структуру).

Рассматриваемая фреймовая организация при определенных допущениях позволяет имитировать модели данных с 1-й по 5-ю. Необходимо отметить, что концепция фреймов в практическом применении имеет инвариантную реализуемость, поэтому ее формула носит сугубо теоретическое, обобщенно-концептуальное значение.

Но проблема ИС состоит не только в том, как представлять, но и как использовать знания, которыми располагают эксперты. Модели «использования» знаний будем относить к *процедурным* моделям знаний (в отличие от декларативных моделей) [50].

Выделим два подхода к интеллектуализации автоматизированных систем, связанных с процедурным представлением моделей знаний:

1. Логический подход связан с процессами решения различных задач путем формальной логики: в исчислении высказываний, логике предикатов. Чрезмерный перебор вариантов, порождаемых ЭВМ на основе формальных законов логики, не позволил подойти к решению практических задач.

2. Когнитивный подход. Модели знаний представляют собой симбиоз системы вывода решений на основе правил продукции с системой расширенного исчисления предикатов, заданных на дискретных структурах. Его тезис – «знания специалиста решают все».

На сегодняшний день можно говорить о формировании нового логико-когнитивного подхода. На основе объектно-ориентированного (логического) анализа удается установить формальные логические связи между когнитивными слоями формализации и существенно использовать промежуточную интерпретацию применяемых категорий.

1.4.2. Процедурные модели знаний

В соответствии с введенными видами подходов, классифицируем модели представления знаний на логические (формальные) и эвристические (формализованные). В основе логических моделей представления знаний лежит понятие формальной системы (теории). В логических моделях, как правило, используется исчисление предикатов первого порядка, дополненное рядом эвристических стратегий, а отношения, существующие между отдельными единицами знаний, выражаются только с помощью синтаксических правил, используемых формальной системой.

С целью определения канонического представления моделей знаний исследуем наиболее развитые модели представления знаний и унифицируем их морфологию [50], сведя ее к однотипной форме представления в соответствии со структурой рис. 1.1.

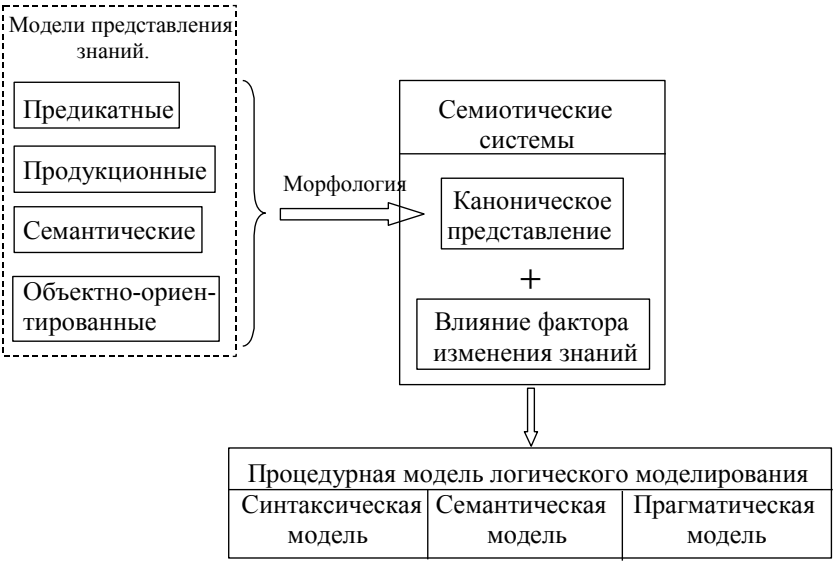


Рис. 1.1. Структура формирования процедурной модели логического моделирования

Модели представления знаний на базе логики предикатов. В основе *логических моделей* лежит понятие формальной теории T_1, задаваемой четверкой:

$$T_1 = (V, L, F, R), \quad (1.1)$$

где V – счетное множество базовых символов (алфавит); L – множество синтаксических правил, позволяющее строить из V синтаксически правильные выражения – формулы теории; F – выделенное множество формул, называемых аксиомами теории T, то есть множество априорно истинных формул; R – конечное множество отношений между формулами, называемыми правилами вывода.

Наиболее распространенной формальной системой, используемой для представления знаний, является исчисление предикатов первого порядка. Алфавит исчисления предикатов состоит из набора символов, к которым относятся знаки пунктуации, логические связки, знаки-кванторы, символы формирования переменных и пр. Основной задачей, решаемой в рамках исчисления предикатов, является выяснение истинности или ложности заданной формулы на некоторой области интерпретации. При этом особая роль отводится общезначимым формулам, то есть формулам, истинным при любой интерпретации, и невыполнимым формулам, то есть формулам, ложным при любой интерпретации.

Вывод в логических моделях [29]**.** Вывод в формальной логической системе является процедурой, которая из заданной группы выражений выводит отличное от заданных семантически правильное выражение. Эта процедура, представленная в определенной форме, и является правилом вывода. Если группа выражений, образующая посылку, является истинной, то должно гарантироваться, что применение правила вывода обеспечит получение истинного выражения в качестве заключения.

Наиболее часто используются два метода [2]. Первый – метод правил вывода, или метод естественного (натурального) вывода, названный так потому, что используемый тип рассуждений в исчислении предикатов приближается к обычному человеческому рассуждению. Второй – метод резолюций. В его основе лежит исчисление резольвент.

Метод правил вывода. В разной литературе можно встретить разные названия метода правил вывода, например, правила дедуктивных выводов или более часто **modus ponens**. Принцип работы правил вывода хорошо иллюстрирует следующий пример: «Если

высказывание *«A»* влечет (имплицирует) высказывание *«B»* и высказывание *«A»* истинно, то, следовательно, *«B»* истинно».

В логике предикатов имеются универсальные правила, оперирующие с формулами, содержащими свободные переменные. Решение задач (получение выводов) в логических моделях может основываться на применении подобных правил к исходной совокупности истинных предикатов как доказательство правильности какого-либо составного предиката. Поскольку ответ получается как заключение из комбинации уже существующих логических формул, то по аналогии с выводами в продукционных моделях его можно назвать прямым (обратным) выводом. Однако всегда следует учитывать, что в формальной логике причинно-следственные отношения игнорируются.

Суть процедуры вывода заключается в рекурсивном применении подстановки известных значений в составной предикат. При этом принципиально гарантируется, что доказательство истинности результата можно проверить формальной процедурой.

Если в формальной логической модели механизм логического вывода использует метод правил вывода, то есть основания эту модель отнести к продукционным или логико-лингвистическим.

Пример: вывод решения в логической модели на основе правила вывода – modus ponens.

Даны утверждения:

– «Карась – рыба»;
– «Рыбы – обитают в воде»;
– «Обитающие в воде – могут попасть в сети».

Требуется доказать утверждение **«Карась может попасть в сети»**.

Решение:

Шаг 1. Представим высказывания в предикатной форме:

Рыба(Карась)

$\forall$(**X**)(Рыбы(**X**)$\rightarrow$Обитают_в_воде(**X**))

$\forall$(**X**)(Обитающие_в_воде(**X**) $\rightarrow$ Могут_попасть_в_сети(**X**)

Шаг 2. На основе правила вывода во втором предикате получим утверждение: «Карась обитает в воде».

Шаг 3. На основе правила вывода в третьем предикате получим утверждение: «Карась может попасть в сети».

Метод резолюций. Метод резолюций опирается на исчисление резольвент. Существует теорема, утверждающая, что вопрос о доказуемости произвольной формулы в исчислении предикатов сводится к вопросу о доказуемости пустого списка в исчислении резольвент. Поэтому доказательство того, что список формул в исчислении резольвент пуст, эквивалентно доказательству ложности формулы в исчислении предикатов.

Метод резолюций основывается на доказательстве истинности или ложности выдвинутого предположения методом «от противного». Для этого в исходное множество предложений включают аксиомы формальной системы и отрицание доказываемой гипотезы. Если в процессе доказательства возникает противоречие между отрицанием гипотезы и аксиомами, выражающееся в нахождении пустого списка (дизъюнкта), то выдвинутая гипотеза правильна.

В методе резолюций множество предложений обычно рассматривается как составной предикат, содержащий несколько предикатов, соединенных логическими функциями и кванторами существования и общности. Так как одинаковые по смыслу предикаты могут иметь разный вид, то предложения преобразуются в клаузальную форму – разновидность конъюнктивной нормальной формы (КНФ), в которой удалены кванторы существования, всеобщности, символы импликации, равнозначности и др. Клаузальную форму называют сколемовской конъюнктивной формой.

Если требуется методом резолюций доказать истинность какого-либо логического утверждения, то отрицание этого утверждения преобразуется в клаузальную форму, по его предложениям выполняется поиск пустого предложения с использованием унификации и вывода резольвент. Невыполнимость отрицания подтверждает истинность рассматриваемого утверждения.

Метод резолюций получил широкое распространение из-за высокой эффективности машинной обработки. На его основе построен язык «Prolog».

Существенным недостатком метода резолюций является то, что он предназначен только для доказательства теорем. Он не пригоден для порождения новых предложений. К тому же, если предложение не является теоремой, резолюция может привести к построению бесконечного дерева решений.

Решение задачи примера метода правил вывода. Требуется доказать утверждение **«Карась может попасть в сети»**.

Шаг 1. Преобразуем высказывания в дизъюнктивную форму:

Предикатная форма:	Дизъюнктивная форма:
Рыба(карась)	Рыба(Карась)
∀(X)(Рыбы(X)→Обитают_в_воде(X))	¬(Рыбы(X) ∨ Обитают_в_воде(X))
∀(Y)(Обитающие_в_воде(Y) → Могут_попасть_в_сети(Y)	¬ (Обитающие_в_воде(Y) ∨ Могут_попасть_в_сети(Y))

Шаг 2. Запишем отрицание целевого выражения (требуемого вывода):

¬ Могут_попасть_в_сети(карась)

Шаг 3. Составим конъюнкцию всех дизъюнктов (т. е. построим КНФ), включив в нее отрицание целевого выражения:
(¬(Рыбы(X) ∨ Обитают_в_воде(X))) ∧ (¬ (Обитающие_в_воде(X) ∨ Могут_попасть_в_сети(X)))∧ (Могут_попасть_в_сети(карась)) ∧ (¬ Могут_попасть_в_сети(карась))

Шаг 4. В цикле проведем операцию поиска резольвент над каждой парой дизъюнктов:

1 Дизъюнкт	2 Дизъюнкт
¬(Рыбы(X) ∨ Обитают_в_воде(X))	¬ (Обитающие_в_воде(Y) ∨ Могут_попасть_в_сети(Y))
Резольвента: ¬ Рыбы(Y) ∨ Могут_попасть_в_сети(Y)	
¬ Рыбы(Y) ∨ Могут_попасть_в_сети(Y)	Рыбы(карась)
Резольвента: Могут_попасть_в_сети(карась)	
Могут_попасть_в_сети(карась)	¬ Могут_попасть_в_сети(карась)

Получение пустого дизъюнкта означает, что высказывание «карась не может попасть в сети» ложно, значит истинно высказывание «карась может попасть в сети».

В целом метод резолюций интересен благодаря простоте и системности, но применим только для ограниченного числа случаев (доказательство не должно иметь большую глубину, а число потенциальных резолюций не должно быть большим). Кроме метода резолюций и правил вывода существуют другие методы получения выводов в логике предикатов.

Продукционные модели знаний. Такие модели превосходят предикатные путем использования эвристик *F*, которые дают возможность как адекватного представления проблемной среды, так и эффективного использования правил вывода *R*. К эвристическим

моделям, используемым в экспертных системах, можно отнести сетевые, продукционные и объектно-ориентированные модели. Следует отметить, что продукционные модели, используемые для представления знаний в экспертных системах, отличаются от формальных продукционных систем тем, что они используют более сложные конструкции правил, а также содержат эвристическую информацию о специфике проблемной среды, выражаемую часто в виде семантических структур. Эвристики строятся в соответствии с набором правил конструирования L на базе алфавита V. Продукционные модели будем представлять формулой:

$$T_p = (V, L, F, R), \qquad (1.2)$$

где V – набор символов для формирования правил; L – язык конструирования правил; F – эвристики в виде правил; R – правила вывода (отношения эвристик).

Продукции являются наиболее популярными средствами **представления знаний** в информационных системах [39]. В общем виде под продукцией понимают выражение вида A ® B. Обычное прочтение продукции выглядит так: ЕСЛИ A, ТО B. Импликация может истолковываться в обычном логическом смысле, как знак логического следования B из истинного A. Возможны и другие интерпретации продукции, например, A описывает некоторое условие, необходимое, чтобы можно было совершить действие B.

Продукционная модель *или модель, основанная на правилах,* позволяет представить знания в виде предложений типа

«**Если** <условие>, **то** <действие>».

Под *условием* понимается некоторое предложение – образец, по которому осуществляется поиск в базе знаний, а под *действием* – действия, выполняемые при успешном исходе поиска (они могут быть промежуточными, выступающими далее как условия, и терминальными или целевыми, завершающими работу системы).

При использовании продукционной модели база знаний состоит из набора правил. Программа, управляющая перебором правил, называется *машиной вывода*. Чаще всего вывод бывает прямой (от данных к поиску цели) или обратный (от цели для ее подтверждения – к данным). *Данные* – это исходные факты, на основании которых запускается машина вывода.

Если в памяти системы хранится некоторый набор продукций, то они образуют систему продукций. В системе продукций должны быть заданы специальные процедуры управления продукциями, с помощью которых происходит актуализация продукций и выполнение той или иной продукции из числа актуализированных.

В состав системы продукций входит база правил (продукций), глобальная база данных и система управления. *База правил* – это область памяти, которая содержит совокупность знаний в форме правил вида ЕСЛИ – ТО.

Глобальная база данных – область памяти, содержащая фактические данные (факты). Система управления формирует заключения, используя базу правил и базу данных. Существуют следующие способы формирования заключений – прямые и обратные выводы.

Правила вывода бывает удобно представлять в виде дерева решений (теория графов).

В прямых выводах выбирается один из элементов данных, содержащихся в базе данных, и если при сопоставлении этот элемент согласуется с левой частью правила (посылкой), то из правила выводится соответствующее заключение и помещается в базу данных или исполняется действие, определяемое правилом, и соответствующим образом изменяется содержимое базы данных.

В обратных выводах процесс начинается от поставленной цели. Если эта цель согласуется с правой частью правила (заключением), то посылка правила принимается за подцель или гипотезу. Этот процесс повторяется до тех пор, пока не будет получено совпадение подцели с данными.

При большом числе продукций в продукционной модели усложняется проверка непротиворечивости системы продукций, то есть множества правил. Поэтому число продукций, с которыми работают современные системы искусственного интеллекта, как правило, не превышает тысячи.

Продукционная модель привлекает разработчиков своей наглядностью, высокой модульностью, легкостью внесения дополнений и изменений и простотой механизма логического вывода.

Семантические модели знаний. В основе этих моделей лежит понятие сети, образованной помеченными вершинами и дуга-

ми. Вершины сети представляют некоторые сущности $V = \{v_1,\ldots, v_n\}$ (объекты, события, процессы, явления), а дуги – отношения между сущностями $R^v = \{r^v{}_1,\ldots, r^v{}_m\}$, которые они связывают $(v_l, r^v{}_i, v_j)$.

Наложив ограничения на описание вершин и дуг, можно получить сети различного вида. Если вершины не имеют собственной внутренней структуры, то соответствующие сети называют простыми сетями. Если вершины обладают некоторой структурой s_i $(v_i \rightarrow s_i(v_i))$, то такие сети называют иерархическими сетями. В настоящее время в большинстве приложений, использующих семантические сети, они являются иерархическими.

Одно из основных отличий иерархических семантических сетей от простых семантических сетей состоит в возможности разделить сеть на подсети (пространства) p_i $(P = \{p_1,\ldots, p_g\}$, $p_i \subseteq V$, $\bigcap_{i=1}^{g} p_i = \varnothing)$ и устанавливать отношения не только между вершинами R^v, но и между пространствами $R^p = \{r^p{}_1,\ldots, r^p{}_z\}$ $(p_i, r^p{}_k, p_j)$. Все вершины и дуги являются элементами, по крайней мере, одного пространства. Отметим, что понятие пространства аналогично понятию скобок в математической нотации. Различные пространства, существующие в сети, могут быть упорядочены в виде дерева пространств, вершинам которого соответствуют пространства, а дугам – отношения видимости (R^p). Например, из пространства пространство-потомок видимы все вершины и дуги, лежащие в пространствах-предках, а остальные пространства невидимы. Отношение видимости позволяет сгруппировать пространства в упорядоченные множества – *перспективы*. Перспектива обычно используется для ограничения сетевых сущностей, видимых некоторой процедурой, работающей с сетью, и *могут быть положены в основу визуализации моделей знаний по иерархическим уровням представления*. Обычно в перспективу включают не любые, а иерархические сгруппированные пространства. Исходя из изложенного, будем представлять семантические модели в виде четверки:

$$T_c = (V, R^v, P, R^p), \tag{1.3}$$

где V – сущности; R^v – отношения между сущностями; P – пространства; R^p – отношения видимости.

При необходимости в иерархических сетях можно представить любые логические связки и кванторы. Кроме представления логических связок и кванторов сеть может быть использована также для кодирования других структур высших порядков.

Объектно-ориентированные модели знаний. Наиболее развиваемым способом представления знаний при разработке программных систем является объектно-ориентированная парадигма. Этот подход является развитием фреймового представления. В его основе лежат понятия объект v_i, $V = \{v_1,\ldots, v_n\}$ и класс k_j, $K = \{k_1,\ldots, k_m\}$. Каждый объект является представителем некоторого класса однотипных объектов $v_i \in k_j$. Класс определяет общие свойства $k_i = (x_1,\ldots, x_g)$ для всех его объектов. Объекты и классы обладают характерными свойствами (инкапсуляция, наследование, полиморфизм и пр.), которые активно используются при объектно-ориентированном подходе и во многом определяют его преимущества. Функционирование системы рассматривается как взаимодействие объектов, с которыми ассоциируются, определенные правила R^v. Иерархический характер сложной системы отражается в виде иерархии классов $k^r = \bigcup_{i\in(r)} k_i^{r+1}$, r – ранг иерархии. Правила иерархической классификации R^k базируются на использования префикса ссылки на класс, к которому данное правило применимо. Указанный префикс с точки зрения декларативного представления знаний семантически подобен квантору всеобщности в исчислении предикатов. Таким образом, объектно-ориентированную модель можно описать как зависимость:

$$T_o = (V, R^v, K, R^k), \tag{1.4}$$

где V – объекты; R^v – отношения объектов; K – классы; R^k – правила классификации.

Если сравнить вид и содержание формул (1.1) – (1.4), то можно сделать вывод о возможности общего канонического представления логических моделей представления знаний в виде:

$$T = ((V, R^v), (P, R^p)), \tag{1.5}$$

где (V, R^v), – синтаксическая микромодель: V – алфавит, R^v – язык формирования элементов модели; (P, R^p) – семантическая микро-

модель: P – элементы модели, R^p – правила задания семантических отношений.

Однако такое представление формальных систем не учитывает влияние фактора количественного изменения знаний на составляющие. Стремление устранить недостатки формальных систем при их использовании в качестве моделей представления привело к появлению семиотических систем:

$$T = ((V, R^v), (P, R^p), Z(V, R^V, P, R^p)). \quad (1.6)$$

Семиотическая система формально задается как логическая модель, первые четыре компонента которой те же, что и в определении формальной системы, а $Z(V, R^V, P, R^p)$ – правила изменения первых четырех компонентов под влиянием накапливаемого опыта о строении и функционировании сущностей в данной проблемной среде.

Основываясь на вышеизложенном материале, процедурную мета-модель знаний будем представлять как формальную модель в виде композиции трех моделей:

$$T = (A, B, C), \quad (1.7)$$

где A – синтаксическая модель (V, R^v); B – семантическая модель (P, R^p); C – прагматическая модель $Z(V, R^V, P, R^p)$.

Вопросы по материалам первой главы:

1. Основные термины и определения ИС.
2. Типовые задачи и задачи, решаемые методами ИС.
3. Научно-технических направлений и научных школ по искусственному интеллекту и интеллектуальным системам.
4. Проблематика и основные направления развития областей искусственного интеллекта.
5. Данные и знания, их преемственность.
6. Базы данных и базы знаний.
7. Пять отличительных свойств знаний.
8. Определение знания с позиции формальных семиотических систем.
9. Декларативные модели данных и знаний.
10. Процедурные модели знаний.
11. Модели представления знаний на базе логики предикатов.
12. Вывод в логических моделях: метод резолюций, метод правил вывода.
13. Продукционные модели знаний.
14. Семиотические системы.

Глава 2.
ЭКСПЕРТНЫЕ СИСТЕМЫ

В течение последних десятилетий в рамках исследований по искусственному интеллекту сформировалось самостоятельное направление – *экспертные системы* (ЭС). Основной целью построения экспертных систем являются выявление, исследование и применение знаний высококвалифицированных экспертов для решения сложных задач, возникающих на практике [15, 17, 23, 38]. При построении таких систем, 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.

Основными задачами этого направления являются исследование и разработка программ (устройств), использующих знания и процедуры вывода для решения задач моделирующих людей-экспертов. В отличие от специализированных систем ИИ экспертные системы могут быть отнесены к системам ИИ общего назначения – системам, которые не только исполняют заданные процедуры, но на основе мета-процедур поиска генерируют и используют процедуры решения новых конкретных задач [14, 15, 44, 51].

Экспертные системы – это наиболее распространенный класс ИС, ориентированный на тиражирование опыта высококвалифицированных специалистов в областях, где качество принятия решений традиционно зависит от уровня экспертизы, например таких, как геология, медицина, юриспруденция, экономика, военное дело и др.

Огромный интерес к ЭС со стороны пользователей вызван, по крайней мере, тремя причинами:

– во-первых, они ориентированы на решение широкого круга задач в неформализованных областях, на приложениях, которые до недавнего времени считались малодоступными для вычислительной техники.

– во-вторых, с помощью ЭС специалисты, не знающие программирования, могут самостоятельно разрабатывать интересую-

щие их приложения, что позволяет резко расширить сферу использования вычислительной техники.

– в-третьих, ЭС при решении практических задач достигают результатов, которые не уступают, а иногда и превосходят возможности людей-экспертов, не оснащенных ЭВМ.

Широкое распространение ЭС получили в медицине, проектировании интегральных микросхем, в поиске неисправностей, в военных приложениях и автоматизации программирования. В последнее время ведутся разработки ЭС для следующих приложений: раннее предупреждение национальных и международных конфликтов и поиск компромиссных решений; принятие решений в кризисных ситуациях; охрана правопорядка; образование; планирование и распределение ресурсов; система организационного управления (кабинет министров, муниципалитет, учреждение) и т.д.

2.1. Структура ЭС

Знания, которыми обладает специалист в какой-либо области (дисциплине), можно разделить на формализованные (точные) и неформализованные (неточные). *Формализованные знания* формируются в книгах и руководствах в виде общих и строгих суждений (законов, формул, моделей, алгоритмов и т.п.), отражающих универсальные знания. *Неформализованные знания,* как правило, не попадают в книги и руководства в связи с их конкретностью, субъективностью и приблизительностью. Знания этого рода являются результатом обобщения многолетнего опыта работы и интуиции специалистов. Они обычно представляют многообразие эмпирических (эвристических) приемов и правил.

В зависимости от того, какие знания преобладают в той или иной области (дисциплине), ее относят к формализованным (если преобладают точные знания) или к неформализованным (если преобладают неточные знания) описательным областям. Задачи, решаемые на основе точных знаний, называют формализованными, а задачи, решаемые с помощью неточных знаний – неформализованными. (Речь идет не о неформализуемых, а о неформализованных задачах, то есть о задачах, которые, возможно, и формализуемы, но эта формализация пока неизвестна.)

Традиционное программирование в качестве основы для разработки программы использует алгоритм, то есть формализован-

ное значение. Поэтому до недавнего времени считалось, что ЭВМ не приспособлены для решения неформализованных задач. Расширение сферы использования ЭВМ показало, что неформализованные задачи составляют очень важный класс задач, вероятно, значительно больший, чем класс формализованных задач. Неумение решать неформализованные задачи сдерживает внедрение ЭВМ в описательные науки. По мнению авторитетов, основной задачей информатики является внедрение ее методов в *описательные* науки и дисциплины. На основании этого можно утверждать, что исследования в области ЭС занимают значительное место в информатике.

К неформализованным задачам относятся те, которые обладают одной или несколькими из следующих особенностей:

– алгоритмическое решение задачи неизвестно (хотя, возможно, и существует) или не может быть использовано из-за ограниченности ресурсов ЭВМ (времени, памяти);

– задача не может быть определена в числовой форме (требуется символьное представление);

– цели задачи не могут быть выражены в терминах точно определенной целевой функции.

Как правило, неформализованные задачи обладают неполнотой, ошибочностью, неоднозначностью и/или противоречивостью знаний (как данных, так и используемых правил преобразования).

Экспертные системы не отвергают и не заменяют традиционного подхода к автоматизированным информационным системам, они отличаются от традиционных программных систем тем, что ориентированы на решение неформализованных задач и обладают следующими особенностями:

– алгоритм решения не известен заранее, а строится самой ЭС с помощью символических рассуждений, базирующихся на эвристических приемах;

– ясность полученных решений, то есть система "осознает" в терминах пользователя, как она получила решение;

– способность анализа и объяснения своих действий и знаний;

– способность приобретения новых знаний от пользователя-эксперта, не знающего программирования, и изменения в соответствии с ними своего поведения (открытая система);

– обеспечение «дружественного», как правило, естественно-языкового (ЕЯ) интерфейса с пользователем.

Обычно к ЭС относят *системы, основанные на знании*, то есть системы, вычислительная возможность которых является в первую очередь следствием их наращиваемой базы знаний (БЗ) и только во вторую очередь определяется используемыми методами. *Методы инженерии знаний* (методы ЭС) в значительной степени инвариантны тому, в каких областях они могут применяться. В настоящее время ЭС используются при решении задач следующих типов: принятие решений в условиях неопределенности (неполноты информации), интерпретации символов и сигналов, предсказание, диагностика, конструирование, планирование, управление, контроль и др.

