Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Севастопольский государственный университет»

Программирование на языке Пролог для систем искусственного интеллекта

Методические указания к лабораторным работам по дисциплине
“Методы и системы искусственного интеллекта”
для студентов дневной и заочной форм обучения
направлений: 09.03.02 — “Информационные системы и технологии”
09.03.03 — “Прикладная информатика”

Севастополь
2025

УДК 004.8 (075.8)

Программирование на языке Пролог для систем искусственного интеллекта : методические указания к лабораторным работам по дисциплине “Методы и системы искусственного интеллекта” для студентов дневной и заочной формы обучения направлений 09.03.02 — “Информационные системы и технологии”, 09.03.03 — “Прикладная информатика”/ СевГУ ; сост. **В. Н. Бондарев**, **Т.И. Сметанина —** Севастополь : Изд-во СевГУ, 2025. **—** 64с.

Методические указания предназначены для проведения лабораторных занятий по дисциплине “Методы и системы искусственного интеллекта” у студентов дневной и заочной форм обучения направлений: 09.03.02 — “Информационные системы и технологии”, 09.03.03 — “Прикладная информатика”. **Целью методических указаний** является обучение студентов основным методам и моделям искусственного интеллекта при реализации их средствами языка Пролог.

Методические указания составлены в соответствии с требованиями программы дисциплины “Методы и системы искусственного интеллекта” для студентов дневной и заочной форм обучения направлений: 09.03.02 — “Информационные системы и технологии”, 09.03.03 — “Прикладная информатика”.

Утверждены на заседании кафедры Информационных систем (протокол № от 202 г.)

Рецензент: Брюховецкий А.А., к.т.н., доцент кафедры информационных технологий и компьютерных систем

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дисциплина «Методы и системы искусственного интеллекта» относится к числу нормативных дисциплин цикла профессиональной подготовки. Целью преподавания дисциплины является обучение студентов основным принципам построения компьютерных систем, способных выполнять функции, традиционно считающиеся интеллектуальными.

Лабораторный практикум по дисциплине состоит из двух частей. Первая часть включает решение задач искусственного интеллекта (ИИ) на языке Python, а вторая — на языке Пролог.

В настоящих методических указаниях рассматриваются: основы логического программирования на языке Пролог; применение языка Пролог для организации динамических баз данных, поиска решений логических задач на основе распространения ограничений, построения экспертных систем продукционного типа.

Все работы, приведенные в методических указаниях, выполняются за 4 академических часа.

Варианты заданий студенты выбирают самостоятельно по номеру зачетной книжки. Номер варианта для каждой лабораторной работы определяется как остаток от деления двух последних цифр номера зачетки на 25. Например, две последние цифры 37. Остаток от деления 37 на 25 равен 12. Следовательно, номер варианта задания 12.

Студенты заочной формы обучения выполняют данные лабораторные в два этапа. **На первом этапе**, выполняемом самостоятельно, студенты оформляют решение соответствующих вариантов заданий в виде контрольной работы. Контрольная работа оформляется в тетради или на листах формата А4. Контрольная работа должна содержать: вариант задания; краткие теоретические сведения, относящиеся к решаемой задаче; программу на языке Пролог, решающую поставленную задачу; описание программы. Срок сдачи контрольной работы студентами заочниками — 12-ая неделя семестра.

На втором этапе, выполняемом в течение лабораторно-экзаменационной сессии, студенты в лабораториях кафедры осуществляют ввод и отладку программ на ЭВМ, демонстрируют их выполнение, анализируют полученные результаты и отвечают на вопросы преподавателя.

Лабораторные работы ориентированы на применение эдингбургского синтаксиса языка Пролог. В частности, рекомендуется использовать свободно распространяемый **SWI-Prolog**, который поддерживает работу, как в указанном стандарте синтаксиса, так и в стандарте ISO [3,10].

1. ЛАБОРАТОРНАЯ РАБОТА № 1 «СОЗДАНИЕ ДИНАМИЧЕСКИХ БАЗ ДАННЫХ»

1.1. Цель работы

Изучение технологии подготовки и выполнения Пролог-программ в интегрированной среде, исследование способов организации динамических баз данных (БД) средствами языка Пролог.

1.2. Краткие теоретические сведения

1.2.1 Введение в Пролог

Программа на языке Пролог состоит из фактов, правил и целевых утверждений. *Факт* представляет собой истинное утверждение. *Правило* представляет собой утверждение, которое истинно при определенных условиях. Совокупность фактов и правил образует *базу данных Пролога*.

Когда база данных загружена в память Пролог-системы, к ней можно обращаться с вопросами, формулируемыми в виде *целевых утверждений*.

Примеры простейших фактов, правил и целевых утверждений, а также основы работы в интегрированной среде изложены в методических указаниях [3].

1.2.2. Объекты данных Пролога

Все объекты данных языка Пролог представляют собой термы [1, 2]. *Терм* может быть *константой, переменной* или *структурой (составным* термом). *Константы* подразделяются на *числа и атомы*. *Атомы* — это символьные константы, которые начинаются строчной буквой или заключаются в одинарные кавычки, или состоят из специальных символов. Примеры атомов: **сергей, 'Сергей'. ===>.**

Имена переменных начинаются с заглавных букв или символа подчеркивания "_". Областью действия переменной является утверждение (факт или правило). Переменные, которым присвоены значения, называются *конкретизированными*. Существуют *анонимные* переменные — переменные без имени. Они обозначаются символом подчеркивания. Каждая анонимная переменная уникальна, т.е. отличается от всех других анонимных переменных в утверждении.

Структура (составной терм) является объектом, состоящим из нескольких компонент. Она состоит из атома, который называется *функтором*, и последовательности термов. Например, **f(T1,T2,T3)** — структура, состоящая из функтора **f** и трёх компонент **T1,T2,T3** (термов). Число компонент структуры называется *арностью.*

Рассмотрим структуру, представляющую информацию о некотором сотруднике:

```
сотрудник( фио('Петренко','Сергей', 'Иванович'),
           рабочее_место('Отдел 4', 'инженер'),
           адрес(улица('Тенистая',7),город('Ростов')),
           телефон('68-23-42')).
```

Структура **сотрудник** содержит вложенные подструктуры (термы): **фио, рабочее_место, адрес, телефон**. В свою очередь, подструктура **адрес** состоит из двух подструктур: **улица** и **город.** Таким образом, все объекты данных Пролога **—** это термы, компонентами которых являются другие термы [1,2].

Широко используемым объектом данных языка Пролог является список. *Список* состоит из произвольного числа элементов, заключаемых в квадратные скобки и разделяемых запятыми. Например: **[a, b, c, d]**. Для приведенного списка элемент **a** — это голова списка, а подсписок **[b, c, d]** — хвост списка.

Для представления списка в виде структуры, состоящей из головы и хвоста, в Прологе широко используется еще одно обозначение, в котором голова и хвост списка отделяются вертикальной чертой: **[Голова | Хвост]**.

1.2.3. Сопоставление (унификация) термов

Сопоставление — это процесс, на вход которого подаются два терма, а он проверяет, соответствуют ли эти термы друг другу. Если термы не сопоставимы, то процесс терпит *неудачу*. Если термы сопоставимы, то процесс находит конкретизацию переменных, делающую эти термы тождественными, и завершается *успешно*.

Возможность явного сопоставления двух термов проверяется с помощью оператора “**=**”. Например, сопоставление

? – отец('Иван',P) = отец(O,'Сергей').

закончится успехом при следующей конкретизации переменных: **O='Иван'; P='Сергей'**.

В общем, два терма сопоставляются по следующим правилам [1]:

а) если термы **X** и **Y** — константы, то они сопоставимы, только когда одинаковы;

б) если терм **X** представлен константой или структурой, а терм **Y** — не конкретизированной переменной, то термы **X** и **Y** сопоставимы, и в **Y** подставляется значение **X**;

в) если термы **X** и **Y** — структуры, то они сопоставимы, когда у них совпадают главные функторы и арность, а также сопоставимы соответствующие компоненты структуры.

Поиск решений в пролог-системах протекает в автоматическом режиме с использованием принципа возврата к альтернативным вариантам возможных сопоставлений.

1.2.4. Управление выполнением пролог-программ

При рассмотрении пролог-программы полезно выделять два уровня ее смысла: декларативный и процедурный. *Декларативный* смысл пролог-программы связан с отношениями, объявленными (декларируемыми) в программе, он определяет, достижимо ли целевое утверждение, и при каких значениях переменных

оно будет верным. *Процедурный* смысл определяет, как пролог-система обрабатывает отношения пролог-программы, каким образом пролог-система отвечает на вопросы.

Рассмотрим правило **P:- Q, R.** Запятая между условиями **Q** и **R** обозначает *конъюнкцию*. Декларативная интерпретация правила может быть следующей: **P** истинно, если условия **Q** и **R** истинны. А процедурный вариант можно сформулировать так: чтобы решить задачу **P** , сначала реши подзадачу **Q** , а затем — подзадачу **R** . Таким образом, процедурная интерпретация фиксирует порядок, в котором обрабатываются подцели **Q** и **R.**

Дизъюнкция условий обозначается точкой с запятой. Например: **P:- Q ; R.** В этом случае пролог система должна оценить верность условия **Q** или **R**. Чтобы доказать **P** пролог-система сначала предпримет попытку доказательства **Q**. Если доказательство **Q** завершится неудачей (**fail)**, то пролог-система вернется к точке выбора вариантов, удалит все сделанные подстановки и перейдет к доказательству альтернативного утверждения **R.** Однако она перейдет к доказательству **R** и в том случае, когда **Q** верно. Так как в процессе доказательства выполняются подстановки в переменные, то доказательство альтернативного утверждения обеспечивает нахождение дополнительных вариантов подстановок, которые могут интересовать пользователя**.**

Если проверку альтернативного условия требуется исключить, то применяют встроенный предикат *отсечения*, который обозначается знаком “ **!** ”. Этот предикат всегда выполняется успешно и стирает все альтернативные ветви в пределах утверждения, в котором он введен. Например: **P:- Q, ! ; R.** В этом случае при успешной попытке доказательства **Q** предикат отсечения сотрёт точку выбора альтернативного условия **R** и оно не будет проверяться.

Для управления процессом обработки условий также широко применяется встроенный предикат **fail** , который обеспечивает создание *состояния искусственной неудачи.* Состояние искусственной неудачи заставляет пролог-систему возвращаться к имеющимся точкам выбора и искать другие варианты доказательства утверждений. Точки выбора могут формироваться предикатами, допускающими альтернативные подстановки (такие предикаты называют *недетерминированными*), или создаваться искусственно с помощью специальных предикатов.

Таким специальным предикатом является предикат **repeat** [1,2]. Предикат определяется рекурсивно: **repeat:-true; repeat.** Благодаря этому он создаёт бесконечное число точек выбора. Его часто применяют совместно с **fail** для организации циклов, которые называются циклами **repeat-fail.** Общая схема организации таких циклов следующая:

```
цикл:- repeat,
       (<проверка условия выхода из цикла>, !;
        <тело цикла>, fail).
```

Здесь скобки образуют *группу* и тем самым ограничивают область действия дизъюнкции. Если условие завершения цикла выполняется успешно, то предикат отсечения стирает все точки выбора и цикл завершается. Иначе выполняется тело

цикла, а предикат **fail** обеспечивает возврат к точкам выбора, создаваемым с помощью **repeat,** и действия повторяются.

1.2.5. Способы представления базы данных

Существует несколько простых способов представления реляционных баз данных на языке Пролог [6]. Первый способ основан на представлении целостных информационных элементов, т.е. записей (кортежей) базы в виде множества фактов. Например:

```
%          N      Фам.      Имя      Отч.      Отд.  Должн.     Филиал  Тел.
сотрудник(1001,  петренко, сергей, иванович,  4,    инженер,   Ялта,   68-23-42).
```

Совокупность всех таких записей (кортежей) образует отношение **сотрудник.**

Второй способ состоит в представлении базы данных в виде множества фактов, которые устанавливают отношения между отдельными атрибутами записей. Один из атрибутов при этом должен выступать в роли ключа, объединяющего все остальные атрибуты:

```
фамилия(1001, петренко).
имя(1001, сергей).
отчество(1001, иванович).
отдел(1001,4).
должность(1001,инженер).
филиал(1001, ялта).
телефон(1001, 68-23-42).
```

Здесь в качестве ключевого атрибута используется табельный номер сотрудника. Все атрибуты сотрудника можно объединить в отношение **сотрудник** при помощи правила:

```
сотрудник(N, Фам, Имя, Отч, Отд, Должн, Филиал, Тел):-
        фамилия(N, Фам), имя(N, Имя), отчество(N, Отч),
        отдел(N, Отд), должность(N, Должн),
        филиал(N, Филиал), телефон(N, Тел).
```

Представление базы данных в виде отдельных атрибутов является более гибким, чем применение целостных записей, поскольку новые атрибуты можно добавлять без переписывания существующей базы данных.