Типичная ЭС состоит из следующих основных компонентов (рис. 2.1): решателя (интерпретатора), рабочей памяти (РП), называемой также базой данных (БД), базы знаний (БЗ), компонентов приобретения знаний, объяснительного и диалогового.

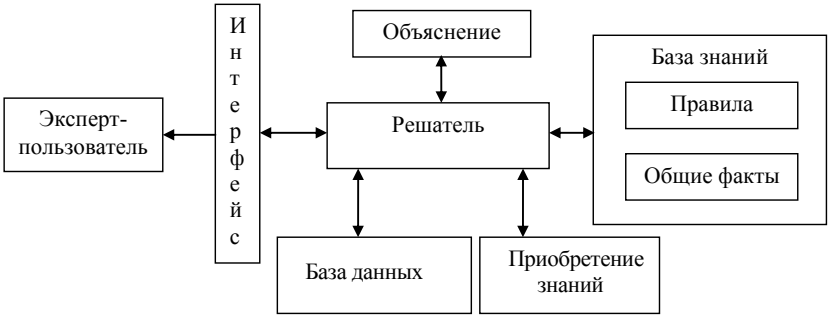


Рис. 2.1. Схема обобщенной экспертной системы

База данных предназначена для хранения исходных и промежуточных данных решаемой в текущей момент задачи.

База знаний в ЭС предназначена для хранения долгосрочных данных (фактов), описывающих рассматриваемую область, и правил, описывающие преобразования (выводы) этой области.

Решатель (интерпретатор), используя исходные данные из БД и знания из БЗ, формирует такую последовательность правил, которые, будучи примененными к исходным данным, приводят к решению задачи.

Компонента приобретения знаний автоматизирует процесс наполнения ЭС знаниями, осуществляемый пользователем-экспертом.

Объяснительная компонента объясняет, как система получила решение задачи (или почему она не получила решения) и какие знания при этом она использовала, что облегчает эксперту тестирование системы и повышает доверие пользователя к полученному результату.

Интерфейс необходим для формирования понятийного, человеко-ориентированного взаимодействия (диалога) со всеми категориями пользователей как в ходе решения задач, так и приобретения знаний, объяснения результатов работы.

В разработке ЭС участвуют представители следующих специальностей:

– эксперт в той проблемной области, задачи которой будет решать ЭС;

– инженер по знаниям (когнитолог) – специалист по разработке ЭС;

– программист – специалист по разработке инструментальных средств.

Эксперт определяет знания (данные и правила), характеризующие проблемную область, обеспечивает полноту и правильность введения в ЭС знаний.

Инженер по знаниям (когнитолог) помогает эксперту выявить и структурировать знания, необходимые для работы ЭС. Он выбирает тот инструментарий, который наиболее подходит для данной проблемной области, и определяет способ представления знаний в нем, выделяет и программирует (традиционными средствами) стандартные функции (типичные для данной проблемной области), которые будут использоваться в правилах, вводимых экспертом.

Уже при разработке первых ЭС стало очевидно, что наиболее ответственным этапом является построение БЗ, для чего в общем случае и необходим посредник – так называемый *инженер по знаниям*, который должен обеспечить проведение до машинных этапов разработки СОЗ, заключающихся, как правило, в анализе предметной области, извлечении знаний из эксперта и их структурировании.

Эти процедуры оказались самыми тяжелыми [1, 6], поскольку, с одной стороны, чрезвычайно высок уровень требований, предъявляемых к личности инженера по знаниям (высококвалифицированный специалист в вычислительной науке, обладающий способностями к контакту с экспертами, умеющий побудить эксперта поставлять нужную информацию, умеющий отделять главное от второстепенного и т.д.), а с другой – стали наблюдаться трудности с поисками собственно экспертов (например, эксперт испытывает затруднения с четкой формулировкой своих знаний, не всегда расположен полностью делиться знаниями и др.).

Поэтому почти одновременно с появлением индустрии знаний стали разрабатываться автономные системы, автоматизирующие процессы получения необходимой информации от экспертов. Подобные программные средства получили название «оболочек приобретения».

Программист разрабатывает инструментальное средство, содержащее в пределе все основные компоненты ЭС, осуществляет сопряжение с той средой, в которой оно может быть использовано.

2.2. Режимы работы ЭС

Экспертная система работает в двух режимах:

– приобретения знаний;

– решения задач (называемом также режимом консультации или режимом использования ЭС).

В режиме приобретения знаний общение с ЭС осуществляет эксперт через посредничество инженера по знаниям. Эксперт описывает проблемную область в виде совокупности данных и правил. Данные определяют объекты, их характеристики и значения, существующие в области экспертизы. Правила определяют способы манипулирования данными, характерные для рассматриваемой проблемной области. Эксперт, используя компонент приобретения знаний, наполняет систему знаниями, которые позволяют ЭС в режиме решения самостоятельно (без эксперта) решать задачи из проблемной области.

Важную роль в режиме приобретения знаний играет объяснительный компонент. Именно благодаря ему эксперт на этапе тестирования локализует причины неудачной работы ЭС, что позво-

ляет эксперту целенаправленно моделировать старые или вводить новые знания. Обычно, объяснительный компонент сообщает следующее: как правильно использовать информацию пользователя; почему использовались или не использовались данные или правила; какие были сделаны выводы и т.п. Все объяснения делаются, как правило, на ограниченном естественном языке или языке графики. Отметим, что режиму приобретения знаний при традиционном подходе к разработке программ соответствуют этапы алгоритмизации, программирования и отладки, выполняемые программистом. Таким образом, в отличие от традиционного подхода разработку программ осуществляет эксперт (с помощью ЭС), не владеющий программированием, а не программист.

В режиме консультации общение с ЭС осуществляет конечный пользователь, которого интересует результат и/или способ получения решения. Пользователь в зависимости от назначения ЭС может не быть специалистом в данной проблемной области, в этом случае он обращается к ЭС за советом, не умея получить ответ сам, или быть специалистом, в этом случае он обращается к ЭС, чтобы либо ускорить процесс получения результата, либо возложить на ЭС рутинную работу. Термин "пользователь" является многозначным, так как кроме конечного пользователя применять ЭС может и эксперт, и инженер по знаниям, и программист. Поэтому, когда хотят подчеркнуть, что речь идет о том, для кого делалась ЭС, используют термин "конечный пользователь".

В режиме консультации данные о задаче пользователя обрабатываются диалоговой компонентой, которая выполняет следующие действия: распределяет роли участников (пользователя и ЭС) и организует их взаимодействие в процессе кооперативного решения задачи; преобразует данные пользователя о задаче, представленные на первичном для пользователя языке, во внутренний язык системы; преобразует сообщения системы, представленные на внутреннем языке в сообщения на языке, привычном для пользователя (обычно это ограниченный естественный язык или язык графики).

После обработки данные поступают в РП. На основе входных данных из РП, общих данных о проблемной области и правил из БЗ решатель (интерпретатор) формирует решение задачи.

В отличие от традиционных программ ЭС в режиме задачи не только исполняет предписанную последовательность операций, но и предварительно формирует ее. Если ответ ЭС не понятен пользователю, то он может потребовать объяснения, как ответ получен.

2.3. Классификация экспертных систем

Экспертные системы как любой сложный объект можно определить только совокупностью характеристик. Выделим следующие характеристики ЭС:

1. Назначение.
2. Проблемная область.
3. Глубина анализа проблемной области.
4. Тип используемых методов и знаний.
5. Инструментальные средства.
6. Стадия существования.
7. Класс системы.

Перечисленный набор характеристик не претендует на полноту (в связи с отсутствием общепринятой классификации), а определяет ЭС как целое, не выделяя отдельных компонентов (способ представления знаний, решения задач и т.п.).

Назначение ЭС определяется следующей совокупностью параметров: цель создания ЭС – для обучения специалистов, для решения задач, для автоматизации рутинных работ, для тиражирования знаний экспертов и т.п.; основной пользователь – не специалист в области экспертизы, специалист, учащийся.

Проблемная область. Проблемная область может быть определена совокупностью параметров: предметной областью и задачами, решаемыми в предметной области, каждый из которых может рассматриваться с точки зрения как конечного пользователя, так и разработчика ЭС.

С точки зрения пользователя, предметную область можно характеризовать описанием области в терминах пользователя, включающим наименование области, перечень и взаимоотношение подобластей и т.п., а задачи, решаемые существующими ЭС, – их типом. Обычно выделяют следующие *типы задач:*

– интерпретация символов или сигналов – составление смыслового описания по входным данным;

– предсказание – определение последствий наблюдаемых ситуаций;

– диагностика – определение состояния неисправностей, заболеваний по признакам (симптомам);

– конструирование – разработка объекта с заданными свойствами при соблюдении установленных ограничений;

– планирование – определение последовательности действий, приводящих к желаемому состоянию объекта;

– слежение – наблюдение за изменяющимся состоянием объекта и сравнение его показателей с установленными или желаемыми;

– управление – воздействие на объект для достижения желаемого поведения.

С точки зрения разработчика, целесообразно выделять *статические* и *динамические предметные области.* Предметная область называется статической, если описывающие ее исходные данные не изменяются во времени (точнее рассматриваются как не изменяющиеся за время решения задачи). Статичность области означает неизменность описывающих ее исходных данных. Если исходные данные, описывающие предметную область, изменяются за время решения задачи, то предметную область называют динамической. Кроме того, предметные области можно характеризовать следующими аспектами: числом и сложностью сущностей; атрибутами и значениями атрибутов; связностью сущностей и их атрибутов; полнотой знаний; точностью знаний (знания точны или правдоподобны; правдоподобность знаний представляется некоторым числом или высказыванием).

Решаемые задачи, с точки зрения разработчика ЭС, также можно разделить на статические и динамические. Будем говорить, что ЭС решают *динамическую* или *статическую* задачу, если процесс решения задачи изменяет или не изменяет исходные данные о текущем состоянии предметной области.

В подавляющем большинстве существующих ЭС исходят из предположения статичности предметной области и решают статические задачи, будем называть такие ЭС *статическими.* ЭС, которые имеют дело с динамическими предметными областями и решают статистические или динамические задачи, будем называть

их *динамическими.* В последние годы стали появляться первые динамические ЭС. Видимо, решение многих важнейших практических неформализованных задач возможно только с помощью динамических, а не статических ЭС. Следует подчеркнуть, что на традиционных (числовых) последовательных ЭВМ с помощью существующих методов инженерии знаний можно решать только статические задачи, а для решения динамических задач, составляющих большинство реальных приложений, необходимо использовать специализированные символьные ЭВМ.

Решаемые задачи, кроме того, можно характеризовать следующими аспектами: числом и сложностью правил, используемых в задаче; связностью правил; пространством поиска; количеством активных агентов, изменяющих предметную область; классом решаемых задач.

По степени сложности *выделяют простые* и *сложные правила.* К сложным правилам относят правила, текст знаний которых на естественном языке занимает 1/3 страницы и больше. Правила, текст которых занимает менее 1/3 страницы относят к простым.

По степени связности правил, задачи делятся на связные и мало связные. К связным относят задачи (подзадачи), которые не удается разбить на независимые задачи. Мало связные задачи удается разбить на некоторое количество независимых подзадач. Можно сказать, что степень сложности определяется не просто общим количеством правил данной задачи, а количеством правил в ее наиболее связной независимой подзадаче.

Пространство поиска может быть определено, по крайней мере, тремя подаспектами: размером, глубиной и шириной. Размер пространства поиска дает обобщенную характеристику сложности задачи. Выделяют малые (до 10! состояний) и большие (свыше 10! состояний) пространства поиска. Глубина пространства поиска характеризуется средним числом последовательно применяемых правил, преобразующих исходные данные в конечный результат, ширина пространства – средним числом правил, пригодных к выполнению в текущем состоянии.

Класс решаемых задач характеризует методы, используемые ЭС для решения задачи. Данный аспект в существующих ЭС применяет следующие значения: задачи расширения, доопределения,

преобразования. Задачи расширения и доопределения являются статическими, а задачи преобразования – динамическими.

К задачам расширения относятся задачи, в процессе решения которых осуществляется только увеличение информации о предметной области, не приводящие ни к изменению ранее выведенных данных, ни к выбору другого состояния области. Типичной задачей этого класса являются задачи классификации.

К задачам доопределения относятся задачи с неполной или неточной информацией о реальной предметной области, цель решения которых – выбор из множества альтернативных текущих состояний предметной области того, которое адекватно исходным данным. В случае неточных данных альтернативные текущие состояния возникают как результат ненадежности данных и правил, что приводит к многообразию различных доступных выводов из одних и тех же исходных данных. В случае неполных данных альтернативные состояния являются результатом до определения предметной области, то есть результатом предположений о возможных значениях недостающих данных.

К задачам преобразования относятся задачи, которые осуществляют изменения исходной или выведенной ранее информации о предметной области, являющиеся следствием изменений либо реального мира, либо его модели.

Большинство существующих ЭС решают задачи расширения, в которых нет ни изменений предметной области, ни активных агентов, преобразующих предметную область. Подобное ограничение неприемлемо при работе в динамических областях.

Глубина анализа проблемной области. По степени сложности структуры ЭС делят на поверхностные и глубинные. *Поверхностные* ЭС представляют знания об области экспертизы в виде правил (условие → действие). Условие полного правила определяет образец некоторой ситуации, при соблюдении которой правило может быть выполнено. Поиск решения состоит в выполнении тех правил, образцы которых сопоставляются с текущими данными (текущей ситуации в РП). *Глубинные* ЭС, кроме возможностей поверхностных систем, обладают способностью при возникновении неизвестной ситуации определять с помощью некоторых общих принципов, справедливых для области экспертизы, какие действия следует выполнять.

Тип используемых методов представления и знаний. По типу используемых методов и знаний ЭС делят на традиционные и гибридные. *Традиционные* ЭС используют в основном неформализованные методы инженерных знаний и неформализованные знания, полученные от экспертов. *Гибридные* ЭС используют и методы инженерии знаний и формализованные методы, а также данные традиционного программирования и математики.

Сейчас говорят о трех поколениях ЭС. К *первому поколению* следует относить статические поверхностные ЭС, ко *второму* - статические глубинные ЭС (иногда ко второму поколению относят гибридные ЭС), а к *третьему* – динамические ЭС (вероятно, они, как правило, будут глубинными и гибридными).

Инструментальные средства. Большинство *инструментальных средств (ИНС)* предназначено для создания прототипов ЭС, решающих статические задачи (обычно задачи расширения) в статических проблемных областях. По степени применимости ИНС выделяют следующие стадии существования: исследовательская, промышленная, коммерческая. Разделяют следующие типы ИНС:

- языки программирования;
- языки инженерии знаний;
- средства автоматизации разработки (проектирования) ЭС;
- оболочки ЭС.

С точки зрения потребителя, на выбор ИС влияют моменты:

- затраты труда на построение ЭС или ее прототипа с помощью ИНС;
- эффективность функционирования ЭС, построенной на основе выбранного ИНС;
- квалификация разработчика, необходимая для применения ИНС.

Оболочки ЭС ориентированы на работу с пользователем – непрофессионалом в области программирования. Основным свойством оболочек является то, что они содержат все компоненты ЭС в готовом виде и их использование не предполагает программирования, а сводится лишь к вводу в оболочку знаний о проблемной области. Каждая оболочка характеризуется фиксированным способом представления знаний и организации вывода и фиксирования компонентов, которые будут использоваться во всех положениях,

где будет применяться оболочка. Наиболее популярные оболочки обладают следующими свойствами:

1. Решают задачи класса расширения в статических предметных областях в условиях ненадежности знаний.

2. Представляют процедурные знания в виде правил.

3. Описывают предметную область в виде значений неструктурированных переменных и утверждений, снабженных мерой их истинности (определенности).

Желание представить разработчику ЭС разнообразные средства для учета особенностей приложения, привело к объединению в рамках одной системы различных методов решения задач, представления и интерпретации знаний. В их состав могут входить средства модификации функционирования оболочки, набор компонентов, позволяющих конструировать собственные оболочки, средства комплексирования компонентов в виде языка высокого уровня, развитые интерактивные графические средства общения с пользователем. Подобные средства называют средствами автоматизации проектирования (разработки) ЭС.

Характеристика «*Универсальность»* определяет возможности ЭС в использовании различных способах представления знаний в рабочей памяти и базе знаний и различных парадигм функционирования системы. Наличие универсальности позволяет адекватно отображать в системе различные типы знаний о проблемной области. К настоящему времени в большинстве ЭС при представлении знаний используют фреймы и сети, а в качестве механизма функционирования, как правило, программирование, ориентированное на правила.

Характеристика «*Основные свойства*» определяет особенности, которые присущи инструментарию ЭС в реализации основных программных компонентов системы. Для решателя наиболее важны способы сопоставления и основной способ планирования вычисления (построение цепочек вывода от данных или от целей). Задача сопоставления состоит в том, чтобы определить, какое из правил, хранящихся в БЗ, может быть применено к текущему состоянию предметной области, хранимой в РП. Способы сопоставления в значительной мере зависят от: типа ссылки на объекты РП, используемого в правиле; вида данных РП, сопоставляемых со ссылками; вида проверок, выполняемых в ходе сопоставления.

Выделяют следующие типы ссылок:

– конкретная, когда ссылка (идентификатор) в условии правила является адресом конкретного элемента РП;

– абстрактная, когда ссылка (идентификатор и его описание) в условии правила именует не конкретный, а любой элемент в РП, свойства которого сопоставляются с описанием ссылки, указанным в правиле (то есть описание определяет класс элементов РП).

Существующие инструментальные средства ЭС допускают следующие виды данных РП: константы; переменные, имеющие значения; сложные структуры (типы фрейм), логически объединяющее множество переменных. В них используются либо тривиальные виды проверок, сводящиеся к проверке наличия (отсутствия) указанных элементов в РП, либо сложные, требующие вычисления некоторых соотношений между значениями ссылок, указанных в условиях правил. В первом приближении способ сопоставления определяется используемым типом ссылок, видом данных РП к видам проверок.

Средства приобретения знаний в существующих ЭС можно оценивать с точки зрения допустимых способов формирования БЗ. Выделяют следующие способы формирования БЗ:

- редакторы;
- средства отладки;
- средства индуктивного вывода новых знаний.

Редакторы позволяют отображать и модифицировать БЗ, возможно, в графическом виде, поддерживая ее целостность. Средства отладки обеспечивают анализ содержимого БЗ, переформирование и отображение его результатов пользователю. Средства индуктивного вывода осуществляют формирование новых знаний (правил) на основе вводимых пользователем примеров ситуаций с их решениями.

Класс ЭС. Как правило, выделяются два больших класса ЭС (существенно отличающихся по технологии их проектирования), которые условно можно назвать простыми и сложными ЭС. Простая ЭС может быть охарактеризована следующими значениями основных параметров: поверхностная ЭС; традиционная ЭС (реже гибридная); выполненная на персональной ЭВМ. Сложная ЭС может быть охарактеризована следующими значениями параметров:

глубинная ЭС; гибридная ЭС; выполненная либо на символьной ЭВМ, либо на мощной универсальной ЭВМ, либо на интеллектуальной рабочей станции.

Следует отметить, что единую классификацию всех существующих на сегодня ЭС провести достаточно сложно, так как, с одной стороны, можно выделить большое количество специфических характеристик ЭС, а с другой стороны – у разных авторов существуют значительные различия в терминологии обозначения одних и тех же вещей.

Предложим классификацию ЭС на основе следующих базовых параметров:

- уровень используемого языка;
- машина вывода (решатель);
- методы описания ПО;
- способ представления знаний;
- парадигма программирования и др.

Примеры классификации ЭС по указанным параметрам:

1. Классификация ЭС по парадигмам программирования (механизм реализации исполняемых утверждений):

- процедурное программирование;
- объектно-ориентированное программирование;
- программирование, ориентированное на данные;
- программирование, ориентированное на правила.

2. Классификация ЭС по способу представления знаний (характеризующемуся моделью представления знаний):

- в виде правил (продукций);
- в виде фреймов или объектов;
- в виде семантических сетей;
- логические модели представления знаний (исчисление предикатов).

3. Классификация ЭС по реализации различных способов рассуждений, принятыми в конкретных предметных областях:

- дедуктивный способ рассуждений;
- индуктивный способ рассуждений;
- способ рассуждений по аналогам или на основе прецедентов;
- способ рассуждений посредством выдвижения гипотез.

4. Классификация ЭС по уровню используемого языка:

– традиционные (в том числе объектно-ориентированные) языки программирования (C, C++ и др.);

– символьные языки программирования (LISP, Prolog и их разновидности);

– инструментарий, содержащий часть компонент ЭС (OPS-5, ИЛИС и др.);

– оболочки и среды разработки общего назначения, содержащие все компоненты ЭС (EMMYCIN, Leonardo, ЭКО, GURU, Nexpert Object, ProKappa, и др)

– проблемно-специализированные средства (ориентированные на некоторый класс решаемых задач);

– предметно-ориентированные средства (включающие знания о некоторых типах предметных областей).

5. Классификация ЭС по методам описания проблемных областей.

Подход на базе поверхностных знаний заключается в извлечении из эксперта фрагментов эвристических знаний о данной ПО, которые релевантные решаемой задаче, причем не предпринимается никаких попыток глубинного изучения области, что предопределяет использование поиска в пространстве состояний в качестве универсального механизма вывода. Как правило, этот подход применяется к задачам, которые не могут быть точно описаны, и в качестве способа представления знаний выбираются правила. Если же задача может быть заранее структурирована или при ее решении можно воспользоваться некоторой моделью, то такой подход неэффективен.

Структурированный подход используется в качестве развития поверхностного подхода в том случае, если применение поверхностного не обеспечивает решения задачи. Используя декомпозицию задачи на подзадачи (дерево подзадач), можно затем решать каждую задачу на основе поверхностного или глубинного подхода, а возможно, и их комбинации.

Глубинный подход. При использовании глубинного подхода к решению задачи качество и компетентность ЭС будут зависеть от модели ПО, причем эта модель может быть определена различными способами (декларативно, процедурно). При глубинном подходе используются ЭС с мощными моделирующими возмож-

ностями, а именно: объекты (фреймы) с присоединенными процедурами, иерархическое наследование свойств, активные объекты, механизмы передачи сообщений объектам и др. Если сравнить, описанные выше подходы, с типами ПО, то можно более детально классифицировать и оценить конкретные ЭС по данному параметру.

Подавляющее большинство совместимых статических ЭС ориентированы на реализацию дедуктивного способа рассуждений, причем акценты делаются на такие параметры логического вывода, как:

- структура процесса получения решения;
- методы поиска решения;
- стратегии разрешения конфликтов;
- управление достоверностью и др.

Конкретные значения этих параметров могут выступать как некоторые критерии оценки машины вывода.

Следует отметить, что из всего многообразия моделей в современных ЭС используются только правила и объекты (фреймы). Напомним также, что ЭС, имеющие в своем составе более двух моделей представления знаний, называются гибридными.

2.4. Формы представления знаний в ЭС

Экспертные системы представляют собой одно из направлений в области искусственного интеллекта. Эти системы используют большое количество знаний, передаваемых им специалистами, а с другой – способны вступать в диалог и объяснять свои собственные выводы. Это предполагает наличие эффективного управления большой по объему и хорошо структурированной базой знаний, строгое разграничение между различными уровнями знаний, наличие множества удобных представлений для правил, схем предикатов или прототипов и четко определенный процесс обмена информацией между различными источниками.

База знаний содержит знания в символьной форме. Точнее говоря, в нее могут входить также таблицы чисел, диапазоны значений величин и, где это требуется, некоторые вычислительные процедуры. Однако основное содержание БЗ – *факты* и *эвристики.*

В состав фактов БЗ входят описания объектов, их признаки и соответствующие числовые данные для области, в которой пред-

полагается применять ЭС. Например, для систем управления технологическими процессами в состав фактических знаний могут быть включены описания конкретного предприятия или его части, характеристики отдельных компонентов, значения, получаемые от датчиков, состав запасов и т.д.

Эвристики, или правила, представляют собой пути вынесения суждений на основании фактов для решения конкретной проблемы. Эти знания базируются на прошлом опыте; эксперты постоянно пользуются ими, но часто держат их в тайне. Инженерию знаний можно охарактеризовать как некий процесс, в ходе которого информация "добывается и обогащается" создателями экспертных систем. Для систем управления технологическим процессом эти знания могут содержать такой набор правил: когда требуется профилактический ремонт завода или подсистемы; каким должен быть объем запасов, исходя из текущих цен; правила диагностики неисправностей и рекомендации по их устранению и др.

Ценность ЭС для коллективов специалистов, начиная от лабораторий и кончая целыми фирмами, проявляется в нескольких аспектах:

1) сбор, уточнение, кодирование и распространение экспертных знаний («Эксперт под рукой»);

2) решение проблем, сложность которых превышает человеческие возможности;

3) решение проблем, требующих объема знаний, которого один человек не в состоянии охватить;

4) решение проблем, для которых требуется экспертные знания из нескольких областей («сплав» знаний);

5) сохранение наиболее уязвимой ценности коллектива – коллективной памяти;

6) обеспечение высокой конкурентоспособности за счет применения новой технологии.

Создание *комплексных* баз знаний открывает широкие возможности, которые обусловлены безошибочностью и тщательностью, присущими машине, и синтезом знаний нескольких экспертов. Если же БЗ объединяет информацию по нескольким дисциплинам, такой "сплав" знаний приобретает дополнительную ценность.

Типы знаний в ЭС. Выделим восемь основных типов знаний по следующим признакам:

1. *Базовые элементы, объекты реального мира.* Они связаны с непосредственным восприятием, не требуют обсуждения и добавляются к нашей базе фактов в том виде, в котором они получены.

2. *Утверждения и определения.* Они основаны на базовых элементах и заранее рассматриваются как достоверные.

3. *Концепции.* Они определяют собой перегруппировки или обобщения базовых объектов. Для построения каждой концепции используются свои приемы.

4. *Отношения.* Они выражают как элементарные свойства базовых элементов, так и отношения между концепциями. Кроме того, к свойствам отношений относятся их большее или меньшее правдоподобие, большая или меньшая связь с данной ситуацией. Еще раз отметим, что представление знаний в ЭС близко к моделям, используемым в БД. Таким путем построена реляционная (обобщенная) модель БД в некоторых ЭС. Пара понятий "свойство–значение" хорошо известна в семантических сетях; фреймы и скрипты являются ничем иным, как наиболее простыми бинарными отношениями. Некоторые ЭС в качестве базы фактов используют уже базы существующих данных.

5. *Теоремы и правила перезаписи.* Они являются частными случаем продукционных правил с вполне определенными свойствами. Теоремы не представляют никакой пользы без экспертных правил их использования. Явное присутствие теорем ЭС представляет главное отличие от систем управления классическими базами данных (СУБД), в которых они либо отсутствуют, либо программируются. Модификация или добавление новых теорем является весьма трудоемкой, хотя и необходимой процедурой, так как нужно обеспечить хорошее структурированное управление БД и оптимизировать получение ответов.

6. *Алгоритмы решений.* Они необходимы для выполнения определенных задач. Во всех случаях они связаны со знанием особого типа, поскольку определяемая ими последовательность действий оказывается оформленной в блок, в строго необходимом порядке в отличие от других типов знания, где элементы информации могут появляться и располагаться без связи друг с другом.

Очевидно, что очень трудно работать с длинными процедурами, состоящими из большого числа различных действий. Использование чистых алгоритмов ограничено очень частными случаями,

большая часть которых имеет дело с обработкой числовой информации. ЭВМ должна рассматривать и неалгоритмические ситуации.

7. *Стратегии и эвристика.* Этот тип представляет врожденные или приобретенные правила поведения, которые позволяют в данной конкретной ситуации принять решение о необходимых действиях. Он использует информацию в порядке, обратном тому, в котором она была получена. Например, рассуждения типа: «Я знал, что это действие приводит к такому-то результату (информация типа в п. 4), поэтому, если я хочу получить именно этот результат, я могу рассмотреть это действие». Человек постоянно пользуется этим типом знаний при восприятии, формировании концепций, решении задач и формальных рассуждениях. Появление ЭС связано с необходимостью принятия в расчет именно этого фундаментального типа человеческих знаний.

8. *Метазнание.* Без сомнения оно присутствует на многих уровнях и представляет собой знание того, что известно и определяет значение коэффициента к этому знанию, важность элементарной информации по отношению ко всему множеству знаний. Кроме того, сюда же относятся вопросы организации каждого типа знаний и указаний, когда и как они могут быть использованы. Метазнание представляет собой любое знание о знании. Оно является фундаментальным понятием для систем, которые не только используют свою БЗ такой, какая она есть, но и умеют на ее основе делать выводы, структурировать ее, абстрагировать, обобщать, а также решать, в каких случаях она может быть полезна.

Приведем краткий список наиболее распространенных способов (методов) представления знаний в соответствии с современным производственным сленгом программно-информационных систем.

1. Конечный автомат.
2. Программа.
3. Скрипт (схема).
4. Семантическая связь.
5. Фрейм (прототип).
6. Графы, сети.
7. Формальная спецификация.
8. Исчисление предикатов.
9. Теоремы, правила перезаписи.
10. Продукционные правила.

11. Предложения на языке.

Следует отметить, что конечные автоматы, программы, исчисление предикатов и системы с продукционными правилами с теоретической точки зрения эквивалентны друг другу, поскольку их можно свести к универсальной машине Тьюринга, хотя их "эксплуатационные характеристики» сильно различаются.

Такие формальные понятия как фреймы, скрипты, семантические сети, возникли из реальных потребностей ИИ и приносят большую помощь в понимании языка.

Фреймы (или прототипы) предложены Минским в 1975 г. и представляют собой сложные структуры данных, описывающих какую-либо типичную ситуацию. Фрейм состоит из позиций (слотов) для размещения объектов, характеризующих данную ситуацию. Позиция может быть передана другому фрейму.

Скрипты, или схема, представляет собой описание стереотипного сценария действий с участием определенных объектов. Они могут вызывать другие скрипты и обладают большими, чем фреймы, возможностями для описания динамических аспектов знания.

Семантические сети – это графы, которые часто представляют собой объединение двух предыдущих понятий и объектом описания которых являются элементы окружающего мира и связи между ними.

При формализации предметной области при реализации конкретных ЭС возникает задача тестирования и отбора специалистов, которые могли бы выступить в роли экспертов. Для этих целей целесообразно создавать специальный блок, который формировал бы модель «идеального» эксперта (личностного портрета) в виде набора критериев, которым должен удовлетворять человек, претендующий на роль эксперта по решению конкретного класса задач.

Системы поддержки знаний. Если рассматривать ЭС с информационной точки зрения, то ее структуру можно представить состоящую из следующих компонент:

База знаний. БЗ фактов и правил вывода является центральной частью ЭС. В отличие от обычной информационной БД в ней хранятся не только факты, но и правила, позволяющие вывести новые факты.

Операторы. Три класса операторов ЭС:

– *пользователи* обращаются к системе за советом по специальным проблемам в узкой области, представляя ей специфические факты и свои гипотезы о следствиях или целях;

– *эксперты* обращаются к системе, чтобы передать ей свои знания по частной проблеме, а также общепринятые факты и процедуры вывода;

– *инженеры* действуют как промежуточные звенья между экспертом и системой, помогая первому зондировать свои знания и проверяя работу законченной ЭС.

Оболочка ЭС представляет собой программную систему, используемую для приложения БЗ с целью выработки решения задачи:

– прикладная система проверяет гипотезы пользователя или ищет пути достижения поставленных целей путем вывода последовательности фактов о конкретной ситуации со слов пользователя, а также используя общие факты и правила вывода, представленные экспертом и введенные в ЭС инженером по знаниям;

– система обоснования отвечает на вопросы о том, каким способом были выведены факты, на основе какой информации и каких правил вывода;

– система приобретения служит для опроса эксперта с целью получения его словаря, общих фактов и правил вывода;

– система отображения обеспечивает необходимыми средствами для представления БЗ в понятной форме, а также связей между фактами и правилами вывода, содержащимися в БЗ;

– система редактирования содержит инструментарий для компоновки и редактирования БЗ, сохраняя при этом ее целостность при помощи словаря, переменных и применяемых операций;

– система проверки содержит инструментарий для испытаний и предохранения БЗ от специфических историй с известными последствиями.

Информационные оболочки большинства современных ЭС:

– предлагают определенную БЗ, представленную посредством правил, продукций и фреймов;

– имеют некоторые прикладные возможности, ограниченные в выводе следствий и испытании утверждений;

– дают простые объяснения посредством используемых фактов и правил;

– не имеют способностей интегрированного сбора знаний;

– имеют ограниченные возможности по отображению БЗ, в частности, в графической форме;

– имеют такой интерфейс пользователя с редактором БЗ, который требует понимания внутренних структур данных;

– представляют ограниченные контролирующие возможности и не имеют объединенной БД, фактических историй.

Однако в последние годы происходят быстрые положительные изменения в сторону улучшения методов представления знания, получения выводов, их обоснования, отображения и обучения ЭС и все это благодаря интенсивному развитию инженерии знаний.

Проблемы *инженерии знаний* можно сформулировать следующим образом. Приобретение знания является узким местом в построении ЭС. Инженер по знаниям имеет меньшее знание области, чем эксперт, возникающие проблемы связи препятствуют процессу превращения экспертизы в программу. Словарь, используемый экспертом при разговоре с новичком о предметной области, часто не годится для решения проблемы; таким образом, инженер по знаниям и эксперт должны работать вместе, чтобы обновить и расширить его. Один из наиболее трудных аспектов деятельности инженера по знаниям заключается в оказании помощи эксперту в разработке структуры знания предметной области, в определении и формализации основных концепций этой области.

Это узкое место является главным препятствием на пути применения ЭС и реализации их полного рабочего потенциала. Автоматизация приобретения и передачи знания стала главной целью многих исследований. Уменьшение стоимости аппаратных и программных средств поддержки ЭС вывело рассматриваемую технологию инженерии знаний на массовый спрос гораздо раньше, чем ожидалось.

В то же время движение в направлении интенсификации человеческого труда, которое наблюдается в инженерии знаний, противоречит основным тенденциям современной промышленности.

Все это усилило интерес к разработкам систем поддержки знания, способным автоматизировать инженерию знания как процесс прямого воздействия между экспертами и ЭВМ.

Очевидным подходом является анализ беседы, который и был использован во многих исследованиях. Анализ текстов из руководства оператора представляет собой другой важный источник информации и его поведении.

Очевидно, что эти методы являются только частью всей иерархии методологии передачи знания, определяя лишь те области приложения инженерии знания, которые апробированы.

База знаний. В центре, как и прежде, находятся факты и правила вывода.

Оболочка ЭС – операционная система для приложения БЗ к поддержке знания:

– система вывода определяет следствия фактов по конкретной ситуации;

– планирующая система выбирает оптимальный способ использования системы выводов для достижения поставленных целей;

– поясняющая система дает сведения о том, на какой основе сделаны выводы.

Система сбора данных. В верхнем овале: процессы передачи знания, которые могут применяться при формировании базы знаний:

– система генерирования знания реализует сбор знания способом грубой индукции моделей, без культурной поддержки;

– система моделирования экспертизы осуществляет приобретение знания методом имитирования поведения эксперта;

– система пополнения характеристик принимает знание по линиям обратной связи;

– система извлечения знания накапливает знание в процессе диалога с экспертом;

– система структурирования знания получает новые знания, основываясь на аналогиях;

– система основных законов осуществляет выбор знаний путем построения моделей, как это делается в моделирующих языках;

– система систематических принципов берет знания путем их вывода из абстрактных принципов.

На рис. 2.2 иерархия методологий получения знаний представлена как единая концепция интеллектуальных информационных систем, интегрирующая в себе различные методологии ИС. Раньше методы получения, интерпретации, представления знания трактовались как совершенно раздельные области исследования.

Структура систем поддержки знаний (СПЗ). Перечислим вначале общие требования для СПЗ, назначение которых заключается в сборе знаний для ЭС:

Рис. 2.2. Иерархия методологий получения знания

– СПЗ должны быть независимыми от предметной области;

– эксперты должны пользоваться СПЗ непосредственно, без промежуточного звена;

– СПЗ должны обладать способностью воспринимать знание из разнообразных источников, включая тексты, беседы с экспертами и наблюдения за поведением экспертов;

– СПЗ должна охватывать многообразие перспектив, включая частичные или противоположные входные данные от различных экспертов;

– система должна уметь охватывать разнообразные формы знания и взаимоотношений между знаниями;

– система должна обладать способностью представлять знание из разнородных источников с достаточной ясностью в отношении его получения, следствий и структурных связей;

– пользователям СПЗ должна быть представлена возможность приложить знание во многих предметных областях, а также возможность свободно экспериментировать с его применением;

– система должна иметь средства для изучения их обоснованности;

– работа системы в максимальной степени должна основываться на хорошо разработанных и понятных теориях сбора знаний и их представления;

– по мере разработки СПЗ она должна приближаться к интегрированной системе.

2.5. Критерии отбора специалиста-эксперта

При формализации предметной области при реализации конкретных ЭС возникает задача тестирования и отбора специалистов, которые могли бы выступить в роли экспертов [15, 37, 52]. Для этих целей целесообразно создавать специальный блок, который формировал бы модель «идеального» эксперта (личностного портрета) в виде набора критериев, которым должен удовлетворять человек, претендующий на роль эксперта по решению конкретного класса задач.

В табл. 2.1 представлены 14 критериев (требований), на основе которых может быть произведен отбор специалистов-экспертов. Критерии приводятся в произвольном порядке, то есть не проранжированы по степени важности с точки зрения разработки ЭС. Процесс ранжирования предлагается осуществить самостоятельно по своему усмотрению.

Таблица 2.1

Требования по отбору специалистов-экспертов

№ п/п	Требования	Возможные варианты ответов
1	2	3
1	**Можете ли Вы…** Эксперт четко осознает границы своих познаний	Да Нет Не уверен
2	**Всегда ли Вы…** Эксперт может ответить на поставленный вопрос, если вопрос поставлен в этих границах	Всегда Не всегда Иногда
3	**Как часто ошибаетесь?** Эксперт обычно не ошибается	Часто Редко Иногда
4	**Можете ли Вы…** Эксперта можно попросить оценить значение некоторого параметра, не поддающегося прямому измерению, и *доверять* этой оценке	Могу Нет Затрудняюсь ответить
5	**Обладаете ли Вы…** Эксперт обладает некоторой (хотя бы неявной) моделью предметной области, поэтому его ответы на различные вопросы всегда согласованы между собой	Да Нет Очень приблизительно
6	**Всегда ли Вы можете…** Эксперт может объяснить причины и/или мотивы своих решений, ответов, рекомендаций	Всегда Не всегда Иногда
7	**Важна ли для Вас степень детализации…** Чем подробнее задаваемые эксперту вопросы, тем больше он выдает информации	Важна Нет Не очень важна
8	**Трудно ли для Вас…** Эксперт может сравнивать несколько ситуаций (вариантов, альтернатив и т.д.), находить в них различия и отличать, принципиальны ли они и к чему приводят	Трудно Не трудно По-разному
9	**Способны ли Вы…** Эксперт способен учесть одновременное воздействие нескольких независимых или взаимозависимых факторов, параметров, критериев и т.д.	Да Нет Затрудняюсь ответить
10	**Можете ли Вы о себе сказать, что Вы…** Эксперт рационален и последователен в своих предпочтениях, поэтому принимаемые им решения разумны и поддаются автоматизации	Да Нет Затрудняюсь ответить
11	**Важен ли для Вас…** Ответ эксперта не зависит от формы и последовательности задаваемых вопросов	Да Нет В отдельных случаях

Окончание табл. 2.1

1	2	3
12	**Вы всегда искренни в Ваших ответах?** Искренность ответов эксперта не позволяет вводить в заблуждение тех, кто задает вопросы	Да Нет Не всегда
13	**Вы беспристрастный человек?** Эксперт должен быть беспристрастным	Да Нет Не всегда
14	**Вам удобно работать в группе?** Эксперт не возражает против работы в группе экспертов, что позволяет согласовать и интегрировать несколько мнений. Это приводит к повышению качества экспертизы	Да Нет Терплю

Следует обратить внимание на то, что все требования, предъявляемые к «идеальному» эксперту, даны в утвердительной форме, однако для облегчения разработки соответствующего блока в таблице приведены и возможные варианты вопросительных конструкций, с помощью которых необходимо перефразировать все критерии.

Кроме того, перед началом тестирования желательно предусмотреть специальный вопрос типа:

«Хотите ли Вы узнать, каким должен быть «идеальный» эксперт?»,

«Система предлагает провести тестирование на предмет отбора специалистов-экспертов для Вашей задачи» и т.д.

Варианты ответов тестируемого на задаваемые вопросы необходимо ранжировать и наложить «пороги» (сумму баллов) приближения к «идеальному» эксперту, используя, например, пять градаций, на основе которых будут оформляться результаты тестирования:

1. *Вы идеальный эксперт для разработки ЭС, связанной с …* (очень хорошая степень приближения).

2. *Мы готовы воспользоваться Вашими услугами в качестве эксперта для данной задачи* (хорошая степень приближения).

3. *В принципе мы можем попробовать начать нашу работу* (средняя степень приближения).

4. *К сожалению, наше сотрудничество при разработке ЭС будет затруднительным* (плохая степень приближения).

5. *Извините, Вы нам не подходите* (очень плохая степень приближения).

В заключение добавим, что тестирование с использованием приведенных 14 критериев (табл. 2.3) можно значительно усложнить, если оценивать еще и влияние реальных условий на «идеального» эксперта.

Ниже приведен перечень так называемых «шумовых» личностных особенностей эксперта, от которых зависит искренность ответов тестируемого:

- профессиональная гордость;
- чрезмерная любезность;
- недопонимание;
- зависимость ответа от контекста и формы вопроса;
- умолчание;
- конформизм, страх «не угадать»;
- когнитивная защита;
- собственные интересы эксперта;
- зависимость ответа от состояния эксперта;
- усталость;
- торопливость;

Однако это задание не является обязательным, поскольку здесь требуется введение более тонких механизмов проставления оценочных коэффициентов, выбора шкал, построения диалога с тестируемым и т.п.

2.6. Методы извлечения знаний когнитологом

Рассмотрим две разновидности коммуникативных методов: пассивные и активные.

Пассивные методы. Термин «пассивные» не должен вызывать иллюзий, поскольку он введен как противовес к «активным» методам. В реальности же пассивные методы требуют от инженера по знаниям не меньшей отдачи, чем такие активные методы, как игры и диалог.

Пассивные методы извлечения знаний включают такие методы, где ведущая роль в процедуре извлечения фактически передается эксперту, а инженер по знаниям только фиксирует рассуждений эксперта во время работы по принятию решений.

Согласно классификации к этой группе относятся:

- наблюдения;

– анализ протоколов «мыслей вслух»;
– лекции.

Наблюдения. В процессе наблюдений инженер по знаниям находится непосредственно рядом с экспертом во время его профессиональной деятельности или имитации этой деятельности. При подготовке к сеансу извлечения эксперту необходимо объяснить цель наблюдений и попросить максимально комментировать свои действия. Во время сеанса аналитик записывает все действия эксперта, его реплики и объяснения. Может быть сделана и видеозапись в реальном масштабе времени. Непременное условие этого метода - невмешательство аналитика в работу эксперта хотя бы на первых порах. Именно метод наблюдений является единственно «чистым» методом, исключающим вмешательство инженера по знаниям и навязывание им каких-то своих структур представлений.

Существуют две основные разновидности проведения наблюдений:

– наблюдение за реальным процессом;
– наблюдение за имитацией процесса.

Обычно используются обе разновидности. Сначала инженеру по знаниям полезно наблюдать за реальным процессом, чтобы глубже понять предметную область и отметить все внешние особенности процесса принятия решения. Это необходимо для проектирования эффективного интерфейса пользователя. Ведь будущая ЭС должна работать именно в контексте такого реального производственного процесса. Кроме того, только наблюдение позволит аналитику увидеть предметную область, а, как известно, «лучше один раз увидеть, чем сто раз услышать».

Наблюдение за имитацией процесса проводят обычно также за рабочим местом эксперта, но сам процесс деятельности запускается специально для аналитика. Преимущество этой разновидности в том, что эксперт менее напряжен, чем в первом варианте, когда он работает на «два фронта»: и ведет профессиональную деятельность, и демонстрирует ее. Недостаток совпадает с преимуществом – именно меньшая напряженность эксперта может повлиять на результат – раз работа ненастоящая, то и решение может отличаться от настоящего.

Наблюдения за имитацией проводят также и в тех случаях, когда наблюдения за реальным процессом по каким-либо причинам

невозможны (например, профессиональная этика врача-психиатра может не допускать присутствия постороннего на приеме). Сеансы наблюдений могут потребовать от инженера по знаниям:

– овладения техникой стенографии для фиксации действий эксперта в реальном масштабе времени;

– ознакомления с методиками хронометража для четкого структурирования производственного процесса по времени;

– развития навыков «чтения по глазам», то есть наблюдательности к жестам, мимике и другим невербальным компонентам общения;

– серьезного предварительного знакомства с предметной областью, так как из-за отсутствия «обратной связи» иногда многое непонятно в действиях экспертов.

Протоколы наблюдений после сеансов в ходе домашней работы тщательно расшифровываются, а затем обсуждаются с экспертом.

Таким образом, наблюдения - один из наиболее распространенных методов извлечения знаний на начальных этапах разработки. Обычно он применяется не самостоятельно, а в совокупности с другими методами.

Анализ протоколов «мыслей вслух». Протоколирование «мыслей вслух» отличается от наблюдений тем, что эксперта просят не просто прокомментировать свои действия и решения, но и объяснить, как это решение было найдено, то есть продемонстрировать всю цепочку своих рассуждений. Во время рассуждений эксперта все его слова, весь «поток сознания» протоколируется инженером по знаниям, при этом полезно отметить даже паузы и междометия. Иногда этот метод называют «вербальные отчеты».

Вопрос об использовании для этой цели магнитофонов и диктофонов является дискуссионным, поскольку магнитофон иногда парализующе действует на эксперта, разрушая атмосферу доверительности, которая может и должна возникать при непосредственном общении.

Основной трудностью при протоколировании «мыслей вслух» является принципиальная сложность для любого человека объяснить, как он думает. При этом существуют экспериментальные психологические доказательства того факта, что люди не всегда в состоянии достоверно описывать мыслительные процессы. Кро-

ме того, часть знаний, хранящихся в невербальной форме (например, различные процедурные знания типа «как завязывать шнурки»), вообще слабо коррелируют с их словесным описанием. Автор теории фреймов М. Минский считает, что «только как исключение, а не как правило, человек может объяснить то, что он знает». Однако существуют люди, склонные к рефлексии, для которых эта работа является вполне доступной. Следовательно, рефлексивность является для эксперта более чем желательной.

Расшифровка полученных протоколов производится инженером по знаниям самостоятельно с коррекциями на следующих сеансах извлечения знаний. Удачно проведенное протоколирование «мыслей вслух» является одним из эффективных методов извлечения, поскольку в нем эксперт может проявить себя максимально ярко, он ничем не скован, никто ему не мешает, он парит в потоке собственных умозаключений и рассуждений. Он может блеснуть эрудицией, продемонстрировать глубину своих познаний, для большого числа экспертов это самый приятный и лестный способ извлечения знаний.

От инженера по знаниям метод «мысли вслух» требует тех же умений, что и метод наблюдений. Обычно «мысли вслух» дополняются потом одним из методов для реализации обратной связи между интерпретацией инженера по знаниям и представлениями эксперта.

Лекции. Лекция является самым старым способом передачи знаний. Нас сейчас интересует не способность эту лекцию слушать, конспектировать и усваивать, а то, что чаще всего экспертов не выбирают, и поэтому учить эксперта читать лекции инженер по знаниям не сможет. Если эксперт имеет опыт преподавателя (например, профессор клиники иди опытный руководитель), то можно воспользоваться таким концентрированным фрагментом знаний, как лекция. В лекции эксперту также предоставлено много степеней свободы самовыражения; при этом необходимо сформулировать эксперту тему и задачу лекции. Например, тема цикла лекций «Постановка диагноза – воспаления легких», тема конкретной лекции «Рассуждения по анализу рентгенограмм», задача – научить слушателей по перечисленным экспертом признакам ставить диагноз воспаления легких и делать прогноз. При такой

постановке он может заранее структурировать свои знания и ход рассуждений.

Опытный лектор знает, что все вопросы можно условно разбить:

– на умные вопросы, углубляющие лекцию;
– глупые вопросы или вопросы не по существу;
– вопросы «на засыпку» или провокационные.

Если инженер по знаниям задает вопросы второго типа, то возможны две реакции. Вежливый эксперт будет разговаривать с таким аналитиком как с ребенком, который сейчас не понимает и все равно ничего уже не поймет. Заносчивый эксперт просто выйдет из контакта, не желая терять время. Если же инженер по знаниям захочет продемонстрировать свою эрудицию вопросами третьего типа, то ничего, кроме раздражения и отчуждения, он, по-видимому, в ответ не получит.

Продолжительность лекции рекомендуется стандартная – от 40 до 50 мин и через 5–10 мин – еще столько же. Курс обычно от двух до пяти лекций.

Метод извлечения знаний в форме лекций, как и все пассивные методы, используют в начале разработки как эффективный способ быстрого погружения инженера по знаниям в предметную область.

В заключение несколько советов, как слушать лекции.

1. К лекции подготовьтесь, то есть познакомьтесь с предметной областью.

2. Слушайте с максимальным вниманием, для этого устраните мешающие факторы (скрип двери, шорохи и т.д.), удобно устройтесь, поменьше двигайтесь.

3. Учитесь отдыхать во время слушания (например, когда лектор приводит цифры, которые можно взять из справочника).

4. Слушайте одновременно и лектора, и самого себя (параллельно *с* мыслями лектора по ассоциации возникают собственные мысли).

5. Слушайте и одновременно записывайте, но записывайте текст сокращенно, используя условные значки (для этого вовсе не следует непременно быть стенографом, достаточно только установить для себя ряд условных значков и ими неизменно пользоваться).

6. Расшифруйте записи лекции в тот же день.

7. Не спорьте с лектором в процессе лекции.

8. Рационально используйте перерывы в лекции для подведения итогов прослушанного.

Активные методы. Активные индивидуальные методы извлечения знаний на сегодняшний день – наиболее распространенные. К основным активным методам можно отнести:

- анкетирование;
- интервью;
- свободный диалог;
- игры с экспертом.

Во всех этих методах активную функцию выполняет инженер по знаниям, который пишет сценарий и режиссирует сеансы извлечения знаний. Игры существенно отличаются от трех других методов. Их можно назвать вопросными методами поиска знаний.

Анкетирование. Анкетирование – наиболее жесткий метод, то есть наиболее стандартизированной. В этом случае инженер по знаниям заранее составляет вопросник или анкету, размножает ее и использует для опроса нескольких экспертов. Это основное преимущество анкетирования.

Сама процедура может проводиться двумя способами:

1. Аналитик вслух задает вопросы и сам заполняет анкету по ответам эксперта.

2. Эксперт самостоятельно заполняет анкету после предварительного инструктирования.

Выбор способа зависит от конкретных условий (например, от оформления анкеты, ее понятности, готовности эксперта). Второй способ нам кажется предпочтительнее, так как у эксперта появляется неограниченное время на обдумывание ответов.

Основными факторами, на которые можно существенно повлиять при анкетировании, являются средства общения (в данном случае это вопросник) и ситуация общения. Вопросник (анкета) заслуживает особого разговора. Существует несколько общих рекомендаций при составлении анкет. Эти рекомендации являются универсальными, то есть не зависят от предметной области.

Анкета не должна быть монотонной и однообразной, то есть вызывать скуку или усталость. Это достигается вариациями фор-

мы вопросов, сменой тематики, вставкой вопросов-шуток и игровых вопросов. Анкета должна быть приспособлена к языку экспертов. Следует учитывать, что вопросы влияют друг на друга и поэтому последовательность вопросов должна быть строго продумана.

Желательно стремиться к оптимальной избыточности. Известно, что в анкете всегда много лишних вопросов, часть из них необходима – это так называемые контрольные вопросы, а другая часть должна быть минимизирована.