Третий способ основан на представлении базы данных в виде списка структур. Каждый элемент списка — это отдельная запись базы данных:

```
%      N      Фам.      Имя      Отч.       Отд.  Должн.     Филиал  Тел.
[сотр(1001,  петренко, сергей,  иванович,  4,    инженер,   ялта,   68-23-42),
 сотр(1002,  никулин,  николай, васильевич, 3,    начальник, керчь,  11-24-47),
 сотр(1003,  павлов,   сергей,  николаевич, 2,    оператор,  керчь,  11-23-42)]
```

Если в первом и во втором способах база данных является частью программы, и для работы с ней в значительной степени используются встроенные поисковые механизмы пролог-системы, то в третьем случае база данных отделена от

программного кода и все поисковые процедуры для работы с такой базой должны определяться программистом самостоятельно.

1.2.6. Списки и рекурсия

Над списками выполняют следующие операции: добавление элемента в список, удаление элемента из списка, объединение списков, поиск элемента в списке др. Наиболее просто определяется добавление элемента в список [1]:

```
добавить (X, L, [X|L]).
```

Здесь **X** — добавляемый элемент; **L** — список, в который добавляется элемент; **[X|L]** — результирующий список. Таким образом, в ходе доказательства этого целевого утверждения, пролог-система добавит **X** в начало списка **L** и сформирует результирующий список **[X|L]** . Например, в результате доказательства цели **добавить (a, [b,c], Y)** будет получено **Y=[a,b,c].**

Обработка списков часто выполняется с помощью рекурсивных предикатов. Например, определим отношение принадлежности — **принадлежит (X, L),** где **X** — элемент, а **L** — список. Элемент **X** принадлежит списку **L** , если: 1) **X** есть голова списка **L**; 2) или **X** принадлежит хвосту списка **L.** Это можно записать в виде двух утверждений, первое из которых есть факт, а второе — правило:

```
принадлежит(X, [X| Хвост ]).
принадлежит(X, [Голова|Хвост]): – принадлежит(X,[Хвост]).
```

В общем случае все такие определения обработки списков строятся по следующей схеме [1]:

```
предикат(…[ ]…).
предикат(…[Голова|Хвост]…): – обработка(Голова), предикат(…[Хвост]…).
```

Рекурсивные вызовы прекратятся, когда хвост списка окажется пустым списком. В приведенном определении первое утверждение определяет условие выхода из рекурсии, а второе утверждение — правило, предусматривающее при каждом вызове обработку очередного элемента списка и рекурсивный вызов определяемого предиката, аргументом которого является хвост списка.

1.2.7. Получение сведений из базы данных

Пролог является естественным языком запросов к реляционной базе данных. Пусть база данных представляется множеством фактов:

```
сотрудник(N, Фам, Имя, Отч, Отд, Должн, Филиал, Тел).
```

В запросах можно ссылаться на объекты данных, не указывая всех деталей. Например, проверка наличия в базе данных сведений о сотруднике **Петренко** может быть реализована запросом:

```
?- сотрудник( _, петренко, _, _, _, _, _, _).
true.
```

Символы подчеркивания обозначают различные анонимные переменные, в которые могут выполняться различные подстановки. Чтобы извлечь из базы данных фамилии и телефоны сотрудников, следует в целевом утверждении (вопросе) на соответствующих позициях записать переменные:

```
?- сотрудник( _, X, _, _, _, _, Y).
X = петренко,
Y = 68-23-42.
```

В этом случае пролог-система выполняет просмотр базы данных, находит терм **сотрудник** и сопоставляет **X** с фамилией, а **Y** — с номером телефона сотрудника.

Приведенные примеры запросов соответствуют операции «проекция» реляционной алгебры. В общем, *проекция* состоит в построении отношения, использующего лишь некоторые аргументы исходного n-арного отношения **r(X1,…,Xn).** Например, следующая проекция, записываемая в виде правила Пролога, оставляет первый и третий аргументы исходного отношения **r**

```
r_1_3(X1, X3):- r(X1,…,Xn).
```

Проекции широко используют при построении отношений, с помощью которых можно будет выбирать конкретные компоненты структур. Такие отношения можно назвать *селекторами*. Отношение-селектор будет иметь два аргумента: первый аргумент — объект данных, содержащий компоненту; второй — переменная, представляющая извлекаемую компоненту. Имя отношения-селектора будет совпадать с именем компоненты, которую нужно выбирать [2]:

```
отношение-селектор(Объект, Компонента).
```

Примеры некоторых отношений-селекторов для структуры **сотрудник:**

```
фамилия(сотрудник(_,Фамилия,_,_,_,_,_), Фамилия).
должность(сотрудник(_,_,_,_,_,Должность,_,_), Должность).
телефон(сотрудник(_,_,_,_,_,_,_,Телефон), Телефон).
```

Также просто описывается любой конкретный случай *выборки*. Например, рассмотрим отношение, формирующие наборы данных, в которых второй аргумент отношения **сотрудник** — это Петренко или Павлов:

```
один_из_двух_сотрудников(N, Фам, Имя, Отч, Отд, Должн, Филиал, Тел):-
        сотрудник(N, Фам, Имя, Отч, Отд, Должн, Филиал, Тел),
        (Фам=петренко; Фам=павлов).
```

В этом примере пролог-система для очередной фамилии **Фам**, конкретизируемой при сопоставлении отношения **сотрудник** с фактами базы данных, проверяет, равна ли она Петренко или Павлов.

В том случае, когда формируемые выборки необходимо собрать в список для дальнейшей обработки удобно использовать встроенный предикат **bagof** [1,2]. Цель **bagof(X, C, L)** формирует список **L** всех объектов **X**, удовлетворяющих усло-

вию **C**. Например, требуется сформировать список **L**, представляющий подмножество сотрудников одного филиала:

```
подмножество_сотрудников(Филиал,L):-
  bagof(сотрудник_ф(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел),
        сотрудник(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел),  L).
```

Предикат сформирует список **L** из новых отношений **сотрудник_ф** . В дальнейшем при необходимости эти отношения можно добавить в базу данных.

Кроме проекции и выборки, выделяют еще 4 основные операций реляционной алгебры: объединение, пересечение, разность и декартово произведение.

Операция *объединения* двух n-арных отношений **r1(X1,…,Xn)** и **r2(X1,…,Xn)** строит новое n-арное отношение, содержащие множество кортежей, принадлежащих либо первому отношению, либо второму. Новое отношение задается на Прологе следующим правилом:

```
объединение_r1_r2(X1,…,Xn):- r1(X1,…,Xn); r2(X1,…,Xn).
```

Например, если имеются два отношения **сотрудник**, которые содержат сведения о сотрудниках двух филиалов, то объединенное отношение будет содержать общий перечень сотрудников.

Пересечением называется отношение, которое содержит множество кортежей, принадлежащих одновременно и первому и второму отношениям:

```
пересечение_r1_r2(X1,…,X2):- r1(X1,…,Xn), r2(X1,…,Xn).
```

Например, применительно к базе данных сотрудников двух филиалов с помощью пересечения можно получить список сотрудников, занимающих одну и ту же должность.

Разностью двух отношений **r1** и **r2** называется отношение, содержащее множество кортежей, принадлежащих **r1** и не принадлежащих **r2** :

```
разность_r1_r2(X1,…,X2):- r1(X1,…,Xn), not r2(X1,…,Xn).
```

Декартово произведение может быть определено следующим правилом. Если **r** *m*-арное отношение, а **s** *n*-арное, то (*m*+*n*)-арное отношение, представляющее декартово произведение, определится так:

```
декартово_произведение_ r _ s (X1,…,Xm+n):- r (X1,…,Xm), s(Xm+1,…,Xm+n).
```

Операция декартова произведения обычно используется для генерации некоторого отношения, которое характеризует все возможные комбинации между кортежами отношений **r** и **s**. В дальнейшем из этого отношения можно получать различные проекции.

1.2.8. Предикаты для работы с динамической базой данных

В Прологе имеется ряд встроенных предикатов, которые позволяют изменять базу данных пролог-системы в процессе выполнения программ: **assert(X), asserta(X), assertz(X), retract(X)** [1, 2, 10].

Предикат **assert(X)** добавляет в базу данных утверждение **X**. Например:

```
? – assert(столица('Египет', 'Каир')).
     Yes
? – столица('Египет', X).
     X = Каир
```

Предикат **asserta(X)** добавляет утверждение **X** в начало базы данных, предикат **assertz(X)** — в конец базы данных. Предикат **retract(X)** выполняет поиск утверждения **X** в базе данных и удаляет первое найденное утверждение. Встроенный предикат **retractall(X)** удаляет из базы данных все утверждения, функтор и арность которых сопоставимы с **X.**

Для просмотра утверждений, входящих в базу данных, можно воспользоваться предикатами: **listing** — выводит утверждения, содержащиеся в базе данных, в выходной поток; **listing(X)** — печатаются утверждения, сопоставимые с **X**.

1.2.9. Предикаты ввода-вывода

Пролог взаимодействует с *входными* и *выходными потоками.* В каждый момент времени Пролог взаимодействует с одним входным и одним выходным потоками. Перечень встроенных предикатов, предназначенных для управления входными и выходными потоками, приведен в таблице 1.1. Для считывания значений из входных потоков и записи их в выходные потоки применяют следующие предикаты:

read(X) — вызывает чтение очередного терма из входного потока и сопоставление его с **X**, вводимые термы **должны заканчиваться точкой**, в конце файла **X** конкретизируется атомом **end_of_file**;

write (X) — выводит терм X в текущий выходной поток;

nl — переход на новую строку;

tab(N) — выводит в выходной поток N-пробелов;

put(X) — выводит символ с ASCII кодом X на терминал;

get0(X) — считывает один символ с клавиатуры и сопоставляет его **X**;

get(X) — сопоставляет **X** с первым символом, отличным от пробела;

display(X) — выводит терм **X** в стандартной скобочной записи в текущий выходной поток.

Таблица 1.1— Предикаты управления потоками

Входной поток	Выходной поток	Интерпретация предиката
see(<имя файла>)	**tell(<имя файла>)**	Определяет в качестве текущего входного или выходного потока соответствующий файл.
seeing(<имя файла>)	**telling(<имя файла>)**	Отождествляет имя файла с текущим входным или выходным потоком.
seen	**told**	Закрывает текущий входной или выходной поток и опять связывает с текущим входным или выходным потоком поль-

		зовательский терминал.

Файлы Пролога являются текстовыми и обрабатываются последовательно. Типовая последовательность вызова встроенных предикатов при выполнении ввода из файла является следующей:

```
see('имя файла'),          % открытие файла
read(X),                   % чтение терма X
…,                         % обработка
seen.                      % закрытие файла
```

Аналогично строится работа с выходным потоком. Примеры применения предикатов ввода-вывода приведены в приложении Б методических указаний [3].

1.3. Варианты заданий

Написать программу, обеспечивающую создание динамической базы данных. Структура базы данных определяется одной из таблиц в соответствии с вариантом задания. В функции программы должно входить:

− добавление записи в базу данных;
− удаление записи из базы данных;
− просмотр базы данных;
− сохранение базы данных в файле;
− загрузка базы данных из файла;
− реализация операций реляционной алгебры (на примерах).

Кроме этого, программа должна выполнять дополнительные функции, указанные в варианте задания (таблица 1.2).

Таблица 1.2 — Варианты заданий

Вариант	Номер таблицы и дополнительные функции
1.	Таблица 1.3. Корректировка данных в базе по номеру записи; вывод на дисплей фамилий и номеров групп для всех студентов, если средний балл студента больше 4.0; если таких студентов нет, вывести соответствующее сообщение.
2.	Таблица 1.3. Корректировка данных в базе по фамилии; вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5; если таких студентов нет, вывести соответствующее сообщение.
3.	Таблица 1.3. Корректировка данных в базе по номеру группы; вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2; если таких студентов нет, вывести соответствующее сообщение.
4.	Таблица 1.4. Корректировка данных в базе по номеру рейса; вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры; если таких рейсов нет, выдать на дисплей соответствующее сообщение.
5.	Таблица 1.4. Корректировка данных в базе по типу самолета; вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры; если таких рейсов нет, выдать на дисплей соответствующее сообщение.
6.	Таблица 1.5. Корректировка данных в базе по фамилии; вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры; если таких работников нет, вывести на дисплей соответствующее сообщение.