Лишние вопросы появляются, например, в таких ситуациях. Фрагмент анкеты:

12. Считаете ли Вы, что для лечения ангины эффективен эритромицин?

13. Какие дозы эритромицина Вы обычна рекомендуете?

При отрицательном ответе на 12-й вопрос 13-й является лишним. Его можно избежать, усложнив вопрос:

12. Применяете ли Вы эритромицин для лечения ангины и если да, то в каких дозах?

Анкета должна иметь «хорошие манеры», то есть ее язык должен быть ясным понятным, предельно вежливым. Методическим мастерством составления анкеты можно овладеть только на практике.

Интервью. Под *интервью* будем понимать специфическую форму общения инженера по знаниям и эксперта, в которой инженер по знаниям задает эксперту серию заранее подготовленных вопросов с целью извлечения знаний о предметной области Наибольший опыт в проведении интервью накоплен, наверное, в журналистике и социологии. Большинство специалистов этих областей отмечают, тем не менее, крайнюю недостаточность теоретических и методических исследований по тематике интервьюирования.

Интервью очень близко тому способу анкетирования, когда аналитик сам заполняет анкету, занося туда ответы эксперта. Основное отличие интервью в том, что оно позволяет аналитику опускать ряд вопросов в зависимости от ситуации вставлять новые вопросы в анкету, изменять темп, разнообразить ситуацию общения. Кроме этого, у аналитика появляется возможность «взять в плен» эксперта своим обаянием, заинтересовать его самой процедурой и тем самым увеличить эффективность сеанса извлечения.

Вопросы для интервью.

Теперь несколько подробнее о центральном эвене активных индивидуальных методов – о вопросах. Инженеры по знаниям редко сомневаются в своей способности задавать вопросы. В то время как и в философии и в математике эта проблем обсуждается с давних лет. Существует даже специальная ветвь математической логики – *эротетическая* логика (логика вопросов).

Все вопросительные предложения можно разбить на два типа:

1. Вопросы *с неопределенностью,* относящейся ко всему предложению («Действительно, введение больших доз антибиотиков может вызвать анафилактический шок?).
2. Вопросы *с неполной информацией* («При каких условиях необходимо включать кнопку?»), часто начинающиеся со слов «кто», «что», «где», «когда» и т. д.

Открытый вопрос называет тему или предмет, оставляя полную свободу эксперту по форме и содержанию ответа («Не могли бы вы рассказать, как лучше сбить высокую температуру у больного с воспалением легких?»).

В закрытом вопросе эксперт выбирает ответ из набора предложенных («Укажите, пожалуйста, что Вы рекомендуете при ангине: а) антибиотики, б) полоскание, в) компрессы, г) ингаляции»). Закрытые вопросы легче обрабатывать при последующем анализе, но они более опасны, так как «закрывают ход рассуждений эксперта и «программируют» его ответ в определенном направлении. При составлении сценария интервью полезно чередовать открытые и закрытые вопросы, особенно тщательно продумывать закрытые, поскольку для их составления требуется определенная эрудиция в предметной области.

Личный вопрос касается непосредственно личного индивидуального опыта эксперта («Скажите, пожалуйста, Иван Данилович, в Вашей практике Вы применяете вулнузан при фурункулезах?»). Личные вопросы обычно активизируют мышление эксперта, «играют» на его самолюбии, они всегда украшают интервью.

Безличный вопрос направлен на выявление наиболее распространенных и общепринятых закономерностей предметной области («Что влияет на скорость процесса ферментации лизина?»).

При составлении вопросов следует учитывать, что языковые способности эксперта, как правило, ограничены и вследствие ско-

ванности, замкнутости, робости он не может сразу высказать свое мнение и предоставить знания, которые от него требуются (даже если предположить, что он их четко для себя формулирует). Поэтому часто при «зажатости» эксперта используют не *прямые* вопросы, которые непосредственно указывают на предмет или тему («Как Вы относитесь к методике доктора Сухарева?»), а *косвенные,* которые лишь косвенно указывают на интересующий предмет («Применяете ли Вы методику доктора Сухарева? Опишите, пожалуйста, результаты лечения»). Иногда приходится задавать несколько десятков косвенных вопросов вместо одного прямого.

Вербальные вопросы – это традиционные устные вопросы. *Вопросы с использованием наглядного материала* разнообразят интервью и снижают утомляемость эксперта. В этих вопросах используют фотографии, рисунки и карточки. Например, эксперту предлагаются цветные картонные карточки, на которых выписаны признаки заболевания. Затем аналитик просит разложить эти карточки в порядке убывания значимости для постановки диагноза.

Деление вопросов по функции на основные, зондирующие, контрольные связано с тем, что часто *основные* вопросы интервью, направленные на выявление знаний не срабатывают – эксперт по каким-то причинам уходит в сторону от вопросов отвечает нечетко.

Тогда аналитик использует *зондирующие* вопросы, которые направляют рассуждения эксперта в нужную сторону. Например, если не сработал основной вопрос: «Какие параметры определяют момент окончания процесса ферментации лизина?», – аналитик начинает задавать зондирующие вопросы: «Всегда ли процесс ферментации длится 72 часа? А если он заканчивается раньше, как это узнать? Если он продлится больше, то что заставит микробиолога не закончить процесс на 72-м часу?» и т. д.

Контрольные вопросы применяют для проверки достоверности и объективности информации, полученной в интервью ранее («Скажите, пожалуйста, а московская школа психологов так же как Вы трактует шкалу К опросника MMPI?» или «Рекомендуете ли вы инъекции АТФ?» (АТФ – препарат, снятый с производства). Контрольные вопросы должны быть «хитро» составлены, чтобы не обидеть эксперта недоверием (для этого используют повторение вопросов в другой форме, уточнения, ссылки па другие источники). «Лучше два раза спросить, чем один раз напутать».

О нейтральных и наводящих вопросах. В принципе интервьюеру (в нашем случае инженеру по знаниям) рекомендуют быть беспристрастным, отсюда и вопросы его должны носить *нейтральный* характер, то есть не должны указывать на отношение интервьюера к данной теме. Напротив, *наводящий* вопрос заставляют респондента (в данном случае эксперта) прислушаться или даже принять во внимание позицию интервьюера. Нейтральный вопрос: «Совпадают ли симптомы кровоизлияния в мозг и сотрясения мозга?» Наводящий вопрос: «Не правда ли, очень трудно дифференцировать симптомы кровоизлияния в мозг?»

Кроме вопросов, перечисленных выше, полезно различать и включать в интервью следующие вопросы:

– контактные («ломающие лед» между аналитиком и экспертом);

– буферные (для разграничения отдельных тем интервью);

– оживляющие память экспертов (для реконструкции отдельных случаев из практики);

– «провоцирующие» (для получения спонтанных, неподготовленных ответов).

В заключение описания интервью укажем три основные характеристики вопросов, которые влияют на качество интервью:

– язык вопроса (понятность, лаконичность, терминология);

– порядок вопросов (логическая последовательность и немонотонность);

– уместность вопросов (этика, вежливость).

Вопрос в интервью – это не просто средство общения, но и способ передачи мыслей и позиции аналитика.

«Вопрос представляет собой форму движения мысли, в нем ярко выражен момент перехода от незнания к знанию, от неполного, неточного знания к более полному и более точному». Отсюда необходимость в протоколах фиксировать не только ответы, но и вопросы, предварительно тщательно отрабатывая их форму и содержание.

Очевидно, что любой вопрос имеет смысл только в контексте. Поэтому вопросы может готовить инженер по знаниям, уже овладевший ключевым набором знаний.

Вопросы имеют для эксперта диагностическое значение – несколько откровенных «глупых» вопросов могут полностью разочаро-

вать эксперта и отбить у него охоту к дальнейшему сотрудничеству.

Свободный диалог. *Свободный диалог* – это метод извлечения знаний в форме беседы инженера по знаниям и эксперта, в которой нет жесткого регламентированного плана и вопросника. Это определение не означает, что к свободному диалогу не надо готовиться. Напротив, внешне свободная и легкая форма этого метода требует высочайшей профессиональной и психологической подготовки. Подготовка к свободному диалогу практически может совпадать с предлагаемой в работе подготовкой к журналистскому интервью. Подготовка занимает разное время в зависимости от степени профессионализма аналитика, но в любом случае она необходима, так как несколько уменьшает вероятность самого нерационального метода – метода проб и ошибок.

Квалифицированная подготовка к диалогу помогает аналитику стать драматургом или сценаристом будущих сеансов, то есть запланировать течение процедуры извлечения: от приятного впечатления в начале беседы к профессиональному контакту через пробуждение интереса и доверия эксперта.

Так, в одном из исследований по технике ведения профессиональных журналистских диалогов было экспериментально доказано, что одобрительное и поощрительное «хмыканье» интервьюера увеличивает длину ответов респондента. При этом одобрение должно быть искренним.

Вопросы по материалам второй главы:

1. Экспертная система – определение, области применения.
2. Отличительные особенности ЭС в классе ИС.
3. Обобщенная структура ЭС, компоненты и алгоритм работы.
4. Человеческий фактор и специализация в среде ЭС.
5. Режимы работы ЭС: приобретения знаний, решения задач
6. Инструментальные средства разработки ЭС.
7. Классификация ЭС.
8. Особенности организации баз знаний в ЭС.
9. Типы знаний в ЭС.
10. Способы (методы) представления знаний
11. Системы поддержки знаний ЭС.
12. Программно-информационные оболочки ЭС.
13. Три класса операторов ЭС.
14. Инженерия знаний.
15. Критерии отбора специалиста-эксперта.
16. Методы извлечения знаний когнитологом.

Глава 3.
ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ДАННЫХ

Колоссальные информационные ресурсы породили проблему их эффективного использования – возникла потребность в развитии прогнозных и экспертно-ориентированных систем [13, 19, 33, 42, 47] с элементами искусственного интеллекта, в основу которых легли бы современные методы «превращения» накопленных разнородных данных в полезные знания.

Начали развиваться технологии направленные на качественный анализ информационных ресурсов. Такие технологии выводят на иной – революционный уровень применения вычислительной техники, переводя ее использование с рельсов математического прагматизма в сферу технических и гуманитарных исследований и превращая информационную парадигму из простого информационного ресурса «хранилища данных» в интеллектуального помощника анализа данных на базе «компетенции знаний». Однако стремление усовершенствовать процессы принятия решений нередко наталкивается на большие объемы и сложную структуру накапливаемых данных. Указанные обстоятельства стимулировали развитие технологий «обнаружения знаний», технологий интеллектуального анализа данных, предназначенных для автоматического поиска в разнотипных и разнородных данных скрытых закономерностей, раскрывающих взаимосвязи в той или иной предметной области [40, 57].

3.1. Общие сведения и терминология

На протяжении десятков лет активно развивается область компьютерных информационных технологий «обнаружение знаний в базах данных» (knowledge discovery in databases). Рядом с этим названием нередко также звучат термины «раскопка данных» (data mining), машинное обучение (machine learning) и «интеллектуальный анализ данных». Все эти термины можно считать синонимами. Их возникновение связано с новым витком в развитии средств и методов обработки различной информации, и такими фундаментальными проблемами искусственного интеллекта

как распознавание и прогнозирование. Этот виток обязан пришедшему пониманию, что в накопленной информации содержатся скрытые знания, которые можно извлечь и воспользоваться в практических целях.

На сегодняшний день известны [2, 9, 13, 16, 33, 40] десятки методологий и сотни алгоритмов интеллектуального анализа данных (статистические, регрессионные, эвристические и т.д.), развита индустрия программного обеспечения [18, 19, 25, 56, 57], аналитические пакеты успешно используются в различных областях науки, бизнеса, промышленности, инноваций, защиты и безопасности [3, 4, 8, 20, 22, 43]. Большинство современных программных приложений, независимо от функционального назначения, снабжено встроенными модулями анализа, основанными на алгоритмах машинного обучения: инструментарии классификации, распознавания, прогнозирования и др.

Направление ИАД родилось как ответ на сложившуюся проблемную ситуацию. Исходное определение дал наш бывший соотечественник Григорий Пятецкий-Шапиро: «Datamining– это процесс обнаружения в сырых данных ранее неизвестных нетривиальных практически полезных и доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности.» (G. Piatetsky-Shapiro).

В настоящее время ИАД существует в двух ипостасях. Ряд специалистов делает акцент на обработке сверхбольших объемов данных. Здесь предъявляются повышенные требования к быстродействию алгоритмов, естественно, в ущерб оптимальности результатов. Подавляющее большинство классических процедур имеют время выполнения, квадратичное или даже кубическое по объёму исходных данных. При количестве объектов, превосходящем несколько десятков тысяч, они работают неприемлемо медленно даже на самых современных компьютерах.

Сфера применения ИАД. Сфера применения интеллектуального анализа данных ничем не ограничена – она везде, где имеются какие-либо данные. Data Mining представляют большую ценность для руководителей и аналитиков в их повседневной деятельности. Технологии интеллектуального анализа данных выводят отрасли производства на уровень экстенсивного развития. Области применения многочисленны и разнообразны, в том числе:

– геология – поиск полезных ископаемых, сейсмопрогнозирование и т.д.;

– сельское хозяйство – прогнозирование урожая, борьбы с вредителями и т.д.;

– административное управление – составление расписаний, оптимизация информационных потоков, мониторинг и контроль показателей деятельности;

– медицина – постановка диагноза, контроль хода лечения и т.д.;

– молекулярная генетика и генная инженерия – определение так называемых маркеров, контролирующих те или иные фенотипические признаки живого организма и т.д.;

– банковское дело - прогнозные модели ценности клиентов и услуг, выявление мошенничества по транзакциям и т.д.;

– прикладная химия – задачи выяснения особенностей, свойств химического строения тех или иных соединений и новых материалов;

– машиностроение, метеорология, связь и телекоммуникации, торговля и делопроизводство, важнейшее значение в военно-промышленном комплексе и т.д., и т.п.

Решаются различные задачи управления, контроля, оценки, в том числе: защиты и безопасности, машинного зрения (распознавание отпечатков пальцев, радужной оболочки глаза, номерных знаков машин, лиц и т.д.), распознавания текстов и перевода, дефектоскопия (металлопрокат, деревообработка и пр.), диагностика сложных технических систем (автомобиле-, корабле-, авиастроение), прогнозирование экономических, биржевых показателей и множество других примеров прикладного применения.

Краткий обзор и современное состояние инструментария ИАД. Интенсивно развивается индустрия программного обеспечения – аналитические пакеты успешно используются в различных областях науки, бизнеса, промышленности. Большинство современных программных приложений, независимо от функционального назначения, снабжено встроенными модулями анализа, основанными на алгоритмах машинного обучения (machine learning) – инструментарии классификации, распознавания, прогнозирования и др. Классические пакеты аналитики (SPSS, STATGRAPHICS,

See5, WizWhy, Hugin и т.д., и т.п.) достигли высокого уровня по факту реализации аналитических методов и алгоритмов и используются не только в коммерческой эксплуатации, но и в образовании.

Много платформ SaaS Business Intelegence (BI) или AaaS – Analysys as a Service отправились в облака, включая Teradata, IBM, Cloud9 Analytics, Cloudscale, In2Clouds, Vertica, Lucidera. Перечень компаний, которые занимаются аналитикой в облаках [55, 57]:

– Birst – предлагает облачную аналитику для менеджеров, продавцов, финансов и т.д.;

– Cloud9 Analytics – предлагает пакет аналитики для продавцов;

– Cloudscale – позволяет использовать анализ данных из облака с анализом данных в режиме онлайн;

– Good Data – анализ клиентской базы в облаке;

– In2Clouds – переносит data-mining и предсказательные модели в облако для продавцов, заказчиков и риск аналитиков;

– OCO – может разворачиваться в виртуальной среде – а значит и в облаке;

– Panorama OLAP – механизм для облака Google, а именно для электронных таблиц Google;

– PivotLink – аналитика по требованию с облачной моделью оплаты, по факту использования;

– QlikTech – лидер в визуализации данных развернул QlikView в облаке;

– Sonoa – реализовала поддержку облака внутри продукта;

– Teradata – аналитика поверх облаков Amazon;

– Vertica – аналитика с использованием облаков Amazon.

Основная политика разработчиков направлена на коммерческое использование и продажу готового продукта, с закрытым кодом и функциональностью заточенной под ограниченный класс потребителей. В большинстве случаев аналитические модули носят вспомогательный характер и предоставляются в виде готовых библиотек для использования их в виде инструментария настроек к платформам, разворачиваемым у клиентов.

Выявление центров компетенции. В области, обозначаемой термином ИАД, с момента его введения произошел бурный рост инструментария ИАД и центров по созданию и поддержка таких

инструментов. На наиболее популярном портале международного сообщества DataMining [57] в настоящее время зарегистрированы сотни производителей, включая грандов мировой софтверной индустрии SAS, Oracle, IBM, SPSS, EMCи др.

В России в основном предпочитают использовать разработки западных партнеров. Вместе с тем, здесь можно выявить несколько центров компетенции, успешно развивающих положительное наследие советского периода в анализе данных и относительно неплохо вписавшихся в международное сотрудничество.

Megaputer Intelligence [58]. Компания была основана в 1993 г. специалистами МГУ. В настоящее время расположена в Блумингтоне, штат Индиана. Компания обслуживает более 100 компаний, 8 федеральных агентств США и больше 500 клиентов в 30 странах. Имеет два научно-исследовательских и проектно-конструкторских центра (R&D centers) в России – в Москве и в Чебоксарах. Имеет представительства в 19 странах.

Главный инструмент компании MegaputerIntelligence PolyAnalyst™ поддерживает предварительную обработку данных, моделирование и демонстрацию результатов. Система PolyAnalyst™ позволит решать проблемы прогнозирования, классификации, кластеризации, группирование по родству, анализа связей, многомерного анализа и интерактивного создания отчетов.

BaseGroup Labs [1]. Это профессиональный поставщик программных продуктов и решений в области анализа данных, специализирующийся на разработке систем для глубокого анализа данных, охватывающих вопросы сбора, консолидации, очистки данных, построения моделей и визуализации. BaseGroupLabscоздана 22 ноября 1995 г. в Рязани и первоначально занималась созданием заказного программного обеспечения. Начиная с 1999 г., компания сконцентрировала свои ресурсы на разработке программных систем, предназначенных для анализа данных. Было выполнено множество проектов в этой области с российскими и зарубежными компаниями, пока со временем все эти разработки не трансформировались в аналитическую платформу Deductor. Deductor – флагманский продукт BaseGroupLabs, вобравший в себя удачные архитектурные идеи и современный математический аппарат.

BaseGroupLabs оказывает поддержку партнерам: обучение, сертификация, консультации, привлечение к реализации проектов,

совместный маркетинг, учет пожеланий партнеров в процессе разработки новых версий Deductor.

Большие усилия компания прилагает к образовательной программе. Выпущена бесплатная академическая версия Deductor, функционирует образовательный портал и система дистанционного обучения продукту. На сайте публикуются статьи с описанием методологии анализа данных, практического применения аналитических алгоритмов, математического аппарата.

НейрОКТехСофт [35]. Основной продукт компании – iLab – платформа для анализа многомерных данных. Кроме того, специалистами компании НейрОКТехСофт разработана библиотека алгоритмов для анализа многомерных данных. Библиотека позволяет решать следующие основные типы задач: разведочный анализ данных, кластерный анализ, регрессионный анализ, классификационный анализ и ряд других. Ключевыми особенностями библиотеки являются: обеспечение возможности обработки и анализа массивов данных большой размерности при невысоком потреблении вычислительных ресурсов, высокая степень масштабируемости, обеспечение возможности интеграции в бизнес-процесс уровня предприятия.

Компания Zsoft [61]. Санкт-Петербургская компания ZSoft специализируется на проектировании, разработке и внедрении информационно-аналитических систем, обладающих возможностями анализа данных с помощью технологий Data Mining и OLAP. Компания была основана в 1999 г. и, главным образом, ориентировалась на выполнение оффшорных заказов. Однако сегодня, когда на российском рынке стали востребованными программные системы со встроенной аналитической функциональностью, компания ZSoft стала предлагать отечественным заказчикам готовые наработки по созданию нетривиальных информационных решений.

Программный продукт Xelopes, созданный компанией Zsoft, в сотрудничестве с немецким партнером Prudsys, является одним из главных достижений компании и заметным явлением в индустрии Data Mining. Система Xelopes представляет собой интегрируемую, поддерживающую отраслевые стандарты, инфраструктуру для решения задач Data Mining. Для этого, в состав Xelopes входит комплекс алгоритмов анализа данных. ZSoft предлагает

также набор услуг по построению хранилищ данных (DataWarehouse) предприятия и основанным на них системам поддержки принятия решений.

Компания Forecsys (ЗАО «Фóрексис») [21]. Компания Forecsys производит программное обеспечение и оказывает консалтинговые услуги в области анализа данных, прогнозирования, моделирования и оптимизации бизнес-процессов. Компания образована в 2000 г. на базе научной школы академика РАН Ю. И. Журавлева. Компания оказывает полный цикл услуг по проектированию, разработке, интеграции и сопровождению корпоративных программных решений. Продукты Forecsys призваны оптимизировать бизнес-процессы клиента с помощью мощного аналитического аппарата и масштабируемых алгоритмических решений. При разработке систем компания делает ставку на эффективность решения поставленной задачи и удобство использования программных средств. Продукты Forecsys ориентированы на банки, телекоммуникационные компании, розничные торговые сети, дистрибьюторские и производственные компании.

Российский статистический пакет STADIA [48]. НПО "Информатика и компьютеры". Пакет STADIA можно отнести к классу мощного инструментария статистической обработки данных, то есть в нем представлены все самые распространенные методы статистического анализа данных от описательной статистики и проверки различных гипотез до анализа временных рядов и контроля качества, а также многомерных (факторный, кластерный, дискриминантный анализ, шкалирование) и непараметрических методов анализа. Пакет не поддерживает обработку больших и тем более сверхбольших объемов данных и не предназначен для работы в облаке. Пакет ориентирован на конкретные статистические расчеты и построение сопутствующих графиков во всех областях прикладной статистики, снабжая пользователя попутно всей необходимой информацией о работе статистических процедур, что по существу уводит его в сторону от идеологии инструментария интеллектуального анализа данных.

DataMining является мультидисциплинарной областью, возникшей и развивающейся на базе достижений прикладной статистики, распознавания образов, методов искусственного интеллек-

та, теории баз данных и др. Отсюда обилие методов и алгоритмов, реализованных в различных действующих системах DataMining. Многие из таких систем интегрируют в себе сразу несколько подходов. Тем не менее, как правило, в каждой системе имеется какая-то ключевая компонента, на которую делается главная ставка.

3.2. Методы и алгоритмы ИАД

За последние десятилетия значительные усилия в области ИАД были направлены на создание специализированных алгоритмов, способных выполнять те же задачи за линейное или даже логарифмическое время без существенной потери точности.

Другая группа специалистов концентрирует внимание на глубине раскопки данных. В понимании второй группы основные отличия технологии ИАД следующие:

ИАД – это всегда сугубо многомерные задачи – поиск связи между значением целевого показателя и набором значений группы других показателей базы данных.

Технология ИАД способна обрабатывать разнородную информацию, то есть поля могут быть представлены количественными, качественными и текстовыми переменными.

Технология ИАД в отличие от традиционных статистических методов не претендует на поиск взаимосвязей, характерных для полного объема данных (всей выборки). Ищутся правила, связывающие значения показателей, для подвыборок данных. При этом правила всегда высокоточные, а не «размытые» по всей выборке, общие и неточные статистические тенденции.

Алгоритмы ИАД производят поиск указанных выше подвыборок данных и точных взаимосвязей для этих подвыборок в автоматическом режиме.

Таким образом, ключевые слова ИАД – точность, многомерность, разнотипность данных, автоматический поиск. Здесь, конечно, еще нужно добавить важное требование интерпретируемости получаемого результата.

Дополнительную информацию можно найти на многочисленных сайтах интернета, из которых один из самых информативных – портал упомянутого выше Г. Пятецкого-Шапиро [57]. Также весьма полезным является популярный ресурс – репозиторий дан-

ных UCI университета г. Ирвин (Калифорния, США) [59], история которого началась в 1987 г. Здесь можно найти массу данных и ссылок на примеры решения задач из самых разных областей.

Методы ИАД имеют много общего с методами решения задач классификации, диагностики и распознавания образов. Но их одной из главных отличительных черт, как отмечалось выше, является функция интерпретации закономерностей, положеных в основу правил вхождения объектов в классы эквивалентности. Поэтому здесь большое распространение получили логические методы. Есть еще одна важная причина, обусловившая приоритет логических методов. Она заключается в сложной системной организации областей, составляющих предмет приложения современных информационных технологий. Эти области относятся, как правило, к надкибернетическому уровню организации систем, закономерности которого не могут быть достаточно точно описаны на языке статистических или иных аналитических математических моделей. Гибкость и многообразие логических конструкций индуктивного вывода позволяют нередко добиваться успешных результатов при описании таких сложных систем.

Другие методы ИАД для построения диагностических и прогностических моделей имеют менее прозрачную интерпретацию. Сюда относятся байесовские классификаторы, дискриминантный анализ, нейросетевой подход, метод ближайших соседей, метод опорных векторов, генетические алгоритмы и др. Как показала практика последнего десятилетия, в ряде задач (особенно в бизнес-приложениях, где требуется анализировать огромные базы данных), требование интерпретируемости результатов стало отступать на задний план. Акцент здесь стал делаться на стабильности получаемых решений. Более того, на передний план начали выходить методы работы с комитетами, содержащими сотни и тысячи методов и алгоритмов. Как выяснилось, подобные комитеты, состоящие даже из «слабых» алгоритмов, способны превосходить по точности изолированные «сильные» алгоритмы, нацеленные на поиск глубоких закономерностей в массивах данных. Эта тенденция современного ИАД нуждается в самостоятельном рассмотрении. Здесь наблюдается явное отступление от изначальных идеалов ИАД, связанных с попытками извлечения знаний из данных, а не построением моделей в виде «черных ящиков».

При работе с комитетами алгоритмов сегодня широко используются два общих технологических приема или метода, имеющих чрезвычайную важность для ИАД. Это «бустинг» (boosting) и «бэггинг» (bagging – сокращение от «bootstrapaggregation»). Эти приемы предназначены для повышения «обобщающей способности» получаемых моделей – способности выдавать правильные результаты не только для примеров, участвовавших в процессе обучения, но и для любых новых, не участвовавших в процессе обучения данных. Кратко охарактеризуем эти два приема.

Бустинг реализует процедуру последовательного построения композиции алгоритмов машинного обучения, когда каждый следующий алгоритм стремится компенсировать недостатки композиции всех предыдущих алгоритмов. Теоретическое обоснование эффективности бустинга связано с тем, что взвешенное голосование сглаживает ответы алгоритмов, входящих в комитет. Эффективность бустинга объясняется тем, что по мере добавления базовых алгоритмов увеличиваются отступы обучающих объектов. Причём бустинг продолжает раздвигать классы даже после достижения безошибочной классификации обучающей выборки.

Бэггинг – это метод формирования ансамблей классификаторов с использованием случайной выборки с возвратом или бутстрепа. Он был предложен в 1994 г. Breiman Leo. При формировании бутстреп-выборок из множества данных случайным образом отбирается несколько подмножеств. Затем на основе каждого подмножества (выборки) строится классификатор. Выходы полученных классификаторов комбинируются (агрегируются) путем голосования или простого усреднения. Считается, что результат будет намного точнее любой одиночной модели, построенной на исходном наборе данных.

В целом, как было отмечено выше, в области интеллектуального анализа данных за последнее десятилетие произошли существенные изменения. Слово «интеллектуальный» теперь, скорее, нужно воспринимать в контексте автоматического построения классифицирующих и прогнозирующих моделей. Поиск индивидуально сильных методов и алгоритмов для основной массы специалистов ИАД стал не столь привлекательным – их интересы сместились в сторону умений работать с большими коллективами «слабых» методов и алгоритмов.

Вместе с тем, проблема построения «сильных» моделей, на наш взгляд, не потеряла своей актуальности. По-видимому, сильные модели являются более пригодными для интерпретации и объединяются в менее громоздкие коллективы для достижения эффективных результатов в задачах классификации и прогнозирования.

Охарактеризуем три «классические» группы методов ИАД.

Методы решающих функций. Методы основаны на алгоритмах метрической классификации и оценивании сходства объектов [9, 24, 44, 49]. Классифицируемый объект относится к тому классу, которому принадлежат ближайшие к нему объекты обучающей выборки. Наличие обучающей выборки обязательно, поэтому эти методы называют еще основанными на прецедентах.

Деревья решений. Методы обнаружения закономерностей, основанные на древовидных алгоритмах поиска и представления зависимостей [18, 19, 33]. Цель состоит в том, чтобы создать модель, которая предсказывает значение целевой переменной на основе нескольких переменных на входе.

Искусственные нейронные сети (ИНС). Методы построены на основе математических моделей, построенных по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма [2, 7, 31]. ИНС представляют собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Возможность обучения – одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей между нейронами. В процессе обучения нейронная сеть способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение.

3.3. Методы решающих функций

Методы неявно опираются на одно важное предположение, называемое гипотезой компактности: *если мера сходства объектов введена достаточно удачно, то схожие объекты гораздо чаще лежат в одном классе, чем в разных*. В этом случае граница между классами имеет достаточно простую форму, а классы образуют компактно локализованные области в пространстве объектов.

Охарактеризуем кратко наиболее значимые методы [44]:

1. Метод *k* ближайших соседей. Метод k ближайших соседей (другое название – метод Фикса–Ходжеса) реализует относительно простое по сравнению с другими методами распознавания, но во многих случаях достаточно эффективное правило классификации. Эффективность получаемых с помощью данного метода решающих функций в значительной степени зависит от представительности обучающей выборки, числа соседей, на основании которого принимается решение, а также выбранной метрики.

2. **Метод аппроксимации плотности распределения функциями.** При использовании байесовского классификатора вместо условных и априорных вероятностей классов, как правило, приходится применять их оценки. Поскольку на практике часто встречаются задачи, в которых априорные вероятности классов известны или есть основания предполагать их одинаковыми для всех классов, то в этом случае основная проблема заключается в нахождении оценок условных вероятностей классов. Точность получаемой оценки, а значит, и качество распознавания этим методом, существенно зависит от трех факторов: выбранной системы базисных функций, числа элементов разложения, а также объема и представительности обучающей выборки.

3. **Байесовский классификатор для случая нормального распределения классов**. Ограничения и сложности в использовании байесовского классификатора определяются тем обстоятельством, что в реальных задачах распознавания часто не представляется возможным непосредственно задать условные и априорные вероятности классов и приходится тем или иным методом вычислять их оценки на основании имеющегося обучающего материала, что с неизбежностью приводит к ухудшению качества классификации при применении соответствующих решающих правил. Поэтому байесовский классификатор в общем виде обычно используется лишь для частных случаев задач распознавания. Одним из них является случай нормального распределения объектов классов.

4. **Метод корректирующих приращений.** Метод корректирующих приращений представляет собой одну из реализаций применительно к задачам распознавания алгоритма Роббинса–Монро, являющегося разновидностью методов стохастической аппрокси-

мации. В методах распознавания, основанных на алгоритмах стохастической аппроксимации, решающие функции классов строятся посредством использования известной из теории вероятностей формулы Байеса и замены плотности её оценкой. Когда классы линейно неразделимы, метод корректирующих приращений в пределе позволяет построить решающую функцию, обеспечивающую минимум ошибок.

5. **Метод эталонов.** Метод эталонов, как и метод *k* ближайших соседей, реализует решающее правило, основанное на понятии «близости» между объектами в заданной метрике. Распознавание объекта заключается в вычислении характеристических функций для эталонов (элементов обучающей выборки) и определения класса, которому соответствует максимальная из этих функций. Как и для метода *k* ближайших соседей, эффективность получаемой с помощью метода эталонов решающей функции в значительной мере зависит от представительности обучающей выборки и выбранной метрики.

6. **Метод вычисления оценок.** Распознавание в методе вычисления оценок происходит по следующей процедуре. Выбирается система опорных множеств. Для каждого опорного множества и каждого объекта обучающей выборки вычисляется значение функции близости, после чего формируются оценки. Подсчитываются оценки распознаваемого объекта для каждого класса по всей системе опорных множеств. Объект относится к тому классу, которому соответствует максимальная оценка. Эффективность получаемой посредством метода вычисления оценок решающей функции определяется выбранной системой опорных множеств и значениями порогов близости.

7. **Метод потенциальных функций.** Метод потенциальных функций базируется на использовании специальных функций, называемых потенциальными. Решающая функция связана с потенциальными функциями и строится в ходе рекуррентной процедуры. Корректирующая функция представляет собой некоторую потенциальную функцию, умноженную на зависящий от шага числовой коэффициент. Одним из аргументов потенциальной функции является вектор описания текущего обучающего объекта, вторым – вектор описания распознаваемого объекта. Для решения

задач распознавания используют две разновидности метода потенциальных функций: на базе детерминистского и статистического подходов к распознаванию. Качество получаемых с помощью метода потенциальных функций решающих правил в значительной мере определяется сложностью используемых потенциальных функций.

3.4. Методы и алгоритмы построения деревьев решений

Деревья решений – это способ представления правил в иерархической, последовательной структуре, где каждому объекту соответствует единственный узел, дающий решение. Под правилом понимается логическая конструкция, представленная в виде "если ... то ...".

Деревья решений являются одним из наиболее популярных подходов к решению задач DataMining [1,19]. С их помощью решаются задачи прогнозирования, классификации, распознавания образов, сегментации БД, извлечения из данных скрытых знаний, интерпретации данных, установления ассоциаций в БД и др. Они создают имеющую вид дерева иерархическую структуру классифицирующих правил типа "ЕСЛИ... ТО..." (if-then). Популярность подхода связана с эффективностью таких алгоритмов, наглядностью, понятностью, интерпретируемостью результата. Большое количество систем ИАД используют именно этот метод. Самыми известными являются See5/C5.0 (RuleQuest, Австралия), Clementine (IntegralSolutions, Великобритания), SIPINA (UniversityofLyon, Франция), IDIS (InformationDiscovery, США), KnowledgeSeeker (ANGOSS, Канада). Стоимость этих систем варьируется от 1 до 10 тыс. долл.

Алгоритмы «деревья решений» реализуют наивный принцип последовательного просмотра признаков, и их главной проблемой является проблема перебора вариантов за приемлемое время. Известные методы [33] либо искусственно ограничивают такой перебор (алгоритмы КОРА, WizWhy), либо строят деревья решений (алгоритмы CART, CHAID, ID3, See5, Sipina и др.), имеющих принципиальные ограничения эффективности поиска if-then правил. Другие проблемы связаны с тем, что известные методы поиска логических правил не поддерживают функцию обобщения

найденных правил и функцию поиска оптимальной композиции таких правил. Удачное решение указанных проблем может составить предмет новых конкурентоспособных разработок.

На сегодняшний день существует значительное число алгоритмов, реализующих деревья решений CART, C4.5, NewId, ITrule, CHAID, CN2 и т.д. Но наибольшее распространение и популярность получили следующие два:

CART (ClassificationandRegressionTree) – это алгоритм построения бинарного дерева решений – дихотомической классификационной модели. Каждый узел дерева при разбиении имеет только двух потомков. Как видно из названия алгоритма, он решает задачи классификации и регрессии.

C4.5 – алгоритм построения дерева решений, количество потомков у узла не ограничено. Не умеет работать с непрерывным целевым полем, поэтому решает только задачи классификации.

Большинство из известных алгоритмов являются «жадными алгоритмами». Если один раз был выбран атрибут и по нему было произведено разбиение на подмножества, то алгоритм не может вернуться назад и выбрать другой атрибут, который дал бы лучшее разбиение. И поэтому на этапе построения нельзя сказать, даст ли выбранный атрибут, в конечном счете, оптимальное разбиение.

Для построения дерева на каждом внутреннем узле необходимо найти такое условие (проверку), которое бы разбивало множество, ассоциированное с этим узлом на подмножества. В качестве такой проверки должен быть выбран один из атрибутов. Общее правило для выбора атрибута можно сформулировать следующим образом: выбранный атрибут должен разбить множество так, чтобы получаемые в итоге подмножества состояли из объектов, принадлежащих к одному классу, или были максимально приближены к этому, то есть количество объектов из других классов («примесей») в каждом из этих множеств было как можно меньше.

1. Алгоритм CART (ID3)

Алгоритм CART использует так называемый индекс Gini (в честь итальянского экономиста CorradoGini), который оценивает "расстояние" между распределениями классов. CART был предложен Л. Брейманом (L. Breiman) и др.

$$\text{Gini}(c) = 1 - \sum_j p_j^2, \quad (3.1)$$

где c – текущий узел; p_j – вероятность класса j в узле c.

Очень часто алгоритмы построения деревьев решений дают сложные деревья, которые "переполнены данными", имеют много узлов и ветвей. Такие "ветвистые" деревья очень трудно понять. К тому же ветвистое дерево, имеющее много узлов, разбивает обучающее множество на все большее количество подмножеств, состоящих из все меньшего количества объектов.

Ценность правила, справедливого, скажем, для 2-3 объектов, крайне низка, и в целях анализа данных такое правило практически непригодно. Гораздо предпочтительнее иметь дерево, состоящее из малого количества узлов, которым бы соответствовало большое количество объектов из обучающей выборки. И тут возникает вопрос: а не построить ли все возможные варианты деревьев, соответствующие обучающему множеству, и из них выбрать дерево с наименьшей глубиной. К сожалению, это задача является NP-полной, это было показано Л. Хайфилем (L. Hyafill) и Р. Ривестом (R. Rivest), и, как известно, этот класс задач не имеет эффективных методов решения. Для решения вышеописанной проблемы часто применяется так называемое отсечение ветвей (pruning).

Пусть под точностью (распознавания) дерева решений понимается отношение правильно классифицированных объектов при обучении к общему количеству объектов из обучающего множества, а под ошибкой – количество неправильно классифицированных. Предположим, что нам известен способ оценки ошибки дерева, ветвей и листьев. Тогда возможно использовать следующее простое правило: построить дерево, отсечь или заменить поддеревом те ветви, которые не приведут к возрастанию ошибки.

В отличие от процесса построения, отсечение ветвей происходит снизу вверх, двигаясь с листьев дерева, отмечая узлы как листья, либо заменяя их поддеревом. Хотя отсечение не является панацеей, но в большинстве практических задач дает хорошие результаты, что позволяет говорить о правомерности использования подобной методики.

Основная проблема состоит в том, чтобы построить достаточно хорошее дерево решений. Один из алгоритмов решения этой

задачи, известный как алгоритм ID3, представлен ниже. На шаге 3.1 данного алгоритма используется понятие информационного выигрыша атрибута.

Ввод: Множество *X* объектов обучающей выборки; набор *A* дискретных атрибутов объектов.

Вывод: Построенное дерево решений.

1: Создать вершину *N*;

2: **if** все объекты из *X* одного класса C **then**

3: Возвращаем вершину *N*, как лист, соответствующий классу *C*;

4: **endif**

5: **if** A = ø **then**

6: Возвращаем вершину *N*, как лист, соответствующий наиболее распространенному в *X* классу *C*;

7: **endif**

8: Выбираем среди атрибутов множества *A* атрибут a с наивысшим информационным выигрышем;

9: Сопоставляем вершине *N* атрибут *a*;

10: **forall** известные значения $\overline{a}$ атрибута ado

11: Создаем ветвь из вершины *N*, соответствующую условию $a = \overline{a}$;

12: Пусть $\overline{X}$ – множество объектов из *X*, для которых атрибут a равен $\overline{a}$.

13: **if** $\overline{X}$ = ø **then**

14: Присоединяем к *N* лист и сопоставляем ему метку самого распространенного в *X* класса *C*;

15: **else**

16: {Рекурсивный вызов}

17: Положим, $\overline{N}$:= GenerateDecisionTree ($\overline{X}$, $A \setminus \{a\}$).

18: Присоединяем к созданной ветви из *N* дерево $\overline{N}$;

19: **endif**

20: **endfor**

2. Алгоритм C4.5

Представляет собой усовершенствованный вариант алгоритма ID3. Среди улучшений стоит отметить следующие:

– возможность работать не только с категориальными атрибутами, но также с числовыми. Для этого алгоритм разбивает об-

ласть значений независимой переменной на несколько интервалов и делит исходное множество на подмножества в соответствии с тем интервалом, в который попадает значение зависимой переменной;

– после построения дерева происходит усечение его ветвей. Если получившееся дерево слишком велико, выполняется либо группировка нескольких узлов в один лист, либо замещение узла дерева нижележащим поддеревом. Перед операцией над деревом вычисляется ошибка правила классификации, содержащегося в рассматриваемом узле. Если после замещения (или группировки) ошибка не возрастает (и не сильно увеличивается энтропия), значит, замену можно произвести без ущерба для построенной модели.

Одним из недостатков алгоритма ID3 является то, что он некорректно работает с атрибутами, имеющими уникальные значения для всех объектов из обучающей выборки. Для таких объектов информационная энтропия равна нулю, и никаких новых данных от построенного дерева по данной зависимой переменной получить не удастся. Поскольку получаемые после разбиения подмножества буду содержать по одному объекту.

Алгоритм C4.5 решает эту проблему путём введения нормализации. Оценивается не количество объектов того или иного класса после разбиения, а число подмножеств и их мощность (число элементов). Выражение

$$\text{splitinfo}(x_h) = -\sum_{i=1}^{m} \frac{T_i}{T} \log_2 \left(\frac{T_i}{T}\right) \tag{3.2}$$

оценивает потенциальную информацию, получаемую при разбиении множества T на m подмножеств.

Критерием выбора переменной для разбиения будет выражение:

$$\text{gainratio}(x_h) = \frac{\text{Gain}(x_h)}{\text{splitinfo}(x_h)} \tag{3.3}$$

или

$$\text{gainratio}(x_h) = \frac{\text{Gain}(x_h)}{-\sum_{i=1}^{m} \frac{T_i}{T} \log_2 \left(\frac{T_i}{T}\right)}. \tag{3.4}$$

При условии, что имеется k классов и n – число объектов в обучающей выборке и одновременно количество значений пере-

менных, тогда числитель максимально будет равен $\log_2 k$, а знаменатель максимально равен $\log_2 n$. Если предположить, что количество объектов заведомо больше количества классов, то знаменатель растёт быстрее, чем числитель и, соответственно, значение выражения будет небольшим.

В обучающей выборке могут присутствовать объекты с пропущенными значениями атрибутов. В этом случае их либо отбрасывают (что влечёт за собой риск потерять часть данных), либо применить подход, предполагающий, что пропущенные значения по переменной вероятностно распределены пропорционально частоте появления существующих значений.

3. Алгоритм покрытия

Алгоритм заключается в построении деревьев решений для каждого класса по отдельности. На каждом этапе генерируется проверка узла дерева, который покрывает несколько объектов обучающей выборки. На каждом шаге алгоритма выбирается значение переменной, которое разделяет множество на два подмножества. Разделение должно выполняться так, чтобы все объекты класса, для которого строится дерево, принадлежали одному подмножеству. Такое разбиение производится до тех пор, пока не будет построено подмножество, содержащее только объекты одного класса. Для выбора независимой переменной и её значения, которое разделяет множество, выполняются следующие действия:

1. Из построенного на предыдущем этапе подмножества (для первого этапа это вся обучающая выборка), включающего объекты, относящиеся к выбранному классу для каждой независимой переменной, выбираются все значения, встречающиеся в этом подмножестве.

2. Для каждого значения каждой переменной подсчитывается количество объектов, удовлетворяющих этому условию и относящихся к выбранному классу.

3. Выбираются условия, покрывающие наибольшее количество объектов выбранного класса.

4. Выбранное условие является условием разбиения подмножества на два новых.

После построения дерева для одного класса таким же образом строятся деревья для других классов.

Сравнение алгоритмов построения деревьев решений. На основе анализа популярных алгоритмов построения деревьев решений можно построить следующую сводную таблицу их сильных и слабых сторон (табл. 3.1):

Таблица 3.1

Сравнительная таблица алгоритмов построения деревьев решений.

Алгоритм	Преимущества	Недостатки
CART	1. Бинарное представление дерева решений. 2. Функция оценки качества разбиения. 3. Механизм отсечения дерева. 4. Алгоритм обработки пропущенных значений. 5. Построение деревьев регрессии	1. Низкая скорость работы. 2. Сложность интерпретируемости классификации
ID3	1. Простота и интерпретируемость классификации. 2. Алгоритм синтеза решающего дерева имеет сложность, линейную по длине выборки. 3. Алгоритм очень прост для реализации и легко поддаётся различным усовершенствованиям	1. Жадность. В случае выбора неоптимального предиката алгоритм не способен вернуться на уровень вверх и заменить неудачный предикат. 2. Алгоритм склонен к переобучению – как правило, он усложняет структуру дерева
C4.5	1. Возможность бинарного представления дерева решений. 2. Возможность работать с числовыми атрибутами. 3. Механизм отсечения дерева	1. Жадность. 2. Сложность реализации
Алгоритм покрытия	1. Высокая точность. 2. Возможность работы с любыми видами атрибутов	1. Сложность реализации. 2. Высокое требование к памяти. 3. Нелинейная сложность от количества атрибутов

3.5. 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. 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 определенной степенью достоверности. В качестве таких адаптивных систем применяются искусственные нейронные сети [1, 2, 7].

Искусственная нейронная сеть – это значительным образом упрощенная модель биологической нейронной сети (нервной ткани биологического мозга). Естественная нервная клетка (нейрон) состоит из тела *(сомы),* содержащего ядро, отростков (*дендритов),* по которым в нейрон поступают входные сигналы и одного отростка (аксон) на конце сомы, который служит для передачи выходных сигналов нейрона другим нервным клеткам. Соединение аксона с дендритом другого нейрона называется *синапсом.* Нейрон возбуждается и передает сигнал через аксон, если число пришедших по дендритам возбуждающих сигналов больше, чем число тормозящих.

Из биологии заимствованы следующие основополагающие принципы:

– Нейрон – это переключатель, получающий и передающий импульсы, или сигналы. Если нейрон получает достаточно сильный импульс, то говорят, что нейрон активирован, то есть передает импульсы связанным с ним нейронам. Неактивированный нейрон остается в состоянии покоя и не передает импульс.

– Нейрон состоит из нескольких компонентов: синапсов, соединяющих нейрон с другими нейронами и получающих импульсы от соседних нейронов, аксона, передающего импульс другим нейронам, и дендрита, получающего сигналы из различных источников, в том числе от синапсов.

– Когда нейрон получает импульс, превышающий определенный порог, он передает импульс последующим нейронам (активирует импульс).

– Синапс состоит из двух частей: пресинаптической, соединенной с аксоном передающей импульс клетки, и постсинаптиче-

ской, соединенной с дендритом получающей импульс клетки. Обе части синапса соединяет синаптическая цепь.

Сеть ИНС представляет собой совокупность простых вычислительных элементов – искусственных нейронов, каждый из которых обладает определенным 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. 3.1) представляет собой дискретно-непрерывный преобразователь информации. Информация, поступающая на вход нейрона, суммируется с учетом весовых коэффициентов w_i сигналов x_i, $i = 1, ..., n$, где n – размерность пространства входных сигналов. Потенциал нейрона определяется по формуле $P = \sum_{i=1}^{n} w_i x_i$

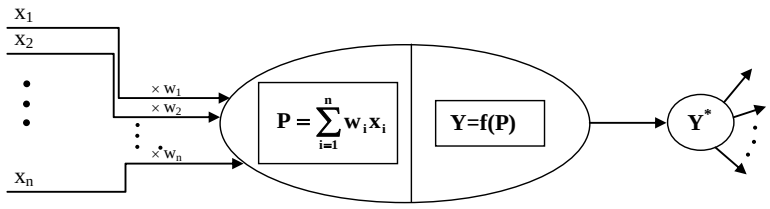


Рис. 3.1. Схема математической модели нейрона

Взвешенная сумма поступивших сигналов (потенциал) преобразуется с помощью передаточной функции $f(P)$ в выходной сигнал нейрона Y, который передается другим нейронам сети, то есть $Y = f(P)$. Вид передаточной (активационной) функции является важнейшей характеристикой нейрона. Существует множество различных функций, используемых в качестве передаточных. Выбор

вида передаточной функции – это сложная задача. Как правило, к ним выдвигаются требования иметь область значения [0, 1] или [-1, 1] и быть возрастающими.

Математическая модель нейрона строится следующим образом:

– Вход модели нейрона X – это вектор, состоящий из n компонент. Каждая из компонент входного вектора X_i – это один из импульсов, получаемых нейроном.

– Выход модели нейрона – это одно число Y^*. Это означает, что внутри модели, входной вектор должен быть преобразован и агрегирован в скаляр. В дальнейшем этот импульс будет передан другим нейронам.

– Известно, что при получении импульса синапс нейрона изменяет его. Математически этот процесс изменения можно описать следующим образом: для каждой из компонент входа x_i задают вес. Импульс, прошедший через синапс, принимает вид $w_i x_i$. Заметим, что веса могут быть назначены при инициализации модели, а могут быть переменными и измеряться в ходе расчетов. Обучение сети – это в первую очередь нахождение весов синапса.

– Сложение полученных импульсов. Агрегирование полученных импульсов – это вычисление их суммы $\sum w_i x_i$.

Будет активирован нейрон или нет, определяется вычислением так называемой передаточной, или активационной, функции нейрона $f(\sum w_i x_i)$. Если значение функции превышает некоторый заранее определенный порог, то нейрон активирован и передает импульс следующим нейронам в сети. То есть передаточная функция должна моделировать скачок, резкий переход в состояние активации. В общем случае эта функция может быть ступенчатой (пороговой), линейной или нелинейной (рис. 3.2).

1. Пороговая функция. Для простой передаточной функции нейросеть может выдавать 0 и 1, 1 и -1 или другие числовые комбинации. Передаточная функция в таких случаях является "жестким ограничителем" или пороговой функцией (рис. 3.2). Пороговая функция $f(P)$ пропускает информацию только в том случае, если алгебраическая сумма входных сигналов превышает некоторую постоянную величину P^*, например: $Y = 1$ если $P >= P^*$ и $Y = -1$ при $P < P^*$.

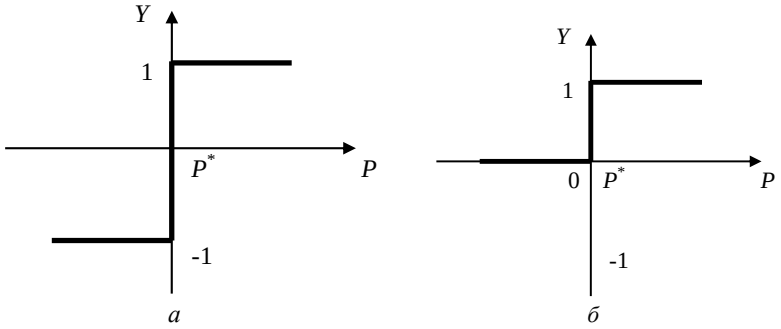


Рис. 3.2. Пороговая функция. *а* – $Y = 1$, если $P >= P^*$ и $Y = -1$ при $P < P^*$; *б* – $Y = 1$, если $P >= P^*$ и $Y = 0$ при $P < P^*$

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.

2. Линейная функция. Линейная функция $Y = kP$ дифференцируема и легко вычисляется (рис. 3.3), 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.