Продолжение таблицы 1.2

Вариант	Номер таблицы и дополнительные функции
7.	Таблица 1.6. Корректировка данных в базе по номеру поезда; вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени; если таких поездов нет, выдать на дисплей соответствующее сообщение.
8.	Таблица 1.6. Корректировка данных в базе по пункту назначения; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение.
9.	Таблица 1.6. Корректировка данных в базе по времени отправления; вывод на экран информации о поезде, номер которого введен с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение.
10.	Таблица 1.7. Корректировка данных в базе по начальному маршруту; вывод на экран информации о маршруте, номер которого введен с клавиатуры; если таких маршрутов нет, выдать на дисплей соответствующее сообщение.
11.	Таблица 1.7. Корректировка данных в базе по номеру маршрута; вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры; если таких маршрутов нет, выдать на дисплей соответствующее сообщение.
12.	Таблица 3.8. Корректировка данных в базе по фамилии; вывод на экран информации о человеке, номер телефона которого введен с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение.
13.	Таблица 1.8. Корректировка данных в базе по номеру телефона; вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры; если таких нет, выдать на дисплей соответствующее сообщение.
14.	Таблица 1.8. Корректировка данных в базе по году рождения; вывод на экран информации о человеке, чья фамилия введена с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение.
15.	Таблица 1.9. Корректировка данных в базе по фамилии; вывод на экран информации о человеке, чья фамилия введена с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение.
16.	Таблица 1.9. Корректировка данных в базе по знаку зодиака ; вывод на экран информации о людях, родившихся под знаком, название которого введено с клавиатуры; если таких нет, выдать на дисплей соответствующее сообщение.
17.	Таблица 3.9. Корректировка данных в базе по месяцу рождения ; вывод на экран информации о людях, родившихся в месяц, значение которого введено с клавиатуры; если таких нет, выдать на дисплей соответствующее сообщение.
18.	Таблица 1.10. Корректировка данных в базе по названию товара; вывод на экран информации о товаре, название которого введено с клавиатуры; если таких товаров нет, выдать на дисплей соответствующее сообщение.
19.	Таблица 1.10. Корректировка данных в базе по названию магазина; вывод на экран информации о товарах, продающихся в магазине, название которого введено с клавиатуры; если такого магазина нет, выдать на дисплей соответствующее сообщение.
20.	Таблица 1.11. Корректировка данных в базе по расчетному счету плательщика ; вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры; если такого расчетного счета нет, выдать на дисплей соответствующее сообщение.
21.	Таблица 1.12 Корректировка данных в базе по фамилии; вывод анкетных данных студентов отличников; если таких студентов нет, вывести соответствующее сообщение.

Продолжение таблицы 1.2

Вариант	Номер таблицы и дополнительные функции
22.	Таблица 1.12. Корректировка данных в базе по году рождения; вывод на дисплей анкетных данных студентов, получивших одну оценку 3; если таких студентов нет, вывести соответствующее сообщение.
23.	Таблица 1.12. Корректировка данных в базе по году поступления; вывод на дисплей анкетных данных студентов, получивших все двойки; если таких студентов нет, вывести соответствующее сообщение.
24.	Таблица 1.12. Корректировка данных в базе по оценке «физика»; вывод на дисплей анкетных данных студентов, получивших все пятерки; если таких студентов нет, вывести соответствующее сообщение.
25.	Таблица 1.12. Корректировка данных в базе по номеру ; вывод на дисплей анкетных данных студентов, получивших одну оценку 4, а все остальные – 5; если таких студентов нет, вывести соответствующее сообщение.
26.	Таблица 1.12. Корректировка данных в базе по фамилии, которая начинается с литеры ‘А’ ; вывод на дисплей фамилий студентов, которые начинаются с литеры ‘А’, и их оценки; если таких студентов нет, вывести соответствующее сообщение.
27.	Таблица 1.12. Корректировка данных в базе по фамилии, которая начинается с литеры ‘Б’; вывод на дисплей фамилий студентов, которые начинаются с литеры ‘Б’, и год их рождения; если таких студентов нет, вывести соответствующее сообщение.
28.	Таблица 1.12. Корректировка данных в базе по фамилии, которая начинается с литеры ‘А’или ‘Д’ ; вывод на дисплей фамилий студентов, которые начинаются с литеры ‘А’или ‘Д’, и год их поступления; если таких студентов нет, вывести соответствующее сообщение.

Таблица 1.3. — Студент группы

Фамилия И.О.	Номер группы	Успеваемость				
		Р1	Р2	Р3	Р4	Р5

Таблица 1.4. — Рейс самолета

Пункт назначения	Номер рейса	Тип самолета

Таблица 1.5. — Сотрудник

Фамилия И.О.	Должность	Год приема на работу

Таблица 1.6. — Поезд

Пункт назначения	Номер поезда	Время отправления

Таблица 1.7. — Маршрут

Начальный пункт	Конечный пункт	Номер маршрута

Таблица 1.8. — Записная книжка

Фамилия Имя	Номер телефона	Дата рождения		
		день	месяц	год

Таблица 1.9. — Знак зодиака

Фамилия Имя	Знак зодиака	Дата рождения		
		день	месяц	год

Таблица 1.10. — Стоимость

Название товара	Название магазина	Стоимость товара, грн

Таблица 1.11. — Счет

Расчетный счет плательщика	Расчетный счет получателя	Перечисляемая сумма, грн.

Таблица 1.12.— Студент

Номер	Фамилия Имя	Год рождения	Год поступления	Оценки		
				Ф	ВМ	Пр.

1.4. Порядок выполнения лабораторной работы

1.4.1. Изучить среду программирования по инструкции в МУДЛ или с использованием материалов из [3]. Выполнить все приведенные примеры в окне консоли среды программирования.

1.4.2. Ознакомиться по лекционному материалу или книгам [1, 2] с объектами данных и сопоставлением в языке Пролог, организацией управления в пролог-программах, предикатами обработки списков, встроенными предикатами работы с базой данных и предикатами ввода-вывода. Изучить примеры применения этих предикатов, приведенные в разделе 1.2 настоящей лабораторной работы.

1.4.2. Ознакомиться с вариантом задания и выбрать один из способов для хранения записей базы данных (см. п. 1.2.5).

1.4.3. Ознакомиться с примером кода, приведенного в приложении А, и по аналогии определить на языке Пролог для заданного варианта предикаты добавления записи в базу, удаления записи, просмотра базы, сохранения базы в файле и загрузки её из файла.

1.4.4. Создать в среде программирования Пролог-проект в соответствии с инструкцией в МУДЛ по использованию SWI-Prolog или методическими указаниями [3], содержащий подготовленные определения предикатов, указанных в п. 1.4.3.

1.4.5. Выполнить частичную отладку проекта.

1.4.6. Разработать определения дополнительных предикатов выборки и корректировки записей базы данных в соответствии с вариантом. При этом выборку записей из базы выполнять путем реализации операции проекции реляционной алгебры, следуя общим рекомендациям, указанным в п. 1.2.7.

1.4.7. Разработать предикаты, реализующие примеры операций реляционной алгебры (объединение, пересечение, разность) в соответствии с п. 1.2.7.

1.4.8. Выполнить полную отладку проекта и зафиксировать результаты работы программы в виде экранных копий.

1.5. Содержание отчета

Цель работы, вариант задания, обоснование выбранного представления базы данных, описание определений предикатов для общей работы с базой данных, описание разработанных дополнительных предикатов в соответствии с вариантом

задания, описание примеров реализации операций реляционной алгебры, тестовых запросов и результатов их выполнения, выводы.

1.6. Контрольные вопросы

1.6.1. Назовите составные части пролог-программы и приведите их определения.

1.6.2. Сформулируйте определения следующих понятий языка Пролог: терм, константа, атом, переменная, анонимная переменная, структура, список.

1.6.3. Что понимают под сопоставлением (унификацией) в языке Пролог?

1.6.4. Сформулируйте правила сопоставления термов в Прологе.

1.6.5. В чем заключаются декларативный и процедурный смыслы пролог-программы?

1.6.6. Объясните процедурный смысл простейших правил с конъюнкцией и дизъюнкцией условий.

1.6.7. Объясните назначение предиката отсечения. Приведите пример.

1.6.8. С какой целью применяется встроенный предикат **fail** ?

1.6.9. Приведите определение цикла **repeat-fail** .Объясните порядок его выполнения.

1.6.10. Объясните понятие группа в языке Пролог.

1.6.11. Назовите и объясните простейшие способы представления базы данных с помощью языка Пролог.

1.6.12. Определите следующие предикаты для работы со списками **L: добавить(X,L), принадлежит(X,L), объединить(L1,L2,L3).**

1.6.13. Определите операцию “проекция” в виде правила Пролога. Покажите, как с помощью этой операции реализуются отношения-селекторы.

1.6.14. Объясните встроенный предикат **bagof(X, C, L).**

1.6.15. Определите следующие операции реляционной алгебры в виде правил Пролога: объединение, пересечение, разность, декартово произведение.

1.6.16. Перечислите и объясните встроенные предикаты добавления утверждений в базу данных Пролога.

1.6.17. Назовите и объясните встроенные предикаты для удаления утверждений из базы данных Пролога.

1.6.18. Как просмотреть утверждения базы данных Пролога?

1.6.19. Что понимают под потоками ввода-вывода? Как открыть поток?

1.6.20. Какие предикаты используют для записи термов в поток?

1.6.21. Какие предикаты используют для чтения термов из потока?

1.6.22. Какие предикаты применяются для ввода-вывода символов?

1.6.23. Какое значение получает считываемый терм при достижении метки конец файла?

2. ЛАБОРАТОРНАЯ РАБОТА № 2
«ПОИСК РЕШЕНИЙ CSP ЗАДАЧ»

2.1. Цель работы

Изучение особенностей задач удовлетворения ограничений (**CSP** — **C**onstraint **S**atisfaction **P**roblem) и исследование основных методов поиска их решений средствами языка Пролог.

2.2. Краткие теоретические сведения

2.2.1. Общая формулировка задачи CSP

В системах искусственного интеллекта используют несколько способов представления задач [1]. Ранее рассматривалось представление задач в пространстве состояний, в соответствии с которым поиск решения задачи сводился к нахождению пути на графе состояний. В том случае, когда целью поиска является непосредственно конечное состояние задачи и сам путь не представляет интереса, может использоваться представление в виде CSP задачи. Любая задача CSP определяется совокупностью трех составляющих [1,2,8]:

1) множеством переменных $X_1, X_2, \ldots, X_n$;
2) областью определения каждой переменной $D_1, D_2, \ldots, D_n$;
3) множеством ограничений (отношений) $C_1, C_2, \ldots, C_m$, каждое из которых включает некоторое подмножество переменных и задает допустимые комбинации значений для этого подмножества.

Состояние задачи определяется путем присваивания значений некоторым или всем переменным. Присваивание, которое не нарушает никаких ограничений, называется *совместимым*. *Полным* называется присваивание, в котором участвует каждая переменная, а *решением* задачи CSP является полное присваивание, которое удовлетворяет всем ограничениям.

Уточним сказанное на примере задачи раскрашивания плоской карты (рис.2.1). Необходимо раскрасить карту, используя только 3 цвета. Соседние регионы на карте не должны раскрашиваться одним и тем же цветом.

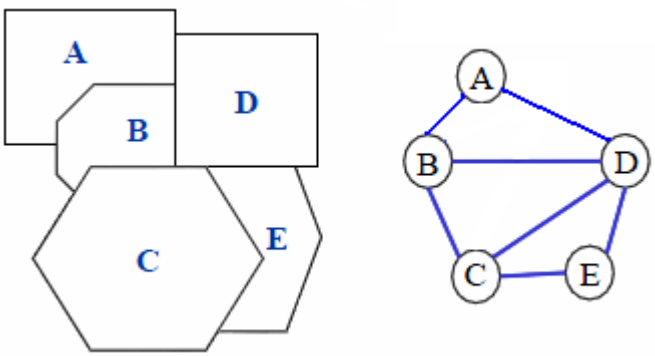

Рисунок 2.1 — Задача раскрашивания карты и граф ограничений

Чтобы определить эту задачу в виде задачи CSP в качестве переменных будем использовать буквенные обозначения регионов на карте: *A, B, C, D, E.* Областью определения каждой переменной является множество цветов D_i={*red, green, blue*}. Ограничения требуют, чтобы соседние регионы имели разные цвета, например, допустимые комбинации цветов для регионов *A* и *B*: { *(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)*}. В общем случае ограничения этой задачи можно записать в виде*:* $Color(X_i) \neq Color(X_j)$, где X_i и X_j переменные, обозначающие соседние регионы. Решение задачи заключается в том, чтобы всем переменным (*A, B, C, D, E*) присвоить совместимые значения.

Задачу CSP удобно представлять в виде *графа ограничений*, узлы которого представляют переменные задачи, а дуги — ограничения (см. рис. 2.1).

Переменные задачи могут быть дискретными или непрерывными, с конечными или бесконечными областями определений [8]. Ниже рассматриваются только простые CSP задачи *с дискретными переменными*, характеризуемыми *конечной областью определения*. Если максимальный размер области определения равен *d,* то в худшем случае количество возможных полных присваиваний оценивается величиной $O(d^n)$. Поэтому время решения задачи экспоненциально зависит от количества переменных. Однако в большинстве практических случаев алгоритмы CSP общего назначения позволяют решать задачи на несколько порядков более крупные, например, по сравнению алгоритмами поиска решений в пространстве состояний [8].