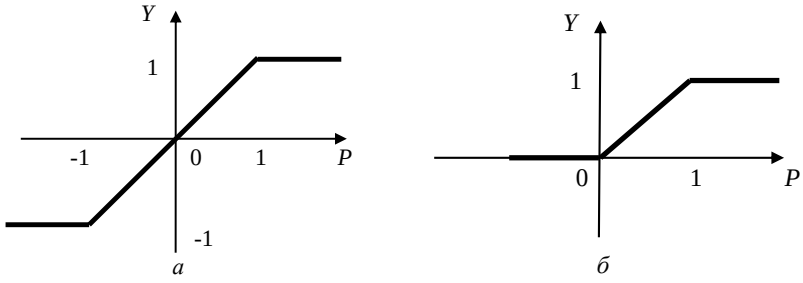
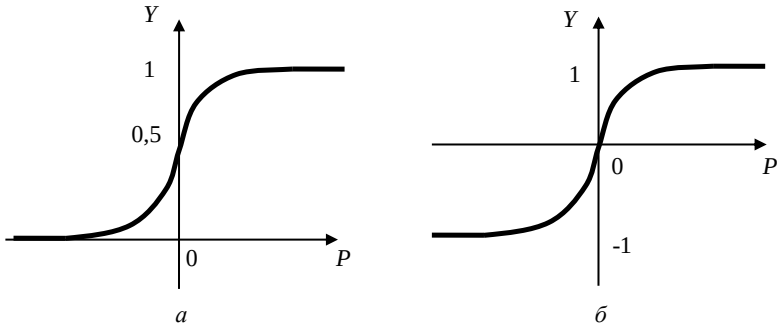


Рис. 3.3. Линейная функция. *а* – $Y = -1$ если $P < -1$, $Y = P$ если $-1 \le P \le 1$, $Y = 1$ если $P >= 1$; *б* – $Y = 0$ если $P < 0$, $Y = P$, если $0 \le P \le 1$, $Y = 1$, если $P >= 1$

3. S-подобная функция. Передаточная функция называется сигмоидой (рис. 3.4, *а*), когда ее диапазон [0, 1], или гиперболиче-

ским тангенсом (рис. 3.4, *б*) при диапазоне [-1, 1]. Определенным компромиссом между линейной и ступенчатой функциями является сигмоидальная функция переноса $Y = 1/(1 + e^{-kP})$, которая удачно моделирует передаточную характеристику биологического нейрона (рис. 3.4, *а*). 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. Важной чертой *S*-кривых является непрерывность функций и их производных (в отличие от ступенчатой функции). *S*-подобная функция дифференцируема, как и линейная функция, и это качество можно использовать при поиске экстремума в пространстве параметров ИНС.



а *б*

Рис. 3.4. *S*-подобные передаточные функции.
а – $Y = 1/(1 + e^{-kP})$; *б* – $Y = 1 - 1/(1 + P)$, если $P \geq 0$, $Y = -1 + 1/(1 - P)$, если $P \leq 0$

Тип функции переноса выбирается с учетом конкретной задачи, решаемой с применением нейронных сетей.

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 соединен с несколькими нейронами следующего слоя, то по-

лучается 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.

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

Модели нейронных сетей. Теоретические основы нейронных сетей были заложены в начале 1940-х годов. Норбертом Винером в его работе «Кибернетика или управление и связь в животном и машине». Дж. Маккалохом и У. Питтс разработали собственную теорию деятельности головного мозга. К главным результатам их работы относятся следующие:

– предположение о том, что нейронная сеть способна обучаться и распознавать образы;

– 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.

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 ее с порогом, чтобы определить свое собственное состояние.

Серьезное развитие нейрокибернетика получила в трудах американского нейрофизиолога Ф. Розенблата, который предложил свою модель нейронной сети в 1958 г. и продемонстрировал созданное на ее основе электронное устройство, названное *перцептроном*. Ф. Розенблат ввел возможность модификации межнейронных связей, что сделало ИНС обучаемой.

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 распознает результат, то весовым коэффициентам дается приращение в сторону повышения качества распознавания.

Теоретический анализ перцептрона, проведенный 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.

Многослойные сети. В многослойных сетях устанавливаются связи только между нейронами соседних слоев, как показано на рис. 3.5.

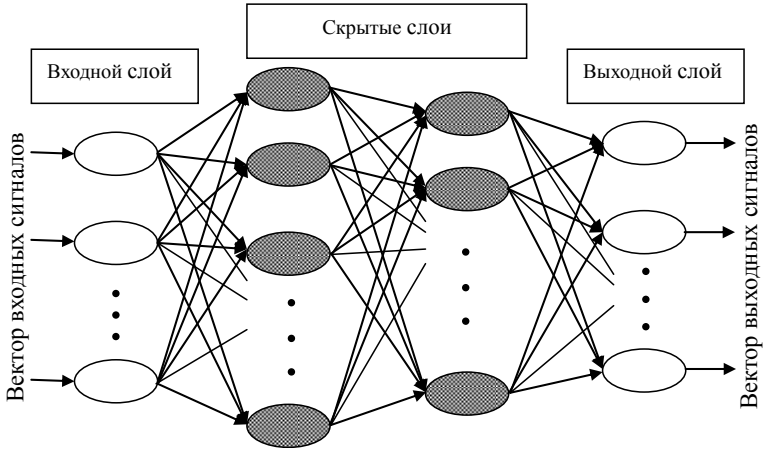


Рис. 3.5. Схема многослойного перцептрона

Каждый элемент может быть соединен модифицируемой связью с любым нейроном соседних слоев, 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 путем последовательного вычисления уровней активности элементов каждого слоя (снизу вверх) с использованием уже известных значений активности элементов предшествующих слоев. 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 30–15–8, то есть 30 узлов находится в первом слое, 15 узлов – в скрытом слое и 8 – в выходном. Определение числа скрытых слоев 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» [2].

Выше отмечалось, что простой перцептрон с одним слоем обучаемых связей формирует границы областей решений в виде гиперплоскостей. Двухслойный перцептрон может выполнять операцию «логического и» $\wedge$ над полупространствами, образованными гиперплоскостями первого слоя весов. Это позволяет формировать любые выпуклые области в пространстве входных сигналов. С помощью трехслойного перцептрона, используя «логическое или» $\vee$ 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 еще 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.

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. 3.6) 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.

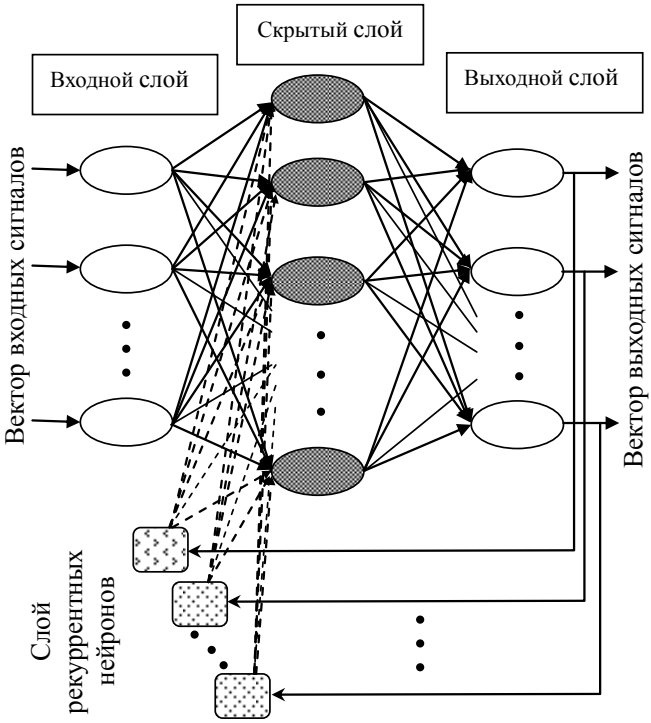


Рис. 3.6. Структура рекуррентной нейронной сети

Модель Хопфилда Американскому биофизику Дж. Хопфилду удалось привлечь к анализу нейросетевых моделей математический аппарат статистической физики. В результате была сформулирована модель ассоциативной памяти на нейронной сети с ис-

пользованием 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 использует три слоя: *входной, слой Хопфилда* и *выходной слой*. Каждый слой имеет одинаковое количество нейронов. Входы слоя Хопфилда подсоединены к выходам соответствующих нейронов входного слоя через изменяющиеся *веса соединений*. Выходы слоя Хопфилда подсоединяются ко входам всех нейронов слоя Хопфилда, за исключением самого себя, а также к соответствующим элементам в выходном слое. В режиме функционирования, сеть направляет данные из входного слоя через фиксированные веса соединений к слою Хопфилда.

Сеть Хопфилда строится с учетом 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. Введенная 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.

Сети Хопфилда получили применение на практике в основном как реализации подсистем более сложных систем. Они имеют

определенные 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 определенных затрат. В биологических нервных системах происходит освобождение от лишних связей за счет 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.

Вопросы по материалам третьей главы:

1. Общие сведения и терминология интеллектуального анализа данных.
2. Инструментарий ИАД.
3. Методы и алгоритмы ИАД.
4. Композиционные методы: бустинг, бэггинг.
5. Методы решающих функций, общая характеристика методов.
6. Деревья решений, общая характеристика методов .
7. Алгоритм CART (ID3).
8. Алгоритм C4.5

9. Алгоритм покрытия.
10. Сравнение алгоритмов построения деревьев решений
11. Искусственные нейронные сети,
12. Модель искусственного нейрона.
13. Передаточные функции и их виды.
14. Модели нейронных сетей.
15. Многослойные сети.
16. Рекуррентные сети.

Глава 4.
ЯЗЫКИ ПРОГРАММИРОВАНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Существует обширное множество языков программирования [45] и программно-инструментальных оболочек, используемых для разработки интеллектуальных систем. Каждый год появляются всё новые языки, представляющие новые концепции и/или предназначенные для эффективного решения задач из определенных прикладных областей. В качестве познавательного примера специализированных языков программирования, направленных на разработку методов и алгоритмов искусственного интеллекта остановимся на трех классических вариантах.

4.1. Функциональное программирование на языках LISP и Scheme

Функциональное программирование основывается на понятии математической функции. Чистый функциональный язык программирования не использует ни переменных, ни операторов присваивания. Отсутствие переменных делает невозможным использование итеративных конструкций, поскольку они управляются переменными. Повторение должно выполняться только с помощью рекурсии. Программы представляют собой определения функций и спецификаций применения функций, а выполнение заключается в вычислении применений функции. Выполнение функции при одних и тех же параметрах всегда приводит к одному и тому же результату. Функциональный язык содержит набор элементарных функций, набор функциональных форм для построения сложных функций из этих элементарных функций, операцию применения функции и некоторые структуры данных, используемые для представления параметров и значений, вычисленных функциями.

Первым функциональным языком был язык LISP [5, 46]. Синтаксическими элементами языка LISP являются *символьные выражения*, которые также называют *s-выражениями*. В виде *s*-выражений представляются и программы, и данные. *S*-выражение может быть атомом или списком. Атомы – это базовые синтаксиче-

ские единицы языка, включающие числа и символы. Символьные атомы, называемые также просто символами, состоят из букв, цифр и некоторых специальных символов. Список — это последовательность атомов или других списков, разделенных пробелами и заключенных в круглые скобки.

Вызовы функций в языке LISP записываются в виде списка в префиксной форме, называемой польской записью, а именно:

(<функция> <аргумент1> <аргумент2> … <аргумент*N*>)

Интерпретатор языка LISP работает в цикле, называемом *циклом чтения–оценки–печати*. Это означает, что интерпретатор выводит приглашение, считывает введенные пользователем данные, пытается их оценить, выводит полученный результат или сообщение об ошибке и снова переходит в режим ожидания ввода. Получив список, интерпретатор LISP пытается проанализировать его первый элемент как имя функции, а остальные элементы – как её аргументы. Выводимое значение является результатом применения этой функции к её аргументам. Оценивая функцию, LISP сначала оценивает её аргументы, а затем применяет функцию, задаваемую первым элементом выражения, к оценке аргументов. Если аргументы сами являются функциями, то LISP рекурсивно применяет это правило для их оценивания. Таким образом, LISP допускает вложенность вызовов функций произвольной глубины. По умолчанию оцениваются все объекты. При этом используются соглашение, что числа соответствуют сами себе. С символами могут быть связаны выражения. Связывание символов может осуществляться в результате вызова функции. При оценивании связанных символов возвращается результат связывания. Если символ не связан, то при его оценке выдается сообщение об ошибке.

Для предотвращения оценивания символа или списка используется специальная функция **quote**. Это функция зависит от одного аргумента и возвращает его без оценки. Функция quote используется для того, чтобы аргументы обрабатывались как данные, а не как оцениваемые выражения. Допускается сокращенное обозначение функции в виде символа ' (одинарной кавычки) непосредственно перед аргументом. Например, выражения **(quote (a b c)) и '(a b c)** эквивалентны и возвращают список **(a b c).**

В LISP также существует функция **eval**, позволяющая оценить переданное в качестве аргумента s-выражение. Аргумент оценива-

ется как обычный аргумент функции, однако полученный результат снова оценивается, и в качестве значения выражения **eval** возвращается окончательный результат. Например, в результате применения функции **(eval (quote (* 3 7)))** будет получено число 21, поскольку в данном случае **eval** отменяет результат выполнения функции **quote**.

Основы языка программирования Scheme. Язык Scheme является одним из диалектов языка LISP и относительно невелик. В нем используется статический обзор данных, а функции обрабатываются как сущности первого класса. Последнее означает, что функции в языке Scheme могут быть значениями выражений и элементами списков, а также могут присваиваться переменным и передаваться как параметры. Параметры функций в языке Scheme передаются по значению, так что независимо от того, что именно функция делает со своими аргументами, фактические параметры не изменяют своего значения. Кроме того, передача по значению подразумевает, что фактические аргументы вычисляются перед применением функции.

Имена в языке Scheme могут состоять из букв, цифр и специальных символов, за исключением скобок. Имена не зависят от регистра символов и не должны начинаться с цифры. Язык Scheme содержит элементарные функции для выполнения основных арифметических операций, а именно +, -, * и / соответственно для сложения, вычитания, умножения и деления.

Поскольку основной структурой данных в языке LISP/Scheme являются списки, язык включает различные функции для работы с ними. В частности, имеются функции для создания списков, выбора их частей и манипулирования ими. Для выбора элементов из списка в языке LISP/Scheme могут использоваться функции car и cdr (произносится как «куд-ер»). Функция car возвращает первый элемент заданного списка (называемый «головой» списка), а функция cdr – список (называемый «хвостом»), получаемый из заданного списка после удаления его первого элемента. Например:

Выражение	Результат
(car '(3 2))	3
(cdr '(3 2))	(2)
(car '((1 2) 3 4))	(1 2)
(cdr '((1 2) 3 4))	(3 4)

(car '(3))	3
(cdr '(3))	()

С функциями **car** и **cdr** в языке LISP/Scheme связано понятие «пара» («точечная пара»). «Пара» представляет собой двухэлементную структуру, первый элемент которой (car-поле или «голова») возвращается функцией car, а второй (**cdr**-поле или «хвост») функцией **cdr**. Для представления «пары», «голова» которой есть **h**, а «хвост» есть **t**, обычно используется точечная нотация **(h . t)**.

Для конструирования списка может использоваться двухаргументная функция **cons**. Она создает «пару», «голова» которой представляет собой первый параметр, а «хвост» соответствует второму аргументу. Для формирования с помощью функции **cons** списка элементов необходимо, чтобы её вторым параметром был какой-либо список. Функция **list** создает список, элементами которого являются переданные ей параметры, число которых может быть произвольным. Функция **length** возвращает количество элементов списка. Например:

Выражение	Результат
(cons 'a '(2 c))	(a 2 c)
(cons '(a 2) 'c)	((a 2) . c)
(cons '(a 2) '(c))	((a 2) c)
(cons '(a 2) '())	((a 2))
(list 'a '(2 c))	(a (2 c))
(list '(a 2) 'c)	((a 2) c)
(list '(a 2) '(c))	((a 2) (c))
(list '(a 2) '())	((a 2) ())
(length '((a 2) c (4 e (f))))	3

В языке LISP/Scheme существуют специальные функции, называемые *предикатными*, или *предикатами*. Предикат – это функция, возвращающая логическое значение («истина» или «ложь») в зависимости от того, удовлетворяют ли её параметры некоторому условию или свойству. В языке Scheme значение «истина» обозначается как **#t**, а «ложь» как **#f**. К предикатным функциям для числовых данных в языке Scheme относятся: = (равно), <> (не равно), > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), **even?** (четное ли число), **odd?** (нечетное ли число), **zero?** (равно ли число нулю).

Функция **eq?** позволяет проверить, эквиваленты ли между собой два символьных параметра. Если неизвестно, являются ли

атомы символьными или числовыми, то проверить их на равенство можно с помощью предиката **eqv?**. Определить, эквивалентны ли два параметра, каждый из которых может являться атомом или списком, позволяет предикат **equal?**. Например:

Выражение	Результат
(eq? 'a 'a)	#t
(eq? 'a 'abc)	#f
(eqv? 'a 1)	#f
(eqv? 1 1)	#t
(equal? 'a '())	#f
(equal? '() '())	#t
(equal? '(a (2 c) (3)) '(a (2 c) (3)))	#t
(equal? '(a (2 c) (3)) '(a (2 c) 3))	#f

Предикат **list?** позволяет проверить, является ли её параметр списком. Функция **null?** возвращает **#t**, если её параметр представляет собой пустой список. В противном случае она возвращает **#f.** Предикат **pair?** проверяет, является ли её аргумент «парой». Наконец, функции **number?** и **symbol?** позволяют определить, представляет ли их аргумент собой соответственно числовой или символьный атом. Например:

Выражение	Результат
(list? '(a 2 c))	#t
(list? 'a)	#f
(list? '())	#t
(null? '(a 2 c))	#f
(null? 'a)	#f
(null? '())	#t
(pair? '(a 2 c))	#t
(pair? 'a)	#f
(pair? '())	#f
(pair? (cons 'a 2))	#t
(number? 1.23)	#t
(number? 'a)	#f
(symbol? 'ab3)	#t
(symbol? 3)	#f

Функция **let** позволяет временно связать имена со значениями *s*-выражений. Эти имена затем могут использоваться при вычислении других выражений в контексте данной функции. Она имеет следующий общий вид:

```
(let (
    (<имя1> <выражение1>)
```

```
    (<имя2> <выражение2>)
    ...
    (<имяN> <выражениеN>)
   )
   <тело>
)
```

Каждое s-выражение связывается с указанным именем, после чего выполняются выражения, образующие тело функции. Значение последнего из них представляет собой результат функции let. Все связанные имена имеют областью действия только тело функции.

Для определения новых функций в языке LISP/Scheme используется система лямбда-обозначений в виде списка, который имеет следующий общий синтаксис:

(lambda <параметры> <тело>),

где <параметры> – имена одного или нескольких формальных параметров функции (лямбда-выражения), <тело> – набор *s*-выражений, представляющих тело функции, результат вычисления последнего из которых определяет результат применения функции. Формальные параметры могут быть заданы в виде одного символьного атома или их списка. В обоих случаях не следует предварять описание параметров кавычками, то есть предотвращать их оценивание функцией quote. Если определение лямбда-выражения содержит один формальный параметр, то это означает, что соответствующая функция может иметь любое количество фактических параметров, однако при вызове функции они преобразуются в список, который связывается с указанным аргументом. Если определение функции включает список параметров, то при её вызове должно быть передано указанное число аргументов, которые связываются с соответствующими формальными параметрами. Связанные с формальными параметрами значения не изменяются в процессе вычисления функции, что отличает их от параметров подпрограмм в императивных языках программирования.

Лямбда-выражение определяет безымянную функцию, которая может применяться так же, как именованная функция, то есть путем размещения её в начале списка, включающего фактические параметры. Например:

Выражение	Результат
((lambda (x y) (* (+ x y) 0.01)) 26 53)	0.79
((lambda (x y) (list (+ x y) (* x y))) 26 53)	(79 1378)
((lambda v (list (cdr v) (car v))) 'a 2 'c)	((2 c) a)

Для определения именованной функции в языке Scheme используется функция **define**. В простейшей форме она позволяет связать s-выражение с именем и имеет следующий общий синтаксис:

(define <имя> <выражение>),

где <имя> – название новой функции, <выражение> – **s**-выражение, связываемое с данной функцией, результат вычисления которого определяет её значение. Например:

Выражение	Результат
(define a 'alpha) a	alpha
(define tail cdr) (tail '(1 2 3 4))	(2 3 4)
(define mult10 (lambda (x) (* x 10))) (mult10 3)	30

Определение именованной функции с параметрами задается с помощью функции **define** следующим образом:

(define (<имя> <параметры>) <тело>),

где <параметры> – набор разделенных пробелами имен формальных параметров, <тело> – набор *s*-выражений, представляющих тело функции, результат вычисления последнего из которых определяет результат применения функции. Семантически использование функции define в данном виде приводит к связыванию заданного имени с лямбда-выражением с указанными параметрами и телом. Например:

Выражение	Результат
(define (parabola a b c x) (+ (* a x x) (* b x) c)) (parabola 1 2 3 -1)	2
(define (swap12 lst) (let ((h1 (car lst)) (h2 (car (cdr lst))) (t (cdr (cdr lst)))) (cons h2 (cons h1 t)))) (swap12 '(a 2 c 4 e))	(2 a c 4 e)

Для организации ветвления вычислений в языке LISP/Scheme можно использовать функции **if** и **cond**. Работа этих функций зависит от значений соответствующих условий. Следует учитывать, что в языке Scheme только значение **#f** рассматривается условными функциями как «ложь». Все остальные значения, включая символы, числа, списки (в том числе пустой) и пары, интерпретируются как «истина». Функция **if** имеет следующий вид:

(if <условие> <выражение> [<альтернатива>]).

Если результат вычисления условия функции **if** соответствует значению «истина», то оценивается заданное выражение и возвращается результат этой оценки. В противном случае возвращается результат оценки третьего аргумента, если он задан, а когда альтернатива не указана, значение функции считается неопределенным. Общий синтаксис функции **cond** приведен ниже:

```
(cond
   (<условие1> <выражение> {<выражение>})
   (<условие2> <выражение> {<выражение>})
   ...
   (<условиеN> <выражение> {<выражение>})
   [(else <выражение> {<выражение>})]
)
```

Условия функции **cond** вычисляются по одному друг за другом до тех пор, пока какое-либо из них не примет значение «истина». В этом случае вычисляются соответствующие этому условию выражения и результат последнего из них возвращается в качестве значения функции. Если все условия оказались ложными, то при наличии условия **else** оцениваются его выражения, а если его нет, то значение функции считается неопределенным. Рассмотрим примеры использования условных функций.

Выражение	Результат

(define (factorial n)
  (if (<= n 0)
  1
  (* n (factorial (- n 1))))
)
(factorial 5)
	120

(define (sum v)
  (cond
  ((number? v) v)
  ((null? v) 0)
  ((list? v) (+ (sum (car v)) (sum (cdr v))))
	14

```
      (else 0) )
    )
    (sum '(4 2 7 1))
```

Поскольку данные и программы имеют одинаковую структуру, пользовательские функции могут создавать программы «на лету» и немедленно выполнять их с помощью функции eval. Например, рассмотренную выше функцию sum можно определить следующим образом (при условии, что параметр-список не содержит вложенных списков):

```
(define (sum2 v)
  (cond
    ((number? v) v)
    ((null? v) 0)
    ((list? v) (eval (cons '+ v)))
    (else 0) )
)
```

Ниже приведены примеры других функций на языке Scheme с необходимыми комментариями.

```
; Складывает два списка как вектора, то есть первый элемент
; результата есть сумма первых элементов аргументов,
; второй элемент – сумма вторых элементов аргументов и т.д.
; Если оба аргумента являются числами, возвращает их сумму
; как число.
(define (lists-sum list1 list2)
  (cond
    ; Если оба аргумента являются числами, возвращаем их
    ; сумму
    ((and (number? list1) (number? list2)) (+ list1 list2))
    ; Если первый аргумент является числом, а второй пустым
    ; списком, возвращаем первый аргумент
    ((and (number? list1) (null? list2)) list1)
    ; Если первый аргумент является числом, а второй списком,
    ; возвращаем этот список, добавив к первому элементу
    ; первый аргумент
    ((and (number? list1) (list? list2)) (cons (lists-sum ⇒ list1 (car list2)) (cdr list2)))
    ; Если первый аргумент является числом, а второй не
    ; является списком, возвращаем первый аргумент
    ((and (number? list1) (not (list? list2))) list1)
    ; Если второй аргумент является числом, то выполняем те
    ; же действия, что и для случая, когда первый аргумент
    ; является числом
    ((and (number? list2) (null? list1)) list2)
    ((and (number? list2) (list? list1)) (cons (lists-sum ⇒ list2 (car list1)) (cdr list1)))
    ((and (number? list2) (not (list? list1))) list2)
    ; Если оба аргумента являются пустыми списками,
    ; возвращаем пустой список
```

```
    ((and (null? list1) (null? list2)) '())
    ; Если один аргумент является пустым списком, а другой
    ; списком, возвращаем этот список
    ((and (null? list1) (list? list2)) list2)
    ((and (null? list2) (list? list1)) list1)
    ; Если оба аргумента являются списками, конструируем
    ; список, первый элемент которого есть сумма
    ; первых элементов аргументов, а "хвост" есть сумма
    ; "хвостов" аргументов.
    ((and (list? list1) (list? list2))
       (cons (lists-sum (car list1) (car list2))
             (lists-sum (cdr list1) (cdr list2))))
     ; Иначе возвращаем пустой список
     (else '())
  )
)

; Функциональная форма конструкция.
; Применяет каждую функцию из заданного списка аргументов
; к указанному аргументу и возвращает полученный список
; значений. Для применения функций используется функция eval.
; Параметры:
;      flst - список имен функций или атом, представляющий имя
;             функции;
;      v - значения, к которому применяется каждая функция.
; Возвращаемый результат:
;      Список, каждый элемент которого представляет собой
;      результат применения соответствующей по порядку
;      функции из списка flst к значению, заданному в
;      параметре v.
(define (construction flst v)
  (let (
        ; Выражение, отменяющее оценку параметра v
        (qv (list 'quote v))
       )
    ; Анализируем список имен функций
    (cond
      ; Если список имен функций пуст, результат – пустой
      ; список
      ((null? flst) '())
      ; Если список имен функций задан, в качестве результата
      ; конструируем список, "голова" которого есть результат
      ; применения первой функции из списка, а "хвост"
      ; результат рекурсивного вызова от "хвоста" списка
      ; функций
      ((list? flst) (cons (eval (list (car flst) qv)) ⇒ (construction (cdr flst) v)))
      ; Если задан не список, а одно имя функции, в качестве
      ; результата конструируем список из одного элемента,
      ; соответствующего значению, полученному после
      ; применения данной функции
      ((symbol? flst) (list (eval (list flst qv))))
      ; Иначе результат - пустой список
```

```
      (else '())
      )
  )
)

; Преобразует исходный список таким образом,
; что положительные числа располагаются в начале
; результирующего списка, а остальные атомы - в конце.
(define (positive-head v)
  ; Анализируем аргумент
  (cond
    ; Если аргумент пустой список, возвращаем его
    ((null? v) '())
    ; Если аргумент (непустой) список, выделяем, анализируем
    ; и обрабатываем его "голову" и "хвост"
    ((list? v)
       (let (
            ; Объявляем необходимые локальные имена
            (h (car v))
            (t (cdr v))
            (tres (positive-head (cdr v)))
            )
         ; Анализ "головы"
         (cond
            ; Если "голова" - пустой список, возвращаем
            ; результат рекурсивного вызова функции
            ; от "хвоста"
            ((null? h) tres)
            ; Если "голова" - (непустой) список, а "хвост"
            ; - пустой список, возвращаем результат
            ; рекурсивного вызова функции от "головы"
            ((and (list? h) (null? t)) (positive-head h))
            ; Если "голова" - список и "хвост" - (непустой)
            ; список, возвращаем результат рекурсивного
            ; вызова функции от "разложенной на составляющие
            ; головы" и "хвоста"
            ((list? h) (positive-head (list (car h) ⇒
 (cdr h) t)))
            ; Если "голова" - положительное число,
            ; возвращаем список с данной "головой"
            ; и результатом рекурсивного вызова функции
            ; от "хвоста"
            ((and (number? h) (> h 0)) (cons h tres))
            ; Если "голова" - неположительное число или
            ; атом, анализируем результат рекурсивного
            ; вызова функции от "хвоста"
            (else
               ; Если обработанный "хвост" пуст,
               ; возвращаем список, включающий только
               ; "голову".
               ; Иначе, анализируем его "голову".
               ; Если это положительное число,
```

```
                ; "меняем головы местами" и вызываем
                ; рекурсивно функцию от нового "хвоста",
                ; иначе возвращаем список, включающий
                ; исходную "голову" и обработанный "хвост".
                (if (null? tres)
                    (list h)
                    (let (
                         (htr (car tres))
                         (ttr (cdr tres))
                         )
                      (if (and (number? htr) (> htr 0))
                         (cons htr (positive-head (cons h ⇒ ttr)))
                         (cons h tres))
                     )
                  )
              )
          )
        )
    )
    ; Если аргумент является атомом, возвращаем его в виде
    ; списка
    ((not (list? v)) (list v))
    ; Иначе возвращаем пустой список
    (else '())
  )
)

; Выделяет положительные числа из списка.
; Возвращает двухэлементный список, первый элемент которого –
; список найденных положительных чисел,
; а второй - список остальных атомов.
(define (extract-positives v)
  ; Анализируем аргумент
  (cond
    ; Если аргумент число, проверяем, является ли оно
    ; положительным и включаем его в соответствующий подсписок
    ((number? v) (if (> v 0) (list (list v) '()) (list '()⇒ (list v))))
    ; Если аргумент пустой список, возвращаем пустые
    ; подсписки
    ((null? v) (list '() '()))
    ; Если аргумент (непустой) список, выделяем, анализируем
    ; и обрабатываем его "голову" и "хвост"
    ((list? v)
       (let (
            ; Объявляем необходимые локальные имена
            (h (car v))
            (t (cdr v))
            (tres (extract-positives (cdr v)))
            )
        ; Анализ "головы"
        (cond
```

```
            ; Если "голова" - пустой список, возвращаем
            ; результат рекурсивного вызова функции от
            ; "хвоста"
            ((null? h) tres)
            ; Если "голова" - (непустой) список, а "хвост"
            ; - пустой список, возвращаем результат
            ; рекурсивного вызова функции от "головы"
            ((and (list? h) (null? t)) ⇒
 (extract-positives h))
            ; Если "голова" - список и "хвост" - (непустой)
            ; список, возвращаем результат рекурсивного
            ; вызова функции от "разложенной на составляющие
            ; головы" и "хвоста"
            ((list? h) (extract-positives (list (car h) ⇒ (cdr h) t)))
            ; Если "голова" - положительное число, включаем
            ; его в первый подсписок вместе с "головой"
            ; обработанного "хвоста", а во второй подсписок
            ; "хвост обработанного хвоста"
            ((and (number? h) (> h 0)) (list (cons h ⇒
 (car tres)) (car (cdr tres))))
            ; Если "голова" - отрицательное число или атом,
            ; включаем его во второй подсписок вместе
            ; с "хвостом обработанного хвоста",
            ; а в качестве первого подсписка берем
            ; "голову обработанного хвоста"
            (else (list (car tres) (cons h (car ⇒
 (cdr tres)))))
        )
      )
    )
   ; Если аргумент является нечисловым атомом, включаем его
   ; во второй подсписок
   ((not (list? v)) (list '() (list v)))
   ; Иначе возвращаем пустые подсписки
   (else (list '() '()))
  )
)

; Выделяет список всех атомов исходного списка.
(define (atom-list v)
  ; Анализируем аргумент
  (cond
    ; Если аргумент пустой список, возвращаем его
    ((null? v) '())
    ; Если аргумент (непустой) список, выделяем, анализируем
    ; и обрабатываем его "голову" и "хвост"
    ((list? v)
       (let (
            ; Объявляем необходимые локальные имена
            (h (car v))
            (t (cdr v))
```

```
            (tres (atom-list (cdr v)))
            )
        ; Анализ "головы"
        (cond
            ; Если "голова" - пустой список, возвращаем
            ; результат рекурсивного вызова функции
            ; от "хвоста"
            ((null? h) tres)
            ; Если "голова" - (непустой) список, а "хвост"
            ; - пустой список, возвращаем результат
            ; рекурсивного вызова функции от "головы"
            ((and (list? h) (null? t)) (atom-list h))
            ; Если "голова" - список и "хвост" - (непустой)
            ; список, возвращаем результат рекурсивного
            ; вызова функции от "разложенной на составляющие
            ; головы" и "хвоста"
            ((list? h) (atom-list (list (car h) (cdr h) t)))
            ; Если "голова" - атом, возвращаем список,
            ; включающий его и обработанный "хвост"
            (else (cons h tres))
        )
      )
    )
    ; Если аргумент является атомом, возвращаем его в виде
    ; списка
    ((not (list? v)) (list v))
    ; Иначе возвращаем пустой список
    (else '())
  )
)

; Преобразует исходный список таким образом,
; что положительные числа располагаются в начале
; результирующего списка, а остальные атомы - в конце.
; В данной версии используются функции extract-positives и
; atom-list.
(define (positive-head-v2 v)
  ; Анализируем аргумент
  (cond
    ; Если аргумент пустой список, возвращаем его
    ((null? v) '())
    ; Если аргумент (непустой) список, выделяем в нем
    ; положительные числа и остальные атомы
    ; и объединяем полученные списки в один
    ((list? v) (atom-list (extract-positives v)))
    ; Если аргумент является атомом, возвращаем его в виде
    ; списка
    ((not (list? v)) (list v))
    ; Иначе возвращаем пустой список
    (else '())
  )
)
```

4.2. Основы логического программирования на языке Prolog

В основе логического программирования лежит математический аппарат формальной логики, в частности, исчисление (логика) предикатов. *Предикат* представляет отношение между некоторым (в том числе нулевым) числом объектов предметной области и их свойствами. Объекты предметной области и их свойства обозначаются с помощью *термов*. Терм исчисления предикатов может быть константой, переменной или функциональным выражением. Константа – это символ, представляющий некий объект или его свойство. Переменная – это символ, который может соответствовать разным объектам или свойствам в разное время. Функциональное выражение (*составной терм*) – это символ-идентификатор функции, за которым в круглых скобках перечислены разделенные запятыми термы, представляющие её аргументы или параметры. Идентификатор функции называют также *функтором*. Составные термы соответствуют *атомарным высказываниям или предложениям* в исчислении предикатов. Атомарные предложения с помощью логических операторов $\neg$ (отрицание), $\wedge$ (конъюнкция), $\vee$ (дизъюнкция), $\equiv$ (тождественность/эквивалентность) и $\rightarrow$ (импликация) могут комбинироваться в *предложения*. Переменная, встречающаяся в предложении, представляет неопределенный объект, то есть может быть замещена любой константой. В исчислении предикатов переменные должны быть связаны одним из двух кванторов: универсальности/всеобщности ($\forall$) и существования ($\exists$), которые ограничивают значение предложения, содержащего переменную. Квантор универсальности означает, что предложение истинно для всех значений переменной. Квантор существования указывает, что предложение истинно, по крайней мере, для одного значения из области определения переменной.

Исчисление предикатов обеспечивает основу для логического вывода и доказательства теорем, то есть возможность выводить новые правильные выражения из набора истинных утверждений. Логический вывод и доказательство теорем представляет собой основу логического программирования. Одним из используемых для этого методов является резолюция. *Резолюция* – это правило

логического вывода, позволяющее получать выводимые высказывания по заданным высказываниям. Для использования принципа резолюции логические высказывания должны быть приведены к *дизъюнктивной форме*, имеющей следующий общий вид:

$\mathbf{A_1 \wedge A_2 \wedge \ldots \wedge A_m \rightarrow B_1 \vee B_2 \vee \ldots \vee B_n}$ или, сокращенно, $\mathbf{A \rightarrow B}$,

где $\mathbf{A_i}$ и $\mathbf{B_j}$ – термы. Левая часть формы называется предпосылкой или антецедентом (соответствующие термы предпосылками), а правая часть – следствием, заключением или консеквентом (соответствующие термы следствиями). Смысл логического высказывания, приведенного к дизъюнктивной форме, заключается в следующем: если истинны все предпосылки, то истинно по крайней мере одно из следствий. Ограниченный вид дизъюнктивных форм, называемый *хорновскими дизъюнктами* или *дизъюнктами Хорна*, позволяет упростить процесс логического вывода на основе резолюции. В хорновском дизъюнкте заключение либо отсутствует (хорновский дизъюнкт без головы), либо содержит единственный терм (хорновский дизъюнкт с головой).

Концепция резолюции заключается в следующем. Предположим, что заданы два следующих высказывания: $\mathbf{A \rightarrow B}$ и $\mathbf{B \rightarrow C}$. Логически из этого следует, что $\mathbf{A \rightarrow C}$. Процесс вывода этого высказывания из двух исходных представляет собой резолюцию и в упрощенном виде выглядит следующим образом. Образуется новое высказывание, левая часть которого есть конъюнкция левых частей исходных высказываний, а правая часть – конъюнкция их правых частей, то есть $\mathbf{(A \wedge B) \rightarrow (B \wedge C)}$. Затем термы, появляющиеся в обеих частях нового высказывания, удаляются из него, в результате чего получается выводимое высказывание, то есть $\mathbf{A \rightarrow C}$. Наличие переменных в высказываниях приводит к необходимости выполнять в процессе резолюции поиск значений переменных, позволяющих установить соответствие между термами. Алгоритм определения необходимых подстановок значений переменных с целью приведения в соответствие двух выражений исчисления предикатов называется *унификацией*. Временное присваивание значений переменным с целью унификации называется *конкретизацией*. Обычно во время резолюции переменная конкретизируется неким значением, не полностью удовлетворяющим требуемому соответствию. В этом случае следует отменить последнее дей-

ствие, чтобы конкретизировать эту переменную новым значением. Этот процесс возврата к предыдущему состоянию называется *бэктрекингом*.

Языки логического программирования называются *декларативными языками*, поскольку написанные на них программы состоят из объявлений, соответствующих логическим высказываниям, а не из операторов присваивания и управляющих операторов. Программирование на таких языках является *непроцедурным* в том смысле, что программы не содержат указаний, как именно вычислить результат, а лишь описывают его форму (что нужно найти). В основном в языках логического программирования описание задачи представляется с помощью исчисления предикатов, а средством вывода является метод резолюции.

Еще одна особенность логических языков программирования обусловлена их семантикой, называемой *декларативной семантикой*. Основная её концепция состоит в том, что существует простой способ определения смысла каждого оператора, вне зависимости от того, как именно этот оператор используется для решения задачи. Другими словами, смысл заданного высказывания в языке логического программирования можно точно определить по самому оператору и для этого не требуется рассмотрения контекста или последовательности выполнения программы. Поэтому декларативная семантика проще, чем семантика императивных языков, в которых для определения смысла и результата обработки оператора (например, присваивания или цикла) обычно необходимо учитывать контекст, в котором он выполняется.

Основы языка программирования Prolog [10, 34]. Язык Prolog является первым и наиболее известным на сегодняшний день языком логического программирования. Все операторы в языке Prolog образуются из *термов*. Как и в исчислении предикатов, терм может быть константой, переменной или структурой, составленной из других термов. Константа – это атом или целое число. Атом представляет собой символьную строку, состоящую из латинских букв, цифр и символов подчеркивания, начинающуюся со строчной буквы, либо строку любых символов, заключенных в апострофы. Переменная – это любая строка букв, цифр и символов подчеркивания, начинающаяся с прописной буквы. Пе-

ременные не связываются ни с какими типами с помощью объявлений. Связывание переменной со значением и типом называется *конкретизацией* и происходит только в процессе резолюции. Конкретизации осуществляются только для того, чтобы удовлетворить некую цель, представляющую собой доказательство или опровержение некоторого высказывания. *Структура* соответствует атомарному высказыванию исчисления предикатов и имеет ту же форму: функтор(терм$_1$, ..., терм$_n$). Функтор может быть любым атомом и служит для идентификации структуры. В языке Prolog структуры используются для формулирования фактов и правил. Структура устанавливает некоторое отношение между соответствующими термами. В то же время, когда структура рассматривается как запрос, она представляет собой предикат.

Программа на языке Prolog состоит из набора операторов, образующих базу данных, на основе которой логически может быть выведена новая информация. Существуют две основные формы операторов, соответствующие хорновским дизъюнктам без головы и с головой. Признаком окончания оператора служит символ точки. Простейшая форма хорновского дизъюнкта без головы в языке Prolog представляет собой отдельную структуру, интерпретируемую как факт, то есть высказывание, которое предполагается истинным. Например:

```
male(john).
male(bill).
male(jake).
female(mary).
female(shelley).
father(john, mary).
father(bill, jake).
father(bill, shelley).
mother(mary, jake).
mother(mary, shelley).
```

Оператор языка Prolog, соответствующий хорновскому дизъюнкту с головой, имеет следующий общий вид:

<структура$_0$> :- <структура$_1$>, …, <структура$_n$>,

где <структура$_0$> представляет собой заключение, а <структура$_1$>, …, <структура$_n$> – предпосылки. Разделяющая предпосылки запятая соответствует оператору конъюнкции. Высказывания могут содержать переменные. При этом подразумевается, что переменные неявно связаны

с квантором всеобщности. Хорновские дизъюнкты с головой называются *правилами*, поскольку они определяют правила логического следствия между высказываниями. Данный оператор интерпретируется, как правило, «если-то», а именно: если предпосылка является истинной или она может быть сделана истинной путем некоторой конкретизации её переменных, то следствие считается истинным. Допускается описание правил, имеющих одно и то же следствие, но разные наборы предпосылок. В этом случае предполагается неявное использование оператора дизъюнкции, то есть следствие считается истинным, если истинен хотя бы один набор предпосылок. Ниже приведены примеры правил:

```
parent(X, Y) :- father(X, Y).
parent(X, Y) :- mother(X, Y).
grandparent(X, Y) :- parent(X, P), parent(P, Y).
sibling(X, Y) :- father(F, X), father(F, Y), mother(M, X),
        mother(M, Y).
```

Факты и правила представляют сведения, необходимые для решения задачи, однако не указывают, что требуется получить. Для этого служат высказывания, называемые *целями,* или *запросами*. В языке Prolog цель имеет синтаксическую форму, эквивалентную форме хорновского дизъюнкта без головы. Определение цели соответствует формулировке теоремы, которую система должна доказать или опровергнуть на основе заданных фактов и правил. На любой корректный запрос система реализации языка Prolog должна дать ответ “yes” («да») или “no” («нет»). Ответ «да» означает, что система доказала, что цель была истинной при заданных фактах и правилах. Ответ «нет» указывает, что либо было доказано, что цель ложна, либо система неспособна ни доказать, ни опровергнуть её на основе имеющихся сведений. В качестве целей могут выступать высказывания в форме конъюнкции и высказывания, включающие переменные. При наличии переменных система пытается найти их конкретизации, делающие цель истинной. Обычно в режиме интерактивной работы соответствующие значения переменных выводятся Prolog-системой на экран в диалоговом режиме. Примеры целей и полученных результатов для рассмотренных ранее фактов и правил:

Цель	Результат
male(mary).	No
grandparent(john, shelley).	Yes
female(X).	X = mary ; X = shelley ; No
sibling(X, Y).	X = jake Y = jake ; X = jake Y = shelley ; X = shelley Y = jake ; X = shelley Y = shelley ; No
male(X), sibling(X, shelley).	X = jake ; No

Если цель включает в себя конъюнкцию фактов или структур (как в последнем примере), то они называются *подцелями*. Для доказательства того, что цель/подцель истинна, в процессе логического вывода должна быть найдена цепочка правил логического вывода и/или факты, которые связывают цель/подцель с одним или несколькими фактами в базе данных. Например, если Q - цель, то она либо должна быть найдена как факт в базе данных, либо процесс логического вывода должен найти последовательность высказываний P_1, P_2, P_3, …, P_n, такую что: P_2 :- P_1, P_3 :- P_2, …, Q :- P_n. Эта задача усложняется тем, что правила могут иметь составные правые части или переменные. Процесс поиска фактов P_i, если они существуют, в основном, сводится к сравнению или поиску соответствия между термами и обычно называется *сопоставлением*, или *удовлетворением*. Prolog-система всегда выполняет поиск в базе данных в направлении от первого оператора к последнему и удовлетворение подцелей слева направо. Наличие переменных в высказываниях приводит к необходимости использования унификации для установления соответствия между фактами.

При обработке цели с несколькими подцелями, может возникнуть ситуация, когда система не способна доказать истинность одной из подцелей. В этом случае выполняется *бэктрекинг*, то есть система восстанавливает предшествующее состояние цели, заново рассматривает предыдущую (ранее доказанную) подцель, если она есть, и пытается найти её альтернативное решение. Множественность решений для подцели обусловлена наличием различных конкретизаций её переменных. Новое решение находится в результате поиска, предпринятого с того места, где остановился

предыдущий поиск для этой подцели. Бэктрекинг требует больших затрат времени и объема памяти, поскольку он может найти все возможные решения для каждой подцели.

Рассмотрим следующую цель: **male(X), sibling(X, shelley)**. Система языка Prolog сначала ищет в базе данных первый факт с функтором male. Затем она конкретизирует переменную X параметром найденного факта, например, параметром **john**. Далее она пытается доказать, что высказывание **sibling(john, shelley)** является истинным. Если это не удается сделать, то система возвращается к первой подцели и пробует снова её удовлетворить с помощью некоторой альтернативной конкретизации переменной ***X***. Может оказаться, что выполняя резолюцию, система должна будет найти каждого мужчину в базе данных, прежде чем она найдет одного из них, являющегося братом Шелли. Кроме того, для того чтобы доказать, что цель не может быть удовлетворена, надо перебрать все возможные конкретизации переменной *X*. В данном примере поиск можно сделать более эффективным, если поменять местами подцели. Тогда только после того, как система с помощью резолюции найдет брата или сестру Шелли, она попытается доказать подцель **male(X).**

Для выполнения арифметических вычислений в языке Prolog может использоваться оператор is, имеющий следующий синтаксис:

<Переменная> is <арифметическое выражение>.

Этот дизъюнкт является истинным, если указанную переменную удалось унифицировать значением заданного выражения. Все переменные в выражении должны быть предварительно конкретизированы, а переменная в левой части оператора не должна конкретизироваться заранее. Поэтому оператор вида A is A + B не допустим. Рассмотрим пример правила, позволяющего вычислить стоимость указанного наименования товара, если известны его цена и количество. Данные о ценах и количестве представлены в форме фактов.

```
price(apples, 40).
price(bread, 12.51).
price(milk, 14.30).
price(rice, 27).
price(jam, 41.88).
qty(apples, 2.5).
qty(bread, 2).
```

```
qty(milk, 2).
qty(rice, 4).
qty(jam, 2).
cost(X, Y) :- price(X, P), qty(X, Q), Y is P * Q.
```

Язык Prolog поддерживает списки в качестве базовой структуры данных. Список представляет собой заключенную в квадратные скобки последовательность из любого количества элементов, разделенных запятыми. В качестве элементов списка могут выступать термы или другие списки. Пустой список обозначается символами **[]**. Обработка списков в Prolog основывается на унификации и рекурсии. Обращение к элементам списка осуществляется посредством специальной формы записи: **[H | T]**, где **H** - представляет «голову» (начальные элементы) списка, а **T** - его «хвост» (список остальных элементов, если они есть). Эта форма используется для реализации операций по работе со списками и может обозначать как создание списка, так и его разделение на части. Для представления в базе данных программы каких-либо сведений в виде списка может использоваться обычная структура, в которой один или несколько термов заменены списками. Например:

```
capitals([moscow, london, paris, berlin, prague]).
```

Эта структура связывает список констант [moscow, london, paris, berlin, prague] с именем отношения capitals, а не с именем соответствующей переменной. Поэтому допустимо формирование в программе еще одного высказывания с тем же именем, например: capitals([rome, madrid, athens]). Это приведет к тому, что запросы вида capitals(X) или capitals([H|T]) будут удовлетворяться двумя значениями.

Реализации языка Prolog обычно включают встроенные или библиотечные предикаты (например, **member** и **append**), обеспечивающие выполнение основных операций обработки списков, аналогичных функциям языка LISP. Отличия описаний этих предикатов от LISP-функций заключается в том, что в Prolog-программе не нужно указывать, как следует создать новый список, а надо лишь уточнить его характеристики в терминах заданных списков. Для получения нового списка в процессе резолюции Prolog-системой используется некоторый вид рекурсии.

Предикат **member** позволяет определить, принадлежит ли указанный элемент заданному списку. Он описывается следующим образом:

```
member(Element, [Element | _]).
member(Element, [_ | List]) :- member(Element, List).
```

Символ подчеркивания обозначает анонимную переменную. Эта переменная служит для указания того, что её связывание не является частью процесса вычислений и нам безразлично, что её конкретизация может быть получена путем унификации. Поскольку система языка Prolog сопоставляет высказывания по порядку, начиная с первого, условие завершения размещается до рекурсивного высказывания.

Предикат append применяется для объединения двух списков в третий и имеет следующий вид:

```
append([], List, List).
append([H | List1], List2, [H | List3]) :- append(List1, List2, List3).
```

Предикат reverse изменяет порядок следования элементов заданного списка на противоположный и унифицирует полученный результат с элементами второго списка. Он может быть определен следующим образом:

```
reverse([], []).
reverse([H | T], List):-reverse(T, Result), append(Result,[H],List).
```

В отличие от функций языка LISP, многие предикаты языка Prolog, в том числе приведенные выше, благодаря унификации могут быть использованы не только по своему прямому назначению, но и для выполнения связанных или производных операций. Например, предикат append позволяет найти список, который надо добавить к указанному, чтобы получить заданный список.

Цель	Результат
append([a, b], Y, [a, b, c, d]).	Y = [c, d] ; No
append(X, Y, [a, b, c, d]).	X = [] Y = [a, b, c, d] ; X = [a] Y = [b, c, d] ; X = [a, b] Y = [c, d] ; X = [a, b, c] Y = [d] ; X = [a, b, c, d] Y = [] ; No
reverse(X, [a, b, c, d]).	X = [d, c, b, a] ; No

Помимо того, что пользователь может управлять базой данных и порядком удовлетворения подцелей, язык Prolog позволяет использовать некоторые явные средства управления бэктрекингом. Это осуществляется с помощью *оператора отсечения*, обозначаемого знаком восклицания (**!**). В качестве подцели этот оператор

всегда достигается немедленно, но он не может быть удовлетворен повторно с помощью бэктрекинга. Таким образом, побочный эффект оператора отсечения заключается в том, что подцели, расположенные левее его в составной цели, не могут быть удовлетворены снова с помощью бэктрекинга. Поэтому, если подцели, расположенные справа от оператора отсечения, не удалось удовлетворить, то вся цель считается ложной. Например, в случае цели *a, b,* !, *c, d*, если подцели *a* и *b* достигаются, а подцель *c* - нет, то вся цель не достигается. В данном случае оператор отсечения может использоваться для того, чтобы указать, что если подцель с не достигается в первый раз, то она никогда не достигается, а значит пытаться вновь удовлетворять подцели *a* или *b* - пустая трата времени. Таким образом, назначение оператора отсечения состоит в том, чтобы позволить пользователю сделать программу более эффективной, сообщив системе, когда не следует пытаться повторно удовлетворять подцели, которые предположительно не могут дать результата в завершенном доказательстве. В результате, если найденное множество значений не приводит к решению, то поиск решения не продолжается.