Ограничения (отношения) могут быть унарными, бинарными и ограничениями высокого порядка в зависимости от количества переменных. *Унарные* ограничения ограничивают значение одной переменной. Унарное ограничение можно устранить, удалив соответствующее значение из области определения переменной, нарушающее это ограничение. *Бинарное ограничение* связывает между собой две переменные. CSP задача, в которой используются только бинарные ограничения, может быть представлена графом ограничений. В *ограничениях высокого порядка* участвуют три и больше переменных. Ограничение высокого порядка может сведено к бинарным ограничениям путем введения вспомогательных переменных [8].

2.2.2. Общие методы решения CSP задач

Рассмотрим 4 метода решения CSP задач: «генерируй и тестируй», поиск с возвратами, поиск с предварительной проверкой, поиск с распространением ограничения [8].

2.2.2.1. Метод «генерируй и тестируй» (generate and test). Метод также называют методом «образуй и проверь» [2,11]. В соответствии с этим методом генерируются все возможные полные присваивания переменным, и каждое из них тестируется (проверяется) на совместимость. Структура соответствующей программы весьма проста и выглядит в виде вложенных циклов по каждой переменной. В самом внутреннем цикле выполняется проверка каждого ограничения. Если они все выполняются, то текущее полное присвоение — решение задачи.

В большинстве случае «слепая» генерация всех полных присваиваний весьма неэффективна, т.к. приводит к разрастанию дерева поиска.

2.2.2.2. Поиск с возвратами (backtracking search). По сути, это тот же метод «генерируй и тестируй», но организованный в виде поиска в глубину, в котором присваивается значение очередной переменной, проверяются ограничения и выполняется возврат, если присвоение не допустимо. ***Таким образом, проверка ограничений как бы погружается в процесс генерации решения, что позволяет ограничить разрастание дерева поиска.***

Эффективность метода зависит от порядка выбора переменных. Наиболее часто для определения порядка выбора переменных используют две эвристики: степенную эвристику и MRV-эвристику (**MRV — M**inimum **R**emaining **V**alues). *Степенная* эвристика позволяет уменьшить степень ветвления за счет выбора переменной, которая участвует в наибольшем количестве ограничений. *MRV-эвристика* предусматривает выбор переменной с наименьшим количеством оставшихся допустимых значений. Такая переменная с наибольшей вероятностью, вскоре приведет к неудаче, усекая тем самым дерево поиска. Степенная эвристика обычно используется для начального выбора переменных, а MRV-эвристика в ходе дальнейших присваиваний.

В процессе поиска с возвратами множество переменных, которым уже присвоены значения, на каждом шаге расширяется на очередную переменную, если выполняются соответствующие ограничения. Процесс завершается, если получено полное присваивание.

2.2.2.3. Метод предварительной (опережающей) проверки (forward checking). В случае поиска с возвратами ограничения, в которых участвует некоторая переменная, учитываются непосредственно в момент, когда происходит назначение значения этой переменной. Но выполняя опережающую проверку некоторых ограничений на предшествующих этапах поиска, можно резко сократить пространство поиска.

В соответствии с методом предварительной проверки в момент присваивания значения переменной *X* просматривается каждая переменная *Y*, которой не присвоены значения и которая связана с *X* некоторым ограничением. При этом из области определения Y удаляется любое значение, которое несовместимо со значением, присвоенным переменной *X*.

Вернемся к задаче раскрашивания карты. В соответствии с графом ограничений (см. рис. 2.1) после присваивания *A*=red и *C*=green области определения переменных *B* и *D* сокращаются до единственного значения (рис.2.2).

Переменные	***A***	***B***	***C***	***D***	***E***
Начальная обл. определения	R G B	R G B	R G B	R G B	R G B
После присваивания ***A***=red	**red**	G B	R G B	G B	R G B
После присваивания ***C***=green	**red**	B	**green**	B	R B
После присваивания ***E***=blue	**red**	B	**green**	-	**blue**

Рисунок 2.2 **—** Поиск с предварительной проверкой

Таким образом, ветвление, связанное с поиском значений для этих переменных, полностью устраняется (за счет распространения вперед информации о присвоенных значениях для переменных *A* и *C)*. После присваивания *E*=blue область определения *D* становится пустой, т.е. предварительная проверка обнаруживает, что частичное присваивание {*A*=red , *C*=green, *E*=blue} несовместимо с ограничениями задачи, и алгоритм выполнит возврат, не предпринимая попыток присваивания значений оставшимся переменным.

2.2.2.4. Метод распространения ограничения. Предварительная проверка не позволяет обнаруживать все несовместимости. Распространение ограничения — это общее название методов обнаружения потенциальных несовместимостей на ранних этапах решения задачи за счет распространения последствий применения некоторого ограничения к одной из переменных.

Для быстрого распространения ограничений выполняется проверка *совместимости дуг.* Дуга (*X, Y*) называется *совместимой*, если для каждого значения *x* из области определения переменной *X* существует некоторое значение *y* из области определения переменной *Y*, которое удовлетворяет бинарному ограничению между этими переменными. Обратим внимание на то, что понятие совместимости дуг является направленным, т.е. если дуга (*X, Y*) совместима, то это не означает, что обратная дуга (*Y, X*) также совместима.

Например, в третьей строке (рис. 2.2) текущими областями определения переменных *D* и *E* являются {blue} и {red, blue}. При *D*= blue существует совместимое присваивание для *E*, а именно *E*=red. Поэтому дуга от *D* до *E* совместима. С другой стороны, обратная дуга от *E* до *D* несовместима, так как для *E*=blue не существует совместимого присваивания. Эту дугу можно сделать совместимой, удалив значение blue из области определения *E*.

Проверку совместимости дуг можно использовать либо в качестве этапа предварительной обработки перед началом процесса поиска, либо в качестве этапа распространения ограничения после каждого присваивания во время поиска Проверка совместимости дуг требует дополнительных затрат времени, которые в наихудшем случае составляют $O(n^2 d^3)$. По этой причине в большинстве случаев при решении простых CSP задач ограничиваются применением поиска с возвратами или поиска с предварительной проверкой.

2.2.2.5. Обработка специальных ограничений. Встречаются некоторые типы ограничений, которые могут обрабатываться более эффективно с помощью специальных алгоритмов, а не общих алгоритмов, рассмотренных выше. Например, ограничение *Alldiff*, которое указывает, что все переменные задачи должны иметь разные значения. Один из простых способов проверки несовместимости для ограничения *Alldiff* сводится к правилу: если в ограничении участвуют *m* переменных и все они вместе взятые имеют *n* возможных значений, то при $m>n$ ограничение не может быть удовлетворено.

2.2.3. Решение CSP задач на языке Пролог

Так как в Пролог встроен механизм поиска с возвратами, который лежит в основе рассмотренных выше общих методов решения CSP задач, то на нём достаточно просто могут быть реализованы эти методы. Рассмотрим примеры написания соответствующих программ.

2.2.3.1. Прямая реализация метода «генерируй и тестируй». На Прологе метод реализуется простой конъюнкцией двух целей, одна из которых выполняет генерацию возможных решений (полных присваиваний), а вторая проверяет, удовлетворяют ли эти решения ограничениям задачи:

```
решить(X) :- генерировать_решение(X), проверить_ограничения(X).
```

Если проверка завершается неудачей, то происходит возврат к цели **генерировать_решение,** которая генерирует новое решение. Процесс продолжается до тех пор, пока не будет найдено решение, удовлетворяющее ограничениям, или генератор не исчерпает все альтернативные варианты предполагаемых решений.

В качестве основы генератора решений, часто используют запросы к базам данных или предикаты работы со списками: **принадлежит, удалить, перестановка**. Предикат **принадлежит** (встроенный предикат **member)** можно использовать не только для проверки того, что данный элемент входит в список, но и для того, чтобы получать (генерировать) элементы списка:

```
:- принадлежит( X, [a,b,c] ).
X = a ;
X = b ;
X = c
```

Предикат **удалить(X, L, L1)** проверяет, входит ли элемент **X** в список **L**, удаляет его и возвращает список **L1** без элемента **X**. Определение предиката:

```
удалить( X, [X|T], T ).
удалить( X, [H|T], [H|T1] ):-удалить(X, T, T1).
```

Предикат удаляет только одно вхождение **X**. Предикат можно использовать для извлечения элементов из списка (поэтому соответствующий встроенный предикат именуется как **select**):

```
:- удалить(X,[1,2,3],L1).
X = 1
L1 = [2, 3] ;
X = 2
L1 = [1, 3] ;
X = 3
L1 = [1, 2]
```

Его можно также использовать для вставки элемента в список. Например:

```
:- удалить(a, L, [1,2,3]).
L = [a, 1, 2, 3] ;
L = [1, a, 2, 3] ;
L = [1, 2, a, 3] ;
L = [1, 2, 3, a]
```

Для вставки элемента в список разумно определить отдельный предикат: **вставить(X,L1,L):-удалить(X,L,L1).**

Для генерации вариантов решений CSP задач, также полезным может оказаться предикат **перестановка(L, P)** с двумя аргументами-списками, один из которых является перестановкой элементов другого. Предикат определяется следующим образом:

```
перестановка( [ ],[ ] ).
перестановка( [X|L], P ):-перестановка(L,L1), вставить(X, L1, P).
```

Если первый список пустой, то и второй список должен быть пустым. Если первый список не пуст, то находим перестановку **L1** для хвоста списка **L**, а затем выполняем вставку головы списка **X** в произвольную позицию **L1**. Пример вызова предиката:

```
:-перестановка([red, blue, green], P).
P = [red, blue, green];
P = [blue, red, green];
P = [blue, green, red];
P = [red, green, blue];
P = [green, red, blue];
P = [green, blue, red]
```

Как и предполагалось, построены все шесть возможных перестановок.

Применим метод «генерируй и тестируй» к решению задачи раскрашивания карты, изображенной на рис. 2.1. При этом решение **X** будем представлять списком цветов вершин графа ограничений в порядке алфавита, т.е **[A,B,C,D,E].** Списки цветов будем генерировать с помощью предиката **перестановка,** а затем проверять, чтобы смежные вершины имели разные цвета. Программа получается довольно простой:

```
решить(X) :- генерировать_решение(X),  проверить_ограничения(X).

генерировать_решение(Список_цветов) :-
   перестановка([red, green, blue, red, green], Список_цветов).

проверить_ограничения(Список_цветов) :-
   Список_цветов = [A,B,C,D,E],
   A\==B,A\==D,          %цвет A отличается от цвета B и цвета D
   B\==D, B\==C,         %цвет B отличается от цвета D и цвета C
   C\==D, C\==E,         %цвет C отличается от цвета D и цвета E
   D\==E.                %цвет A отличается от цвета E
```

При запросе **решить(X)** получим следующий ответ: **X = [red, green, red, blue, green].**

Чтобы показать, что такой подход к построению программ является достаточно общим, рассмотрим логическую задачу. Известны следующие сведения о людях с именами: Иван, Нина, Сергей, Аня. Иван старше Нины. Сергей самый молодой. Аня не самая старшая. При заданных ограничениях, получить список

имен, упорядоченный по возрасту, т.е. **[И1, И2, И3, И4],** где **И1** самый молодой, т.е. Сергей. Построим программу по образцу предыдущей:

```
решить1(X) :-  генерировать_решение(X),  проверить_ограничения(X).

%формирование очередного решения  путем  перестановки имен в списке
генерировать_решение(Список_имен) :-
   перестановка([иван, нина, сергей, аня], Список_имен).

%проверка выполнения ограничений
проверить_ограничения(Список_имен) :-
   Список_имен = [И1, И2, И3, И4],
   предшествует(нина, иван, Список_имен),        %Иван старше Нины
   И1 = сергей,                                   %Сергей самый молодой
   принадлежит(аня, [И1,И2,И3]).                  %Аня не самая старшая

%отношение предшествует(X,Y,L) верно, если  X следует в списке L  раньше  Y
предшествует(X,Y,[X|T]) :-принадлежит(Y,T).
предшествует(X,Y,[_|T]) :- предшествует(X,Y,T).
```

Программа находит два решения: **X = [сергей, нина, аня, иван]** и **X = [сергей, аня, нина, иван].**

В большинстве случаев программы, реализующие прямой метод «генерируй и тестируй», неэффективны. Чтобы повысить эффективность процесса поиска решения, следует проверку ограничений выполнять по мере присваивания значений переменным. Ранняя проверка ограничений возможна при использовании поиска с возвратами, когда процесс проверки ограничений погружается в процесс генерации решений.