Ниже приведены примеры некоторых предикатов для работы со списками.

```
% Меняет местами первый и второй элементы списка.
swap12([], []).
swap12([X], [X]).
swap12([X, Y | T], [Y, X | T]).

% Меняет местами нечетные и четные элементы списка.
% Например, результатом обработки списка [a, b, c, d, e]
% будет список [b, a, d, c, e].
swap_odd_even([], []).
swap_odd_even([X], [X]).
swap_odd_even([X, Y | T], [Y, X | R]) :- swap_odd_even(T, R).

% Расщепляет заданный список L на список положительных чисел P
% и список остальных атомов A.
extract_positives([], [], []).
extract_positives([X | T], [X | P], A) :- number(X), X > 0, ⇒ extract_positives(T, P,
A).
extract_positives([X | T], P, [X | A]) :- number(X), X =< 0, ⇒ extract_positives(T, P,
A).
extract_positives([X | T], P, [X | A]) :- not(number(X)), ⇒ extract_positives(T, P, A).
% Преобразует исходный список таким образом, что положительные
```

```
        % числа находятся в начале результирующего списка,
        % а остальные атомы - в конце.
        positive_head([], []).
        positive_head([X], [X]).
        positive_head([X, Y | T], R) :- ⇒
extract_positives([X, Y | T],  P, A), append(P, A, R).

        % Возвращает список R, включающий не более чем N первых
        % элементов исходного списка L.
        list_beginning([], _, []).
        list_beginning([_ | _], 0, []).
        list_beginning([X | _], 1, [X]).
        list_beginning([X | T], N, [X | R]) :- N > 1, M is N - 1, ⇒ list_beginning(T, M, R).

        % Возвращает список R, включающий не более чем N последних
        % элементов исходного списка L
        % в следующем порядке: последний, предпоследний и т.д.
        % Например, если L = [a, b, c, d, e], N = 3, то R = [e, d, c].
        % Используемый предикат last позволяет определить последний
        % элемент списка.
        list_ending([], _, []).
        list_ending([_ | _], 0, []).
        list_ending([X], 1, [X]).
        list_ending([_ | T], 1, [X]) :- last(T, X).
        list_ending([X | T], N, R) :- N > 1, reverse([X | T], L), ⇒ list_beginning(L, N, R).

        % Возвращает список с номерами позиций, на которых расположен
        % элемент E в списке L.
        % Нумерация элементов начинается с 1.
        % Например, если E = c, L = [a, b, c, c, b, a, c],
        % то R = [3, 4, 7].
        elem_positions(E, L, R) :- elem_positions(E, L, 0, R).

        % В elem_positions/4 3-й параметр используется в качестве
        % счетчика просмотренных элементов исходного списка L
        % и определяет номер текущего элемента. Таким образом, его
        % начальное значение определяет способ нумерация элементов,
        % то есть, например, для значения -1 элементы списка нумеруются
        % 0, 1, 2, ..., а для значения 0 - 1, 2, 3, ...
        elem_positions(_, [], _, []).
        elem_positions(X, [X], N, [M]) :- M is N + 1, !.
        elem_positions(_, [_], _, []) :- !.
        elem_positions(X, [X | T], N, [M | R]) :- M is N + 1, ⇒ elem_positions(X, T, M, R), !.
        elem_positions(X, [_ | T], N, R) :- M is N + 1, ⇒ elem_positions(X, T, M, R), !.

        % Подсчитывает количество вхождений элемента E в список L.
        elem_occur(_, [], 0).
        elem_occur(X, [X | T], M) :- ⇒
elem_occur(X, T, N), M is N + 1, !.
        elem_occur(X, [_ | T], N) :- elem_occur(X, T, N), !.
```

4.3. Язык прямого логического вывода CLIPS

CLIPS (от англ. *C Language Integrated Production System*) - программная среда для разработки экспертных систем. Синтаксис и название предложены Чарльзом Форги (Charles Forgy) в OPS (*Official Production System*). Первые версии CLIPS разрабатывались с 1984 г. в Космическом центре Джонсона (Johnson Space Center), NASA.

CLIPS является одной из наиболее широко используемых инструментальных сред для разработки экспертных систем благодаря своей скорости, эффективности и бесплатности [36,54,60]. Являясь общественным достоянием, она до сих пор обновляется и поддерживается своим изначальным автором, Гэри Райли (Gary Riley). Потомками CLIPS являются языки программирования Jess (часть CLIPS, работающая с правилами и переписанная на Java, позже развившаяся в другом направлении), ECLiPSe, Haley Eclipse, FuzzyCLIPS и другие.

Существующие версии CLIPS может эксплуатироваться на платформах UNIX, DOS, Windows и Macintosh. CLIPS включает полноценный объектно-ориентированный язык *COOL* для написания экспертных систем. Использование объектно-ориентированных средств в CLIPS позволяет значительно упростить программирование правил, поскольку для обновления данных можно применять механизм передачи и обработки сообщений методами классов. Windows-версия (clipswin.exe) не поддерживают кириллицу (консольная версия CLIPS clipsdos.exe поддерживает только кириллицу в формате UTF-8).

CLIPS разработан для применения в качестве языка прямого логического вывода и реализует модель продукционной системы.

Основная идея состоит в представлении знаний в виде структуры, состоящей из правил и фактов:

```
Правило1:
 ЕСЛИ
  (выполняются условия1)
 ТОГДА
  (выполнить действия1)
Правило2:
 ЕСЛИ
  (выполняются условия2)
 ТОГДА
  (выполнить действия2)
```

CLIPS включает в язык представления порождающих правил и язык описания процедур. Основными компонентами языка описания правил являются база фактов (fact base) и база правил (rule base). На них возлагаются следующие функции:

- база фактов представляет собой исходное состояние проблемы;
- база правил содержит операторы, которые преобразуют состояние проблемы, приводя его к решению.

Процессом помещения правил в рабочий список и их выполнением управляет машина логического вывода (МЛВ). Машина логического вывода CLIPS сопоставляет факты и правила и выясняет, какие из правил можно активизировать. Это выполняется циклически, причем каждый цикл состоит из трех шагов:

1) сопоставление фактов и правил;

2) выбор правила, подлежащего активизации;

3) выполнение действий, предписанных правилом.

Такой циклический процесс иногда называют «циклом распознавание–действие»

Интерпретатор CLIPS работает по принципу командной строки:

```
CLIPS> (<команда>)
```

Факты. Информация, на основании которой экспертная система делает логический вывод, называется **фактами**. В CLIPS есть 2 вида фактов: **упорядоченные** и **шаблонные**. Шаблонные факты имеют шаблон, задаваемый конструкцией **deftemplate**. Упорядоченные не имеют явной конструкции **deftemplate**, однако она подразумевается. В шаблонных фактах поля называются слотами и объявляются конструкцией **slot**. Например, следующий шаблон объявляет шаблон с именем *cars* и полями: *model*, *color* и *number*.

```
(deftemplate cars
 (slot model)
 (slot color)
 (slot number)
)
```

Факты размещаются в рабочей памяти. Новые факты помещаются в рабочую память командой **assert**. Например, следующая команда (assert (cars)) добавит в рабочую память **упорядоченный** факт *cars*.

Следующая команда поместит шаблонный факт с тремя атрибутами.

```
(assert
 (cars
  (model "Audi")
  (color Black)
  (number "WY 2576")
 )
)
```

В режиме интерпретатора пользователь может использовать множество команд. Факты можно включить в базу фактов прямо из командной строки с помощью команды assert, например:

```
CLIPS> (assert (today is Sunday))
CLIPS> (assert (weather is warm))
```

Для вывода списка фактов, имеющихся в базе, используется команда **facts:**

```
CLIPS>(facts)
 f-1 (today is Sunday)
 f-2 (weather is warm)
```

В некоторых версиях CLIPS , в частности, в той, которая работает в операционной среде Windows, такие команды, как **facts**, можно вызвать с помощью меню.

Для удаления фактов из базы используется команда **retract**:

```
CLIPS> (retract 1)
CLIPS> (facts)
  f-0 (today is Sunday)
```

Эти же команды , **assert** и **retract**, используются в выполняемой части правил (заключении правила) и с их помощью выполняется программное изменение базы фактов.

Команда интерпретатора **clear** очищает базу фактов.

```
CLIPS> (clear)
CLIPS> (facts)
```

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 **deffacts** имеет формат, аналогичный выражениям в языке LISP. Выражение начинается с команды **deffacts**, затем приводится имя списка фактов, а за ним следуют элементы списка. Этот массив фактов можно затем удалить из базы командой **undeffacts**:

```
CLIPS> (undeffacts today)
```

Выражение **deffacts** можно вводить и в командную строку интерпретатора, а можно записать его в текстовый файл с помощью редактора CLIPS или любого другого текстового редактора. Загрузить этот файл в дальнейшем можно с помощью команды в меню **File** либо из командной строки:

```
CLIPS> (load "my file")
```

Однако после загрузки файла факты не передаются сразу же в базу фактов CLIPS. Команда **deffacts** просто указывает интерпретатору, что существует массив **today**, который содержит множество фактов. Собственно сама загрузка выполняется командой **reset**:

```
CLIPS> (reset)
```

Команда reset сначала очищает базу фактов, а затем включает в неё факты из всех раннее загруженных массивов.

Правила. Знания предметной области представляются в CLIPS в виде правил. Активация правила – это помещение правила в рабочий список правил или agenda в CLIPS. Чтобы активированные правила выполнились нужно выполнить команду (run).

В языке CLIPS реализован следующий формат:

```
(defrule <имя правила>
    < необязательный комментарий >
    < необязательное объявление >
    < предпосылка_1 >
    ……
   < предпосылка_m >
=>
   < действие_1 >
   ……..
   < предпосылка_n >
)
Например:
(defrule chores
     “Things to do on Sunday”
     (salience 10 )
     (today is Sunday)
     (weather is warm)
=>
     (assert (wash car))
     (assert (chop wood)
)
```

В этом примере **Chores** – произвольно выбранное имя правила. Предпосылки в условной части правила

```
(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. Приведенный в тексте правила комментарий

“Things to do on Sunday” // “Что делать в воскресенье”

поможет в дальнейшем вспомнить, чего ради это правило включено в программу. Выражение

```
(salience 10)
```

указывает на степень важности правила. Пусть например, в программе имеется другое правило

```
(defrule fun
   “Better things to do on Sunday”
   (salience 100)
   (today is Sunday)
   (weather is warm)
 =>
   (assert (drink beer))
   (assert (play guitar))
)
```

Поскольку предпосылки обоих правил одинаковы, то при выполнении оговоренных 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 [-10000, 10000].

Если параметр **salience** в определении правила опущен, ему по умолчанию присваивается значение 0.

Обычно в определении правила присутствуют и переменные. 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).
```

Отметим, что факт

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

будет сопоставлен с любым из представленных ниже образцов

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

Префикс **$?** является признаком сегментной переменной, которая будет связана с сегментом списка. Например, в приведенном выше примере переменная **$?when** будет связана с

```
(on Sunday)
```

Если за префиксами ? и $? не следует имя переменой, они рассматриваются как универсальные символы подстановки, которым соответственно может быть сопоставлен любой элемент или сегмент списка.

Для определения фактов можно использовать не только списочные структуры, но и шаблоны, которые напоминают простые записи. Каждое определение шаблона состоит из произвольного имени шаблона, необязательного комментария и некоторого количества определений слотов. Слот включает поле данных, например, **name**, и тип данных, например, **STRING**. Можно указать и значение по умолчанию. Пример:

```
(deftemplate student “a student record”
    (slot name (type STRING))
    (slot age (type NUMBER) (default 18))
)
```

Если в программу включено приведенное выше определение шаблона, то выражение

```
(deffacts students
     (student (name fred))
     (student (name freda) (age 19))
)
```

приведет к тому, что в базу фактов после выполнения команды **reset** будет добавлено

```
(student (name fred) (age 18))
(student (name freda) (age 19))
```

Определение функций. В языке CLIPS функции конструируются примерно так же, как в языке LIPS. Существенное отличие состоит в том, что переменные должны иметь префикс **?**, как это показано в приведенном ниже определении:

```
(deffunction hypotenuse (?a ?b)
```

```
  (sqrt (+ (* ?a ?a) (* ?b ?b))
)
Формат определения функции в CLIPS следующий:
(deffunction <имя функции> (<аргумент> …..<аргумент>)
   <выражение>
   …….
   <выражение>
)
```

Функция возвращает результат последнего выражения в списке.

Вопросы по материалам четвертой главы:

1. Основы синтаксиса языка LISP.
2. Концептуальная схема работы LISP-интерпретатора.
3. Функции quote и eval.
4. Функции языка Scheme для работы со списками и предикатами.
5. Основные понятия логического программирования.
6. Резолюция, понятия унификации, конкретизации, бэктрекинга.
7. Основные особенности логических языков программирования.
8. Синтаксис языка Prolog.
9. Структура программы на языке Prolog, факты, правила и цели.
10. Особенности процедуры резолюции в языке Prolog.
11. Средства для работы со списками в языке Prolog.
12. Инструментально-языковая система CLIPS, назначение, характеристики, особенности организации.
13. Команды управления интерпретацией CLIPS.
14. Определение, описание фактов и правил в языке CLIPS

ЗАКЛЮЧЕНИЕ

Проблематика интеллектуальных информационных систем поднимает ряд вопросов, решение которых в плоскости практических воплощений еще только предстоит найти: что такое интеллект, что такое знание и информация, что такое мышление и смысл как формализовать эти понятия до уровня реализуемых алгоритмов.

Развитие новых информационных технологий, в том числе всемирной компьютерной сети, настоятельно требует предоставления новых возможностей получения актуальной информации и знаний, которые предприятия и физические лица могут использовать в своей деятельности. Появилось новое направление «интеллектуальный интернет», «интеллектуальный поиск».

Развитие современного общества привело к появлению крупнейших информационных банков в различных отраслях научно-технических знаний. Попытки разработки и внедрения многокритериальных систем классификации информации, а также автоматизации информационного поиска пока не приводят к сколько-нибудь существенным результатам с точки зрения увеличения релевантности и комфортности информационного поиска. Сверхбольшие объёмы хранимой информации приводят к развитию методов интеллектуального анализа данных (англ. Big Data) с учетом экономного, эффективного использования вычислительных ресурсов.

Постоянно растущие объемы и дифференциация используемой в различных областях человеческой деятельности информации затрудняют принятие эффективных управленческих решений и обуславливают необходимость разработки и применения специализированных интеллектуальных систем анализа данных и поддержки принятия решений, учитывающих специфику предметных областей.

Среди вопросов и задач обработки и анализа информации, от решения которых непосредственно зависит качество реализуемого управления, одно из центральных мест занимает комплекс проблем классификации и систематизации данных, в числе которых выделяется проблема обучения распознаванию образов. К настоя-

щему времени разработано большое количество разнообразных методов и систем распознавания, позволяющих достаточно эффективно решать те или иные задачи классификации. Вместе с тем, область применения существующих методов, как правило, ограничивается одной или несколькими категориями задач, а действующие на практике системы распознавания в большинстве своем не могут быть адаптированы и расширены для решения новых типов задач. Все это сужает возможность использования имеющихся наработок в постоянно меняющихся условиях практических задач промышленности, производства, медицины и т.д. В связи с этим особое значение приобретают проблемы повышения эффективности применения существующих методов прогнозирования, распознавания и создания адаптируемых систем распознавания.

Уровень современного развития информационных технологий предъявляет повышенные требования к качеству и надежности интеллектуальных подсистем и систем, а также к скорости их разработки и затратам на изготовление. Эти тенденции обуславливают необходимость решения целого ряда проблем, связанных с технологией изготовления ИС, в том числе задачами формализации предметной области, проектирования, моделирования и пр., как составной части процесса инжиниринга.

Разнообразие существующих подходов, методов и средств информационного и технологического обеспечения компьютерных информационных систем позволяет определить цели научного поиска, показывает, что имеются значительные резервы повышения эффективности создания, повышения качества и уровня развития автоматизации и информатизации за счет внедрения новейших автоматизированных интеллектуальных систем и инструментария интеллектуального анализа данных. Решение проблем, связанных с разработкой таких систем, лежит в сфере исследований существующих методов и средств анализа и создания прикладных интеллектуальных систем различного назначения.

На сегодняшний день актуальными являются вопросы адаптации, диагностики, методов прогнозирования, технологий извлечения знаний, интеллектуального моделирования, человеко-машинных интерфейсов, создание нейрокомпьютеров и пр. Таким образом, интеллектуальная информационная технология – это ди-

намически развивающаяся проблемная область информатики, имеющая свою функциональную основу и прикладную надстройку.

Важным направлением современной науки является создание законченной теории интеллектуальных информационных систем, объединяющей в себе технические, экономические, семиотические, биокибернетические и социально-организационные типы систем. С учетом масштабности и сложности задач создания и применения современных информационных систем можно считать, что разработка методов, алгоритмов и программных средств синтеза автоматизированных интеллектуальных информационных систем и решение на их основе задач управления, контроля, анализа данных, поиска, выбора, обучения и принятия решений являются целым комплексом первостепенных проблем, представляющих сферу исследований не одному поколению ученых.

ЛИТЕРАТУРА

1. Аналитическая платформа Deductor. BaseGroup Labs. URL: http://www.basegroup.ru/ (дата обращения: 07.10.2012).
2. *Андрейчиков А.В., Андрейчикова О.Н.* Интеллектуальные информационные системы. – М.: Финансы и статистика, 2004. – 424 с.
3. *Андрейчикова О.Н.* Системный анализ и синтез стратегических решений в инноватике. Математические, эвристические и интеллектуальные методы системного анализа и синтеза инноваций. – М.: Либроком, 2012. – 308 с.
4. *Арлазаров В., Емельянов Н.* Обработка изображений и анализ данных. – М.: Либроком, 2008. – 368 с.
5. Ассоциация пользователей Лисп. URL: lisp.org (дата обращения: 02.05.2013).
6. *Башмаков А.И., Башмаков И.А.* Интеллектуальные информационные технологии. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2005. – 304 с.
7. *Боровиков В.П.* Нейронные сети. Statistica Neural Networks. Методология и технологии современного анализа данных. – М.: Горячая Линия – Телеком, 2008. – 392 с.
8. *Брускин С.Н., Довженко А.Ю.* Интеллектуальный анализ динамики бизнес-систем. – М.: Инфра-М, 2010. – 320 с.
9. *Вапник В.Н., Червоненкис А.Я.* Теория распознавания образов (статистические проблемы обучения). – М.: Наука, 1974. – 416 с.
10. Введение в Акторный Пролог Алексей А. Морозов. URL: http://www.cplire.ru/Lab144/start/r_index.html (дата обращения: 02.05.2013).
11. Гаврилова Т.А., Червинская К.Р. Извлечение и структурирование знаний для экспертных систем. – М.: Радио и связь, 1992. – 200 с.
12. *Гаврилова Т.А., Хорошевский В.Ф* Базы знаний интеллектуальных систем. – СПб.: Питер, 2000. – 384 с.
13. *Гаврилов А.В.* Гибридные интеллектуальные системы. – Новосибирск: НГТУ, 2002. – 142 с. URL: http://www.insycom.ru/html/Articles/2003/Monograf.pdf (дата обращения: 06.10.2012).
14. *Гаскаров Д.В.* Интеллектуальные информационные системы. – М.: Высш. школа, 2003. – 431 с.
15. *Гаскаров Д.В., Сикулер Д.В., Фомина И.К.* Интеллектуальные информационные системы: Интеллектуальная информационная технология. Экспертные системы. – СПб.: СПГУВК, 2004. – 362 с.
16. *Горелик А.Л., Скрипкин В.А.* Методы распознавания. – М.: Высш. школа, 1989. – 232 с.
17. *Джексон П.* Введение в экспертные системы / Пер. с англ. – М.: Вильямс, 2001. – 624 с.
18. *Джонс М.Т.* Программирование искусственного интеллекта в приложениях / Пер. с англ. – М.: ДМК Пресс, 2006. – 312 с.
19. *Дюк В.А.* Инструментальные средства интеллектуального анализа данных. – СПб.: Изд-во РГПУ им. А.И. Герцена, 2012. – 161 с.

20. *Зайцев О.* «Технологии рассылки спама и методы защиты от него». URL: http://www.compress.ru/article.aspx?id=17269&iid=799 (дата обращения: 10.10.2012).
21. ЗАО «Фо́рексис». URL: www.forecsys.ru (дата обращения: 04.10.2012).
22. *Иванюкович Г. А.* Статистический анализ экогеологических данных. Практикум решения задач с помощью пакета программ Statistica. – СПб.: СПбГУ, 2010. – 204 с.
23. Искусственный интеллект: В 3 кн. Кн. 1. Системы общения и экспертные системы: Справочник / Под ред. Э.В. Попова. – М.: Радио и связь, 1990. – 464 с.
24. Искусственный интеллект: В 3 кн. Кн. 2. Модели и методы: Справочник / Под ред. Д.А. Поспелова. – М.: Радио и связь, 1990. – 304 с.
25. Искусственный интеллект: В 3 кн. Кн. 3. Программные и аппаратные средства: Справочник / Под ред. В.Н. Захарова, В.Ф. Хорошевского. – М.: Радио и связь, 1990. – 368 с.
26. *Корнеев В.В., Гареев А.Ф., Васютин С.В., Райх В.В.* Базы данных. Интеллектуальная обработка информации. – М.: Нолидж, 2000. – 352 с.
27. *Криницкий Н.А., Миронов Г.А., Фролов Г.Д.* Автоматизированные информационные системы / Под ред. А.А. Дородницина. – М.: Наука, 1982. – 384 с.
28. *Лорьер Ж.-Л.* Системы искусственного интеллекта / Пер. с франц. – М.: Мир, 1991. – 568 с.
29. Логическая модель знаний. URL: http://www.aiportal.ru/articles/knowledge-models/logical-model.html (дата обращения: 02.05.2013).
30. *Любарский Ю.Я.* Интеллектуальные информационные системы. – М.: Наука, 1990. – 232 с.
31. *Люгер Дж. Ф.* Искусственный интеллект: стратегии и методы решения сложных проблем. 4-е изд. / Пер. с англ. – М.: Изд. дом «Вильямс», 2005. – 864 с.
32. *Макленнен Д., Танг Ч., Криват Б.* Microsoft SQL Server 2008: Data Mining. Интеллектуальный анализ данных / Пер. с англ. – СПб.: BHV, 2009. – 720 с.
33. *Марманис Х., Бабенко Д.* Алгоритмы интеллектуального Интернета. Передовые методики сбора, анализа и обработки данных / Пер. с англ. – Символ-Плюс, 2011. – 480 с.
34. Международный стандарт ISO для языка Пролог. URL: http://people.sju.edu/~jhodgson/wg17/wg17web.html (дата обращения: 02.05.2013).
35. НейрОКТехСофт. URL: http://www.neurokts.com (дата обращения: 04.10.2012).
36. Образовательный портал Claw.ru. Язык Clips. URL: www.claw.ru (дата обращения: 04.05.2013).
37. *Поспелов Д.А.* Моделирование рассуждений. Опыт анализа мыслительных актов. – М.: Радио и связь, 1989. – 184 с.
38. Построение экспертных систем / Под ред. Ф. Хейеса-Рота, Д. Уотермана, Д. Лената / Пер. с англ. – М.: Мир, 1987. – 441 с.
39. Продукционная модель представления знаний. URL http://itteach.ru/predstavlenie-znaniy/produktsionnaya-model-predstavleniya-znaniy (дата обращения: 03.05.2013).
40. Профессиональный информационно-аналитический ресурс, посвященный машинному обучению, распознаванию образов и интеллектуальному анализу

данных. URL: http://www.machinelearning.ru (дата обращения: 02.05.2013).
41. *Ракитов А.И.* Философия компьютерной революции. – М.: Политиздат, 1991. - 285 с.
42. *Рассел С., Норвиг П.* Искусственный интеллект: современный подход. 2-е изд. / Пер. с англ. - М.: Изд. дом «Вильямс», 2006. - 1408 с.
43. *Романов В.П.* Интеллектуальные информационные системы в экономике. - М.: «Экзамен», 2003. - 496 с.
44. Сикулер Д.В. Интеллектуальные автоматизированные информационные системы. – СПб.: «ПаркКом», 2008. - 135 с.
45. *Сикулер Д.В., Фомин В.В.* Технологии, методы и языки программирования. – СПб.: Изд-во Политехн. ун-та, 2012. - 166 с.
46. Сообщество Лисп-программистов. URL: www.Lisp.ru (дата обращения: 02.05.2013).
47. Технологии анализа данных. Data Mining, Visual Mining, Text Mining, OLAP / А.А. Барсегян, М.С. Куприянов, В.В. Степаненко, И.И. Холод. - СПб.: БХВ-Петербург, 2007. - 384 с.
48. Универсальная статистическая диалоговая система STADIA. Научно-производственное общество "Информатика и компьютеры". URL: http://statsoft.msu.ru/products.htm (дата обращения: 07.10.2012).
49. *Уткин Л.В.* Анализ риска и принятие решений при неполной информации. - СПб.: Наука, 2007. – 404 с.
50. *Фомин В.В., Фомина И.К., Пшеницын В.И.* Теоретические основы автоматизированных информационных систем и процессов / Под ред. В.В. Фомина. - СПб.: СПГУВК, 2006. - 288 с.
51. *Фомина И.К.* Интеллектуальные информационные системы. - СПб.: СПГУВК, 2009. - 198 с.
52. *Фомина И.К.* Методическое обеспечение дисциплины «Интеллектуальные информационные системы». - СПб.: СПГУВК, 2006. - 145 с.
53. *Фомина И.К., Фомин В.В.* Интеллектуальные информационные системы: учебно-методическое пособие (лабораторный практикум). - СПб.: СПГУВК, 2012. - 69 с.
54. A Tool for Building Expert Systems. URL: http://clipsrules.sourceforge.net/ (дата обращения: 04.05.2013).
55. Bime. Modern Interactive Data Analysis and Dashboards. URL: http://bimeanalytics.com (дата обращения: 07.10.2012).
56. Data Applied, Products. URL: http://www.data-applied.com
57. ata-mining. URL: http://www.kdnuggets.com/ (дата обращения: 07.10.2012).
58. Megaputer Intelligence (Polyanalyst 6.0 – Технология анализа данных). URL: http://megaputer.ru/ (дата обращения: 04.05.2013).
59. The UC Irvine Machine Learning Repository URL: http://archive.ics.uci.edu/ml/ (дата обращения: 03.05.2012).
60. This is GNU CLISP – an ANSI Common Lisp Implementation. URL: http://www.clisp.org/
61. *Zdravko Markov, Ingrid Russell.* An Introduction to the WEKA Data Mining System URL: http://www.cs.ccsu.edu/~markov/weka-tutorial.pdf (дата обращения: 07.10.2012).

Учебное издание

Фомин Владимир Владимирович
Миклуш Виктория Александровна

ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ

Учебное пособие

Редактор О.С. Крайнова
Компьютерная верстка Н.И. Афанасьевой

ЛР № 020309 от 30.12.96

Подписано в печать 31.10.13. Формат 60×90 $^{1}/_{16}$. Гарнитура Times New Roman.
Печать цифровая. Усл. печ. л. 9,4. Тираж 200 экз. Заказ № 237.
РГГМУ, 195196, Санкт-Петербург, Малоохтинский пр., 98.
Отпечатано в ЦОП РГГМУ