2.2.3.2. Реализация поиска с возвратами и предварительной проверкой ограничений. Вернемся к задаче раскрашивания карты. Представим карту (см. рис. 2.1.) в виде списка дуг **[A::B, A::D, B::D, B::C, D::C, D::E, C::E],** где символ **::** — инфиксный оператор (объявляемый в программе), который будет обозначать ребро графа. Обратим внимание, что ребра связывают переменные задачи, которым необходимо присвоить согласованные значения. Будем поочередно обрабатывать элементы этого списка, назначая цвета (из списка цветов) очередной паре переменных и проверять, чтобы эти цвета не совпадали. При этом, если какие-либо переменные получат значения, например для первого элемента **A=red** и **B=green**, то эти же значения автоматически назначаются и в других парах, т.е. распространяются вдоль всего списка, ограничивая (сокращая) области определения связанных переменных, в примере **D и C**. Следовательно, автоматически будет выполняться предварительная проверка ограничений.

Для назначения цвета переменной будем использовать предикат **принадлежит(X,L)**, который также будет порождать точки возврата. В случае возврата сделанные назначения автоматически стираются. Определим предикат **раскрасить1(Карта, Список_цветов)** рекурсивно:

```
%объявление инфиксного оператора
:- op(100,xfy,'::').

раскрасить1([X1::X2|Ост], Список_цветов):-
```

```
        принадлежит(X1, Список_цветов),          %назначение цвета X1
        принадлежит(X2, Список_цветов),          %назначение цвета X2
        X1\=X2,                                  %проверка ограничения
        раскрасить1(Ост, Список_цветов).         %раскрасить оставшуюся часть
раскрасить1([ ], _).
```

К недостаткам этого определения следует отнести, то что предикат **принадлежит** при первом вызове назначает переменным **X1** и **X2** одинаковые цвета, что создает очевидный возврат после проверки ограничения **X1\=X2.** Эта проблема легко преодолевается, если для назначения цвета **X1** использовать предикат **удалить(X1, Список_цветов, Оставшиеся_цвета)**, а цвет переменной **X2** выбирать из списка оставшихся цветов.

```
раскрасить2([X1::X2|Ост], Список_цветов):-
        удалить(X1,Список_цветов, Оставшиеся_цвета),  %назначение цвета X1
        принадлежит(X2, Оставшиеся_цвета),            %назначение цвета X2
        раскрасить2(Ост,Список_цветов).
раскрасить2([ ], _).
```

При запросе **раскрасить2([A::B, A::D, B::D, B::C, D::C, D::E, C::E], [red,green,blue])** получим следующий ответ:

```
A = C, C = red,
B = E, E = green,
D = blue.
```

Можно сравнить варианты решения задачи раскрашивания карты с использованием прямой реализации метода «генерируй и тестируй» и с использованием поиска с возвратами и предварительной проверкой. Для этого можно воспользоваться встроенным предикатом **time(Цель),** который возвращает статистику общего числа логических выводов при выполнении предиката **Цель** и процессорное время. Чтобы получить устойчивые результаты будем решение задачи выполнять 1000 раз:

```
:-time(повторять(Цель,1000)).

% цикл повторения выполнения Цели заданное число раз (N)
повторять(Цель,1):-Цель.
повторять(Цель,N):-
            not(not(Цель)),                  %стирание предыдущих подстановок
            M is N-1,повторять(Цель,M).
```

В результате вызова **time(повторять(решить(X),1000))** получим:

```
% 10,673,997 inferences, 2.137 CPU in 2.137 seconds (100% CPU, 4994352 Lips)
X = [red, green, red, blue, green]
```

Вызов **time(повторять(раскрасить2([A::B, A::D, B::D, B::C,D::C, D::E, C::E], [red, green, blue]),1000))** вернет то же решение. Однако поиск его происходит намного быстрее:

```
% 91,051 inferences, 0.047 CPU in 0.046 seconds (102% CPU, 1945522 Lips)
A = C, C = red,
```

B = E, E = green,
D = blue

В приложении Б.1 приведен еще один вариант решения задачи раскрашивания карты, который выполняется 2 раза быстрее **раскрасить2.**

Эффективный вариант реализации можно получить и для примера простейшей логической задачи рассмотренной выше. Оказывается, чтобы упорядочить список имен по возрасту методом поиска с возвратом и предварительной проверкой ограничений, ничего менять в основных определениях не надо. Достаточно просто сначала проверить выполнение ограничений, а затем дополнить полученный шаблон решения некоторыми не присвоенными значениями путем генерации перестановок. Иными словами, необходимо поменять местами цели в предикате **решить1(X).** Тогда получим следующее определение:

решить2(X):- проверить_ограничения(X), генерировать_решение(X).

Такое изменение порядка следования целей возможно, так как цель **проверить_ограничения(X)** может не только проверять ограничения, но и выполнять генерацию частичных решений. В таком случае мы добиваемся того, что проверка ограничений максимально погружается в процесс генерации потенциальных решений.

Вызов **решить2(X)** вернет решение аналогичное вызову **решить1(X).** Но если при выполнении предиката **решить1(X)** генерируется 4!=24 перестановки, то при выполнении предиката **решить2(X)** ограничение «Сергей самый молодой» сразу исключает 18 перестановок, а ограничение «Аня не самая старшая» — еще 2 перестановки. Сравнение времени поиска показывает, что вызов **решить2(X)** выполняется в 3 раза быстрее:

:-time(повторять(решить1(X),1000)).
% 104,997 inferences, 0.047 CPU in 0.043 seconds (109% CPU, 2243511 Lips)
X = [сергей, нина, аня, иван]

:-time(повторять(решить2(X),1000)).
% 41,997 inferences, 0.016 CPU in 0.016 seconds (98% CPU, 2692098 Lips)
X = [сергей, нина, аня, иван]

В приложении Б.2 приведен пример решения более сложной логической задачи с ограничениями типа *Alldiff*, для которой смена порядка следования целей «генерировать» и «тестировать» даёт еще больший выигрыш. В приложении Б.3. приведено решение головоломки Эйнштейна, для которой вариант с предварительной генерацией полных присваиваний совершенно не приемлем, т.к. потребует очень больших временных затрат.

2.3. Варианты заданий

Решить логическую задачу [7] с обязательным использованием методов поиска решений CSP задач на языке Пролог, рассмотренных в п.2.2.

1. Известно, что Единорог лжет по понедельникам, вторникам и средам и говорит правду во все остальные дни недели. Он может сказать: "Вчера я лгал. После завтрашнего дня я буду лгать два дня подряд". В какой день Единорог произнес эту фразу.

2. Три друга — Петр, Роман и Сергей — учатся на математическом, физическом и химическом факультетах. Если Петр — математик, то Сергей не физик. Если Роман не физик, то Петр — математик. Если Сергей не математик, то Роман — химик. Какая специальность у Сергея?

3. Четыре юных филателиста: Митя, Толя, Петя и Саша — купили почтовые марки. Каждый из них покупал марки только одной страны, причем двое из них купили российские марки, один — болгарские и один — чешские. Известно, что Митя и Толя купили марки двух разных стран. Марки разных стран купили Митя с Сашей, Петя с Сашей, Петя с Митей и Толя с Сашей. Кроме того, известно, что Митя купил не болгарские марки. Кто купил болгарские марки?

4. В чашке, стакане, кувшине и банке находятся молоко, лимонад, квас и вода. Известно, что вода и молоко не в чашке; сосуд с лимонадом стоит между кувшином и сосудом с квасом; в банке не лимонад и не вода; а стакан стоит между банкой и сосудом с молоком. Где находится вода.

5. Боря, Витя, Гриша и Егор встретились на олимпиаде. Ребята приехали из разных городов: один — из Твери, другой — из Омска, третий — из Томска, четвертый — из Казани. Известно, что Боря жил в одной комнате с мальчиком из Казани и ни один из них никогда не был ни в Твери, ни в Томске. Гриша играл в одной команде с мальчиком из Твери, а против них обычно сражался приятель из Казани. Егор и мальчик из Твери увлекались игрой в шахматы. Определить город, из которого приехал Егор.

6. На столе лежат в ряд четыре фигуры: треугольник, ромб, круг и квадрат. Квадрат, круг, ромб и треугольник вырезаны из белой, синей, красной и зеленой бумаги. Известно, что: круг не белый и не зеленый; синяя фигура лежит между ромбом и красной фигурой; треугольник не синий и не зеленый; квадрат лежит между треугольником и белой фигурой. Из бумаги какого цвета вырезан ромб?

7. Пятеро школьников из пяти различных городов приехали в Смоленск для участия в олимпиаде по математике. "Откуда вы, ребята?" — спросили их. Вот что они ответили: Андреев: "Я приехал из Ярославля, а Григорьев живет в Гагарине". Борисов: "В Гагарине живет Васильев, я прибыл из Вязьмы". Васильев: "Из Ярославля приехал я, а Борисов - из Ельни". Григорьев: Я приехал из Гагарина, а Данилов из Ярцева". Данилов: "Да, я действительно из Ярцева, Андреев живет в Вязьме". В каждом из высказываний одно утверждение правильное, а другое ложное. Откуда приехал Данилов?

8. Три девушки Аня, Варя и Клава ходили на демонстрацию. Одна из них была в красном платье, другая — в белом, третья — в синем. На вопрос, какое было платье на каждой из девушек они дали ответ: Аня была в красном, Варя — не в красном, Клава — не в синем. В этом ответе из трёх частей одна верна, две неверны. В каком платье была каждая из девушек ?

9. Четыре марсианки, оказавшиеся на Земле в 2222 году, на вопрос об их возрасте дали ответы: а) Ми — 22 года, Ме — 21 год; б) Мо — 19 лет, Ми — 21 год; в) Ма — 21 год, Мо — 18 лет. Все марсианки — разных возрастов, притом только таких — 18, 19, 21 и 22. В каждом ответе одна часть истинна, а другая - ложна. Сколько лет каждой из марсианок?

10. В велогонках приняли участие 5 школьников. После гонок 5 болельщиков заявили: 1) Коля занял 1-е место, а Ваня — 4; 2) Сережа занял 2-е место, а Ваня — 4; 3) Сережа занял 2-е место, а Коля — 3; 4) Толя занял 1-е место, а Надя — 4; 5) Надя заняла 3-е место, а Толя — 5. Зная, что одно из показаний каждого болельщика верное, а другое — ложное. Определить какое место занял Толя.

11. Шестеро юношей наблюдали прохожего, затем каждый из них ответил на вопрос, какого цвета его волосы, глаза и костюм, сколько ему лет. Андрей: рыжий, глаза голубые, костюм серый, 34. Вова: блондин, глаза карие, костюм синий, 30. Борис: рыжий, глаза карие, костюм коричневый, 32. Григорий: брюнет, глаза голубые, костюм, во всяком случае, не коричневый, 30. Дмитрий: шатен, глаза чёрные, костюм серый, 28. Евгений: блондин, глаза карие, костюм

синий, 32. Каждый из них трижды ошибся. Но из шести ответов на каждый из вопросов, по меньшей мере, один был верен. Каковы приметы прохожего?

12. Четверо ребят — Алексей, Борис, Иван и Григорий — соревновались в беге. На следующий день на вопрос, кто какое место занял, они ответили так :

Алексей : «Я не был ни первым, ни последним».

Борис : «Я не был последним».

Иван : «Я был первым».

Григорий : «Я был последним».

Известно, что три из этих ответов правильные, а один — неверный.

Кто сказал неправду, и кто был первым на самом деле?

13. Три купчихи — Олимпиада, Матрена и Евдокия — пили чай. Если бы Олимпиада выпила на 5 чашек больше, то она выпила бы столько, сколько две другие вместе. Если бы Матрена выпила бы на 9 чашек больше, то она выпила бы столько, сколько две другие вместе. Сколько каждая из трех купчих выпила чашек и у кого из них какое отчество, если известно, что: Уваровна пила чай вприкуску; количество чашек чая, выпитых Титовной, кратно трем; Карповна выпила 11 чашек.

14. В одном классе учатся три брата: Алексей, Леонид и Александр. Учитель заметил, что:

— если кто-то из них получает подряд две четверки или две тройки, то дальше он учится кое-как и получает тройку;

— если он получает подряд две пятерки, то совсем перестает заниматься и получает двойку;

— если он получает две разные оценки, то следующей будет большая из них.

В начале полугодия Алексей получил оценки 4 и 5, Леонид — 3 и 2, Александр — 2 и 4. Какие итоговые оценки получит каждый из рассматриваемых школьников за данное полугодие, если учитель выставил каждому по 30 оценок, а итоговая оценка — ближайшее целое число к среднему арифметическому полученных оценок?

15. Идет расследование преступления. Трое подозреваемых — Иванов, Петров и Сидоров — дают показания. Чтобы окончательно запутать следствие, каждый из трех подозреваемых называет правильно либо марку, либо цвет машины, на которой преступники скрылись с места преступления. Показания подозреваемых : Иванов сказал, что машина — синие «Жигули». Петров утверждал, что была черная «Волга». Сидоров показал, что был «Мерседес», при этом отрицая, что цвет машины — синий. Определить марку и цвет машины, на которой преступники скрылись с места преступления.

16. Идет расследование преступления. Трое подозреваемых — Иванов, Петров и Сидоров — дают показания. Иванов: «Я этого не делал. Это — Сидоров». Петров: «Сидоров этого не совершал. Это — Иванов». Сидоров: «Ни я, ни Петров этого не совершали». Следствием было установлено, что один из подозреваемых оба показания дал верно, другой — оба неверно, а третий — одно показание дал верно, другое — неверно. Требуется установить виновника преступления.

17. Есть пять коробочек: белая, черная, синяя, красная и зеленая. Шарики тех же цветов, что и коробочки, по два шарика каждого цвета. В каждой коробочке по два шарика. При этом известно, что:

— ни один шарик не лежит в коробочке того же цвета, что и он сам;

— в красной коробочке нет синих шариков;

— в коробочке нейтрального цвета лежит один красный и один зеленый шарик; нейтральный цвет — это либо белый, либо черный;

— в черной коробочке лежат шарики зеленых и синих цветов;

— в одной из коробочек лежат один белый и один синий шарик;

— в синей коробочке находится один черный шарик.

Требуется распределить шарики по коробочкам в соответствии с вышеуказанными условиями.

18. Пять обладателей выигрышных лотерейных билетов должны получить по два приза. На десяти карточках написали номера от одного до десяти и пригласили каждого счастливчика вытянуть по две карточки. К сожалению, при записи результатов произошла ошибка. В то время, как один из членов тиражной комиссии называл вслух числа, стоявшие на извлеченных карточках (например, : «Пять и семь»), другой по рассеянности складывал эти числа и записывал лишь их сумму (в рассмотренном нами примере это было число 12). Поэтому результаты в протоколе записаны так : Петров — 11, Семенов — 4, Иванов — 7, Сидоров — 16, Локтев — 17. Требуется определить, какие призы нужно получить каждому обладателю счастливого билета (номер каждого выигранного им приза), если карточки назад не возвращались. Написать программу, возвращающую результат в виде троек значений: Фамилия, Номер первого приза, Номер второго приза.

19. Дама сдавала в багаж : диван, чемодан, саквояж, картину, корзину, картонку и маленькую собачонку. Диван весил столько же, сколько чемодан и саквояж вместе, и столько же, сколько картина и картонка вместе. Картина, корзина и картонка весили поровну, причем каждая из них — больше, чем собачонка. Когда выгружали багаж, дама заявила, что собака не той породы. При проверке оказалось, что собака перевешивает диван, если к ней на весы добавить саквояж или чемодан. Требуется написать программу, которая доказала бы справедливость претензии дамы.

20. После представления «Ревизора» состоялся следующий диалог (в скобках указаны номера фраз).

Бобчинский : Это вы, Петр Иванович, первый сказали «Э!» (1). Вы сами так говорили (2).

Добчинский : Нет, Петр Иванович, я так не говорил (3). Это вы семгу первый заказали (4). Вы и сказали «Э!» (5). А у меня во рту зуб со свистом (6).

Бобчинский : Что я семгу первый заказал, это верно (7). И верно, что у вас зуб со свистом (8). А все-таки это вы первый сказали «Э!» (9).

Требуется определить, кто первым сказал «Э!»? Известно, что из девяти произнесенных в этом диалоге фраз четное число верных.

21. Четыре человека с именами: Аня, Борис, Костя, Дима — имеют определенные предпочтения. Предпочтения в цвете: красный, синий, желтый, зеленый. Предпочтения в еде: картофель, стейк, макароны, капуста. Также каждый из них идеализирует кого-либо из группы (возможно и себя). Определите, какой цвет, еда и идеал соответствует каждой из персон, если: Аня предпочитает картофель; тот, кто предпочитает красный цвет, любит стейк; один из них идеализирует самого себя; тот, предпочитает желтый цвет, идеализирует Диму; персона, которая предпочитает макароны, идеализируется тем, кто предпочитает синий цвет; персона, которая идеализирует Костю, любит картофель; персона, которая предпочитает синий цвет, идеализирует того, кто любит желтый цвет.

22. Четыре юных филателиста: Митя, Толя, Петя и Саша — купили почтовые марки. Каждый из них покупал марки только одной страны, причем двое из них купили российские марки, один — болгарские и один — чешские. Известно, что Митя и Толя купили марки двух разных стран. Марки разных стран купили Митя с Сашей, Петя с Сашей, Петя с Митей и Толя с Сашей. Кроме того, известно, что Митя купил не болгарские марки. Кто купил чешские марки?

23. В велогонках приняли участие 5 школьников. После гонок 5 болельщиков заявили: 1) Коля занял 1-е место, а Ваня — 4; 2) Сережа занял 2-е место, а Ваня — 4; 3) Сережа занял 2-е место, а Коля — 3; 4) Толя занял 1-е место, а Надя — 4; 5) Надя заняла 3-е место, а Толя — 5. Зная, что одно из показаний каждого болельщика верное, а другое — ложное. Определить, какое место занял Коля.

24. В чашке, стакане, кувшине и банке находятся молоко, лимонад, квас и вода. Известно, что вода и молоко не в чашке; сосуд с лимонадом стоит между кувшином и сосудом с квасом; в банке не лимонад и не вода; а стакан стоит между банкой и сосудом с молоком. Где находится квас?

25. Пятеро школьников из пяти различных городов приехали в Смоленск для участия в олимпиаде по математике. "Откуда вы, ребята?" — спросили их. Вот что они ответили: Андре-

ев: "Я приехал из Ярославля, а Григорьев живет в Гагарине". Борисов: "В Гагарине живет Васильев, я прибыл из Вязьмы". Васильев: "Из Ярославля приехал я, а Борисов — из Ельни". Григорьев: Я приехал из Гагарина, а Данилов из Ярцева". Данилов: "Да, я действительно из Ярцева, Андреев живет в Вязьме". В каждом из высказываний одно утверждение правильное, а другое ложное. Откуда приехал Андреев?

26. На столе лежат в ряд четыре фигуры: треугольник, ромб, круг и квадрат. Квадрат, круг, ромб и треугольник вырезаны из белой, синей, красной и зеленой бумаги. Известно, что: круг не белый и не зеленый; синяя фигура лежит между ромбом и красной фигурой; треугольник не синий и не зеленый; квадрат лежит между треугольником и белой фигурой. Какая фигура вырезана из зеленой бумаги?

27. Три друга — Петр, Роман и Сергей — учатся на математическом, физическом и химическом факультетах. Если Петр — математик, то Сергей не физик. Если Роман не физик, то Петр — математик. Если Сергей не математик, то Роман - химик. Какая специальность у Петра.

28. Боря, Витя, Гриша и Егор встретились на олимпиаде. Ребята приехали из разных городов: один — из Твери, другой — из Омска, третий — из Томска, четвертый — из Казани. Известно, что Боря жил в одной комнате с мальчиком из Казани и ни один из них никогда не был ни в Твери, ни в Томске. Гриша играл в одной команде с мальчиком из Твери, а против них обычно сражался приятель из Казани. Егор и мальчик из Твери увлекались игрой в шахматы. Определить город, из которого приехал Витя.

2.4. Порядок выполнения лабораторной работы

2.4.1. Изучить методы представления и решения CSP задач по лекционному материалу и книгам [1,2,8].

2.4.2. Ознакомиться по лекционному материалу и книгам [1,2] с объявлением операторов, с расширенным перечнем предикатов обработки списков, метаусловиями и отрицанием в языке Пролог. Изучить примеры применения этих средств Пролога для решения CSP задач, которые приведены в п. 2.2 настоящей лабораторной работы.

2.4.3. Ознакомиться с вариантом задания. Сформулировать задачу в терминах задач CSP (см. п. 2.2.1):

а) определить перечень переменных задачи;

б) специфицировать области определения каждой переменной;

в) определить отношения ограничения между переменными;

2.4.4. Ознакомиться с примером кода, приведенного в приложении Б, и, по аналогии, определить на языке Пролог необходимые предикаты для решения поставленной задачи.

2.4.5. Создать в среде программирования Пролог-проект и выполнить его отладку.

2.4.6. Исследовать свойства разработанной программы. Оценить количество просматриваемых сочетаний переменных, определить с помощью предиката **time** статистику выполнения программы. Сформулировать варианты улучшения программы. Предложить альтернативные методы решения задачи.

2.4.7. Зафиксировать результаты работы программы в виде экранных копий.

2.5. Содержание отчета

Цель работы, вариант задания, описание постановки задачи в терминах удовлетворения ограничений, выбор метода решения задачи с обоснованием, описание разработанной программы, описание машинных экспериментов с программой и анализ результатов, выводы.

2.6. Контрольные вопросы

2.6.1. Приведите формальное определение задачи удовлетворения ограничений.

2.6.2. Что понимают под состоянием SCP задачи, совместимым присваиванием, полным присваиванием, решением?

2.6.3. Приведите пример формулировки задачи раскрашивания карты.

2.6.4. Что понимают под графом ограничения?

2.6.5. Как классифицируются переменные CSP задач?

2.6.6. Как классифицируются ограничения CSP задач?

2.6.7. Объясните метод «генерируй и тестируй».

2.6.8. Объясните поиск с возвратами при решении CSP задач.

2.6.9. Объясните метод предварительной (опережающей) проверки.

2.6.10. Объясните метод распространения ограничения.

2.6.11. В чем суть ограничения *Alldiff* ?

2.6.12. Как на Прологе реализуется прямой метод «генерируй и тестируй»? Какие предикаты Пролога обычно используют для генерации вариантов решений?

2.6.13. Приведите определения предикатов **удалить** и **перестановка.** Приведите примеры их использования.

2.6.14. Объясните все примеры решений задачи раскрашивания карты на Прологе, приведенные в п. 2.2.3.

2.6.15. Объясните пример решения простейшей логической задачи на Прологе при выполнении целей в прямом (**генерировать_решение(X), проверить_ограничения(X))** и обратном порядках.

2.6.16. Как средствами Пролога определить основные статистические параметры выполнения программы?

3. ЛАБОРАТОРНАЯ РАБОТА № 3 «РАЗРАБОТКА И ИССЛЕДОВАНИЕ ЭКСПЕРТНОЙ СИСТЕМЫ»

3.1. Цель работы

Разработка экспертной системы продукционного типа на Прологе, исследование базовых принципов организации экспертных систем.

3.2. Краткие теоретические сведения

3.2.1. Структура экспертных систем

Под *экспертной системой* (ЭС) понимают программную систему, аккумулирующую знания эксперта в определенной области и вырабатывающую решения и рекомендации на уровне эксперта. Перечень типовых задач, решаемых ЭС в самых различных областях, включает [1,9]:

- интерпретацию — извлечение информации из первичных данных (распознавание образов, определение состава вещества и др.);
- диагностику — обнаружение неисправностей и причин их появления в некоторой системе (медицинской, механической, электронной и др.);
- мониторинг — непрерывная интерпретация данных в реальном времени с сигнализацией о выходе тех или иных параметров за допустимые пределы (контроль движения транспорта, наблюдение за состоянием энергетических объектов и др.);
- прогноз — предсказание вероятных последствий на основе прошедших и настоящих событий (предсказание погоды, прогноз ситуаций на финансовых рынках и др.);
- планирование — определение последовательности действий, направленных на достижение заранее поставленных целей (планирование поведения роботов, составление маршрутов движения транспорта и др.);
- проектирование — определение конфигурации системы при заданных ограничениях (синтез электронных схем, оптимальное размещение объектов в ограниченном пространстве и др.);
- отладку и ремонт — выполнение последовательности действий по приведению той или иной системы к требуемым режимам функционирования (помощь при отладке программного обеспечения, ремонт инженерных коммуникаций и др.);
- обучение — интерпретация, диагностика и коррекция знаний и умений обучаемого;
- управление — формирование управляющих воздействий, определяющих поведение сложных систем (управление воздушным транспортом, военными действиями, деловой активностью в сфере бизнеса и др.).

Типовая архитектура ЭС изображена на рисунке 3.1. Взаимодействие с ЭС осуществляется с помощью интерфейса. Кроме обеспечения взаимодействия с различными категориями пользователей, интерфейс (если необходимо) выполня-

ет функции сопряжения с внешними объектами: базами данных, различными датчиками, коммуникационным оборудованием и т.п.

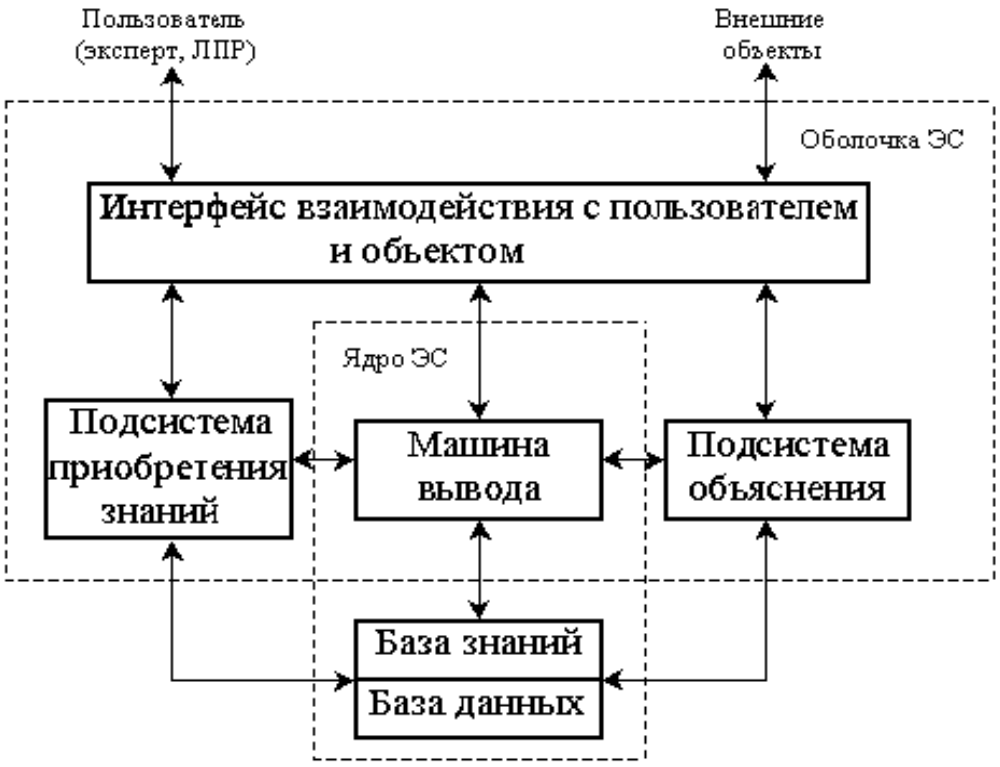

Рисунок 3.1 — Структурная схема ЭС

Ядро ЭС образует база данных, база знаний и машина вывода. *База данных* представляет собой рабочую память, в которой хранятся текущие данные, заключения и другая информация, имеющая отношение к анализируемой системой ситуации. *База знаний* обеспечивает хранение знаний, представленных с помощью одной из моделей: логической, продукционной, фреймовой, сетевой и др.

Машина вывода (подсистема поиска решений), используя данные и знания, организует управление выводом в соответствии с используемой моделью представления знаний. Целью вывода является получение заключений, согласующихся с той информацией, которая содержится в базе знаний.

Подсистема объяснения ЭС позволяет пользователю выяснить, как система получила решение задачи, и какие знания были при этом использованы.

Подсистема приобретения знаний используется как с целью автоматизации процесса наполнения ЭС знаниями, так и при корректировке базы знаний, при ее обновлении, пополнении или исключении элементов знаний.

3.2.2. Методы вывода в ЭС продукционного типа

При создании ЭС используют различные модели представления знаний: логические, продукционные, фреймовые, сетевые [1,4,5,9]. Наибольшее распространение получили ЭС продукционного типа , в которых база знаний представляется в виде набора правил-продукций: “если *A*, то *B* “. Здесь *A* и *B* могут пониматься как “ситуация — действие”, “причина — следствие”, “условие — заключение” и т.п.

В общем случае продукционная система включает следующие компоненты (рис. 2.2): базу продукционных правил, базу данных (рабочую память) и интерпретатор.

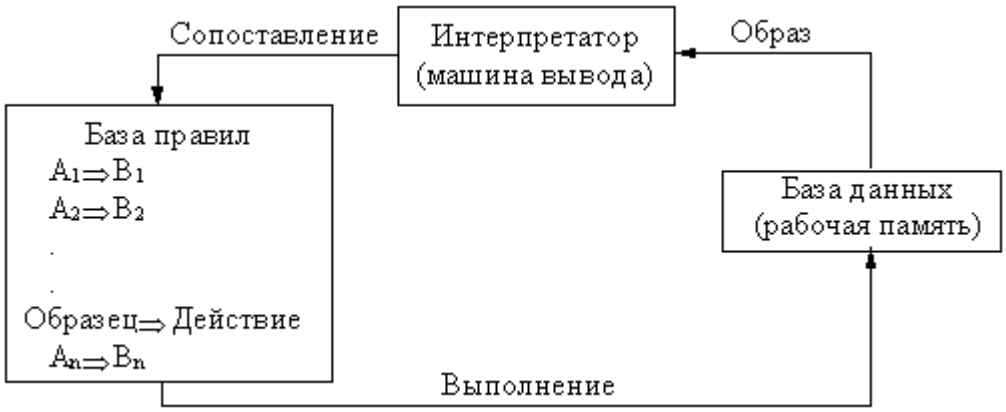

Рисунок 3.2 — Продукционная система

Вывод осуществляется посредством *поиска и сопоставления по образцу*. *Образец* — это некоторая информационная структура, определяющая общие условия, при которых происходит вызов (активизация) правила.

Логический вывод реализуется интерпретатором, который выбирает и активизирует правила. Работу интерпретатора можно описать последовательностью из четырех шагов:

- выбор элементов данных (фактов) из рабочей области;
- сопоставление фактов с образцами правил, если сопоставимы несколько правил, то интерпретатор формирует конфликтный набор правил;
- разрешение конфликтов;
- выполнение выбранного правила.

В продукционных системах применяются два метода вывода — прямой вывод и обратный вывод.

Прямой вывод начинается с задания исходных данных решаемой задачи, которые фиксируются в виде фактов в рабочей памяти системы. Правила, предпосылки которых сопоставимы с исходными фактами, обеспечивают генерацию новых фактов, добавляемых в рабочую память. Процесс применения правил к новым фактам продолжается, пока не будет получено целевое состояние рабочей памяти [1,5,10] .

Обратный вывод, т.е. вывод управляемый целевыми условиями, начинается с внесения целевого утверждения в рабочую память. Затем отыскивается правило-продукция, заключение которого сопоставимо с целью. Условия применения данного правила помещаются в рабочую память и становятся новой подцелью. Процесс повторяется до тех пор, пока в рабочей памяти не будут найдены необходимые факты, подтверждающие все подцели [1,5,9] .

Отметим, что ситуация, когда в рабочую память заранее вносятся все известные факты, встречается редко. Обычно в рабочей памяти изначально содержится только часть фактов, необходимых для вывода. Остальные сведения система выясняет сама, задавая вопросы пользователю.

3.2.3. Формирование объяснений

Для получения объяснений выводов пользователи ЭС могут задавать вопросы двух типов:

- *почему* система сочла необходимым задать пользователю определенный вопрос;
- *как* система пришла к соответствующему заключению.

Выясним механизм формирования ответов на эти вопросы на примере *обратного вывода*. Пусть имеются правила для диагностики неисправностей электрической плиты, изображенные на рис.3.3. Здесь в состав правил включены утверждения, записанные в форме предикатов, например: **лампа(светится), плита(холодная)** и др.

```
правило1 :: если лампа(светится) и плита(холодная)
            то нагреватель(неисправен).
правило2 :: если тока(нет)
            то выключатель(не_включен).
правило3 :: если тока(нет)
            то напряжения(нет).
правило4 :: если плита(холодная) и лампа(не_светится)
            то тока(нет).
правило5 :: если лампа(не_светится) и плита(горячая)
            то лампа(неисправна).
```

Рисунок 3.3. — Правила диагностики неисправностей электрической плиты

Процесс вывода начинается с ввода в рабочую память возможной гипотезы (причины) неисправности, например, **гипотеза(X)**, где **X** – переменная, которая может быть сопоставлена с любой из возможных причин неисправностей, известных системе: **выключатель(не_включен), нагреватель(неисправен), напряжения(нет), лампа(неисправна).**

Если гипотезы проверяются в порядке их записи, то на первом шаге **X** получит значение **выключатель(не_включен)** и система выберет правило 2, которое своим заключением подтверждает эту гипотезу. Далее в соответствии с предпосылками правила 2 система попытается установить справедливость утверждения **тока(нет).** В соответствии с правилом 4 это заключение верно, если имеются факты**: плита(холодная) и лампа(не_светится).** Так как эти факты в рабочей памяти не содержатся, и нет правил, с помощью которых система могла бы установить их значения, то пользователю будут заданы соответствующие вопросы (ответы пользователя набраны курсивом):

плита(холодная)?
:-да.
лампа(не_светится)?
:-да.

После этого система сообщает причину неисправности плиты: **решение: выключатель(не_включен).**

Если пользователь желает выяснить, почему система задает тот или иной вопрос, то в качестве ответа на вопрос вводится слово "*почему*". Например:

лампа(не_светится)?
:-почему.
пытаюсь доказать лампа(не_светится)
с помощью правила: правило4
лампа(не_светится)?
:-

В этом случае система демонстрирует пользователю номер текущего правила, с которым был связан заданный вопрос.

Чтобы ответить на вопрос "*как*" в системе сохраняется *дерево решений*, принятых в течение сеанса работы. Просматривая дерево, можно выяснить, достижение каких подцелей привело к данному решению. Для этого после предъявления решения пользователю система может объяснить, как она его получила. Например, возможен следующий сценарий:

решение: выключатель(не_включен)
объяснить ? [цель/нет]:
:-выключатель(не_включен).
выключатель(не_включен) было доказано с помощью
правило2 :: если тока(нет) то выключатель(не_включен)
объяснить ? [цель/нет]:
:-тока(нет).
тока(нет) было доказано с помощью
правило4 :: если плита(холодная) и лампа(не_светится) то тока(нет)
объяснить ? [цель/нет]:
:-плита(холодная).
плита(холодная) было введено пользователем
объяснить ? [цель/нет]:
:-нет.

Таким образом, формирование ответа на вопрос "как" соответствует просмотру дерева решений в направлении от целевого утверждения до листьев дерева, соответствующих введенным фактам.

Сложнее обстоит дело при прямом выводе, т.е. выводе, управляемом данными. Прямая цепочка рассуждений обеспечивает пользователя менее полезной информацией. Обусловлено это тем, что на промежуточных этапах вывода трудно судить, куда ведет цепочка рассуждений. Так, в ответ на вопрос "почему" пользователю демонстрируется текущее правило. Дальнейшие объяснения не могут быть получены, пока не выполнится следующее правило. Кроме этого, сложно формировать полные ответы на вопрос "как", даже после достижения цели. Информация, которая предоставляется, ограничивается списком выполненных правил.

3.3.4. Реализация продукционной ЭС на языке Пролог

Простейший способ построения ЭС на Прологе — использование в качестве интерпретатора механизма поиска решений, заложенного в Пролог-системе [1,2,4]. Проиллюстрируем это на примере задачи диагностики неисправности электрической плиты. Представим правила, изображенные на рис. 3.3., в виде предикатов языка Пролог:

```
/* структура правила:  причина:– неисправность */
нагреватель(неисправен):– лампа(светится),  плита(холодная).    %правило 1
выключатель(не_включен):– тока(нет).                            %правило 2
напряжения(нет):– тока(нет).                                    %правило 3
тока(нет):– плита(холодная),  лампа(не_светится).               %правило 4
лампа(неисправна):– лампа(не_светится), плита(горячая).         %правило 5
```

Определить причину неисправности плиты можно, указав перечень гипотетических неисправностей и обеспечив проверку выполнимости каждой из гипотез:

```
лампа(не_светится).                    % признаки неисправности
плита(холодная).

гипотеза (нагреватель(неисправен)).    % гипотезы
гипотеза (выключатель(не_включен)).
гипотеза (напряжения(нет)).
гипотеза (лампа(неисправна)).

найти(X): – гипотеза (X), X.           % проверка гипотез/
```

Для того чтобы узнать, какая из гипотез подтверждается, необходимо задать вопрос: **найти(X).** Получив такой вопрос, Пролог-система выполнит необходимый обратный поиск с помощью встроенного механизма вывода и вернет ответ:

```
X = выключатель(не_включен);
X = напряжения(нет).
```

Недостатком рассмотренной программы является необходимость внесения наблюдаемых признаков неисправностей непосредственно в текст программы в виде фактов. Устранить указанный недостаток можно, запросив значения соответствующих признаков неисправностей у пользователя в процессе выполнения программы:

```
лампа(не_светится): – спроси('Лампа не светится ?').
плита(холодная): – спроси('Плита холодная ?').
лампа(светится): – спроси('Лампа светится ?').
плита(горячая): – спроси('Плита горячая ?').
спроси(Вопрос): – write(Вопрос), nl,read('да').
```

В этом случае система будет задавать вопросы пользователю относительно неизвестных фактов. Так как ответы пользователя не запоминаются, то возможно

повторение вопросов. Для запоминания ответов следует воспользоваться встроенным предикатом **assert (**см. ниже). Иным существенным недостатком простейшего способа реализации ЭС является запись продукционных правил в виде кода на языке Пролог.

Усовершенствованный вариант реализации ЭС предполагает запись правил более естественной форме, изображенной на рис. 3.3. Чтобы программа могла интерпретировать правила в такой форме, необходимо определить операторы:

```
определить_операторы: – op(920,  xfy, и),  op(950, xfx, то),
                        op(960, fx, если), op(970, xfx, '::').
: – определить_операторы.
```

Тогда для указанных правил (рис. 3.3) перечень возможных гипотез неисправностей запишется в виде:

```
% гипотезы неисправностей представляются структурой – Имя_факта :: Факт
h1 :: гипотеза(нагреватель(неисправен)).
h2 :: гипотеза(выключатель(не_включен)).
h3 :: гипотеза(напряжения(нет)).
h4 :: гипотеза(лампа(неисправна)).
```

Перечислим также признаки неисправностей, значения которых можно запрашивать у пользователя:

```
% признаки, истинность которых можно выяснить у пользователя
q1 :: признак(лампа(светится)).
q2 :: признак(плита(холодная)).
q3 :: признак(лампа(не_светится)).
q4 :: признак(плита(горячая)).
```

Реализуем интерпретатор ЭС в виде предиката **найти(H),** где **H** — возможная гипотеза, которую требуется подтвердить или опровергнуть. При этом возможны четыре случая:

1) гипотеза **H** подтверждается фактом, уже известным системе;

2) гипотеза **H** соответствует следствию одного из правил, тогда для подтверждения гипотезы необходимо доказать справедливость предпосылок правила;

3) гипотеза **H**, доказываемая на некотором шаге вывода, представляет собой конъюнкцию условий правила, т.е. **H1** и **H2**, тогда необходимо доказать достижимость конъюнкции подцелей: **найти(H1)** и **найти(H2)**;

4) гипотеза **H** сопоставима с одним из признаков, на основе которых устанавливаются причины неисправности, тогда необходимо задать соответствующий вопрос пользователю.

Для того чтобы исключить повтор вопросов, ответы пользователя будем запоминать в базе данных в виде фактов **сообщено(Факт, 'да|нет').** Запоминание соответствующих фактов в базе данных будем выполнять с помощью предиката **assert.** Гипотезы, относительно которых можно задавать вопросы, определяются с помощью предиката **запрашиваемая(H).** Такие гипотезы сопоставимы с призна-

ками. С учетом сказанного, интерпретатор опишется следующей совокупностью правил языка Пролог:

```
найти(Н):- Факт :: Н.                                  % случай 1
найти(Н):- Правило :: если Н1 то Н,  найти(Н1).        % случай2

найти(Н1 и Н2):-найти(Н1), найти(Н2).                  % случай 3
найти(Н):- запрашиваемая(Н),                           % случай 4
           сообщено(Н,да).                             % вопрос уже был
найти(Н):- запрашиваемая(Н),                           % случай 4
           not(сообщено(Н,_)),                         % вопроса  не было
           спроси(Н).

запрашиваемая(Н):- Факт :: признак(Н).

спроси(Н):- nl, write(Н), write('?'), nl,              % вывод вопроса
            read(О), ответ(Н,О).                       % ввод ответа
%обработка ответов
ответ(Н,да):-   assert(сообщено(Н,да)), !.             % добавить ответ в БД
ответ(Н,нет):- assert(сообщено(Н,нет)), !, fail.
ответ(Н,_):- write('правильный ответ: да, нет'), nl,
           спроси(Н).                                  % повторить вопрос
```

Вызов интерпретатора с целью проверки всех гипотез выполняется следующим образом:

```
инициализация:-retractall(сообщено(_,_)).
диагностика(Н):-   инициализация, Факт :: гипотеза(Н), найти(Н).
```

Улучшение разработанной ЭС связано с включением возможности ответов на вопросы типа “почему?” и “как?”. Базовый текст программного кода соответствующей ЭС приведен в приложении В.1. Детальное объяснение программы изложено в [1]. В приложение В.2 включен программный код интерпретатора, позволяющего использовать правила, предпосылки которых могут содержать произвольное количество условий. Для этого предпосылки правил представляются в виде списка условий.

3.3. Варианты заданий

Реализовать продукционную экспертную систему в соответствии с номером варианта, указанного в таблице 3.1. При этом количество рассматриваемых объектов предметной области должно быть не менее 10 и характеризующих их атрибутов также — не менее 10. Система должна уметь давать объяснения вывода. Задачу, решаемую ЭС, выбрать самостоятельно с учетом перечня задач, указанного в п.3.2.1, или с помощью каталога ЭС, приведенного в [10].

Таблица 3.1— Предметная область экспертной системы

Вариант	Предметная область
1, 16	Базы данных
2, 17	Транспорт
3, 18	Обучение

4, 19	Компьютерные сети

Продолжение таблицы 3.1.

Вариант	Предметная область
5, 20	Операционные системы
6, 21	Мобильные устройства
7, 22	Микропроцессоры
8, 23	Языки программирования
9, 24	Компьютерные игры
10, 25	Математика
11, 26	Медицина
12, 27	Метеорология
13, 28	Сельское хозяйство
14, 29	Электроника
15, 30	Юриспруденция

3.4. Порядок выполнения лабораторной работы

3.4.1. Изучить модели представления знаний по лекционному материалу и учебным пособиям [1,5,8].

3.4.2. Детально изучить организацию продукционной системы, механизмы прямого и обратного вывода, основы построения подсистемы объяснения, примеры реализации ЭС на Прологе [1,2,4,10].

3.4.3. Выполнить анализ предметной области ЭС в соответствии с вариантом задания:

а) выбрать задачу, решаемую ЭС, и определить цели системы: конечные, промежуточные и вспомогательные;

б) выделить подзадачи, которые следует решить для достижения цели;

в) разработать продукционную базу правил для решения выделенных подзадач.

3.4.4. Ознакомиться с примерами программных кодов, приведенных в приложении В, и, по аналогии, разработать ЭС продукционного типа для решения задачи в заданной предметной области.

3.4.5. Создать в среде программирования Пролог проект ЭС и выполнить его отладку.

3.4.6. Исследовать свойства разработанной системы. Получить протоколы работы системы при доказательстве различных целевых утверждений.

3.4.7. Зафиксировать результаты работы программы в виде экранных копий.

3.5. Содержание отчета

Цель работы, вариант задания, описание предметной области, структура базы знаний и описание продукционных правил, описание разработанного интерпретатора продукционных правил, описание машинных экспериментов с ЭС и анализ результатов, выводы.

3.6. Контрольные вопросы

3.6.1. Приведите определение экспертной системы.

3.6.2. Перечислите типовые задачи, решаемые с помощью ЭС.

3.6.3. Назовите основные компоненты ЭС и объясните их функции.

3.6.4. Назовите основные этапы разработки ЭС, перечислите задачи, решаемые на каждом из этапов.

3.6.5. Объясните термины “приобретение знаний” и “извлечение знаний”.

3.6.6. Сформулируйте обобщенный алгоритм прямого вывода на правилах-продукциях.

3.6.7. Сформулируйте обобщенный алгоритм обратного вывода на правилах-продукциях.

3.6.8. Объясните суть вопросов “почему” и “как”, используемых для формирования объяснений вывода в ЭС.

3.6.9. Объясните на примере механизм формирования ответа на вопрос типа “как”.

3.6.10. Объясните на примере механизм формирования ответа на вопрос типа “почему”.

3.6.11. Как можно представить базу продукционных правил на языке Пролог?

3.6.12. Приведите пример простейшей реализации ЭС на языке Пролог.

3.6.13. Какие правила положены в основу предиката **найти(H)**, осуществляющего обратный вывод на множестве продукционных правил?

3.6.14. Напишите на языке Пролог простейший вариант реализации предиката **найти(H)**.

3.6.15. Напишете на языке Пролог фрагмент кода, обеспечивающий вывод вопросов, задаваемых ЭС, и анализ ответов.

3.6.16. Напишите на языке Пролог фрагмент кода, обрабатывающий вопросы типа «почему».

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бондарев В.Н. Искусственный интеллект: учеб. пособие / В.Н. Бондарев, Ф.Г. Аде.— Севастополь : Изд-во СевНТУ, 2002.— 615с.
2. Братко Иван. Алгоритмы искусственного интеллекта на языке Пролог, 3-е изд.: пер. с англ. / Иван Братко. — М.: Издательский дом «Вильямс», 2004. — 640с.
3. Введение в среду разработки Пролог-приложений Eclipse ProDT: методические указания к лабораторным работам по дисциплине “Методы и системы искусственного интеллекта” для студентов дневной и заочной формы обучения направления 09.03.02 — “Информационные системы и технологии”/ сост. В. Н. Бондарев; СевГУ. — Севастополь : Изд-во СевГУ, 2015. — 28 с.
4. Ин Ц. Использование Турбо-Пролога : пер. с англ. / Ц. Ин, Д. Соломон — М.: Мир, 1993. — 608с.
5. Люгер Дж. Искусственный интеллект: стратегии и методы решения сложных проблем, 4-е изд. : пер. с англ. / Дж. Люгер — М.: Издательский дом «Вильямс», 2003.—864с.
6. Малпас Дж. Реляционный язык Пролог и его применение: пер. с англ. / Дж.Малпас; Под редакцией В.Н. Соболева .— М.: Наука, Гл. ред. физ.-мат. лит.,1990. — 464с.
7. Мельников В.Н. Логические задачи / В.Н. Мельников .— К.; Одесса: Выща шк., 1989.—344с.
8. Рассел С. Искусственный интеллект: современный подход, 2-е изд. : пер. с англ. / С. Рассел, П. Норвиг— М.: Издательский дом «Вильямс» , 2006.— 1408с.
9. Уотермен Д. Руководство по экспертным системам : пер. с англ. / Д. Уотермен. — М.: Мир, 1989. — 388с.
10. SWI-Prolog: Reference Manual [Электронный ресурс]. Режим доступа:— https://www.swi-prolog.org/download/stable/doc/SWI-Prolog-9.2.9.pdf Последний доступ: 27.01.2025. — Название с экрана.

Приложение А
(справочное)
Создание динамической базы данных

```
:-dynamic          %информирует интерпретатор о том, что  определения предикатов
                   %могут изменяться в ходе выполнения программы
сотрудник/8,       % формат: <имя предиката>/<кол-во аргументов>
сотрудник_ф/8.

% первоначальная  база, загружаемая при запуске программы
%          N    Фам.       Имя       Отч.         Отд. Должн.     Филиал  Тел.
сотрудник(101, петренко, сергей,  иванович,    4,   инженер,  керчь, 68-23-45).
сотрудник(102, бенюк,    иван,    николаевич, 2,   оператор, ялта,   11-25-32).
сотрудник(103, григорьев,григорий,васильевич, 2,   оператор, форос,   20-25-32).
сотрудник(104, михайлов, григорий,андрееевич, 2,   оператор, алупка, 68-23-42).
сотрудник(105, бенюк,    иван,    николаевич, 3,   оператор, керчь, 68-23-42).

start:- menu.      %предикат для запуска программы

%0============ отображение меню ==============================================
menu:-
      repeat, nl,
      write('*******************************'),nl,
      write('* 1. Добавление записи в БД   *'),nl,
      write('* 2. Удаление записи из БД    *'),nl,
      write('* 3. Выборка записей из БД    *'),nl,
      write('* 4. Просмотр БД              *'),nl,
      write('* 5. Загрузка БД из файла     *'),nl,
      write('* 6. Сохранение БД в файле    *'),nl,
      write('* 7. Реляционные операции     *'),nl,
      write('* 8. Выход                    *'),nl,
      write('*******************************'), nl ,nl,
      write('Введите номер пункта меню с точкой в конце!!!'),nl,
      read(C),nl,                    %Ввод пункта меню
      proc(C),                       %Запуск процедуры с номером C
      C=8,                           %Если C не равно 8, то авт. возврат к repeat
      !.                             %Иначе успешное завершение
%0-----------------------------------------------------------------------------

%1======= добавление записи в базу данных ===================================
proc(1):-
    write('Ввод завершайте точкой!!! :'),nl,
    write('Введите номер:'),nl, read(N),
    write('Введите фамилию:'),nl, read(Фам),
    write('Введите имя:'),nl, read(Имя),
    write('Введите отчество:'),nl, read(Отч),
    write('Введите отдел:'),nl, read(Отд),
    write('Введите должность:'),nl, read(Должн),
    write('Введите филиал:'),nl,read(Филиал),
    write('Введите телефон:'),nl,read(Тел),
    assertz(сотрудник(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел)), %добавление факта в БД
    write(Фам),write(' был добавлен в БД'),nl,
    write('Введите любой символ'),nl,                    %ожидание ввода литеры
    get0(C).
%1-----------------------------------------------------------------------------

%2========= удаление записи из базы данных ==================================
proc(2):-
    write('Введите номер для удаления сотрудника'), nl,
    read(N),                              %ввод номера сотрудника
    retract(сотрудник(N,_,_,_,_,_,_,_)), %удаление записи о сотруднике
    write('Сотрудник:'),tab(2),
    write(N), tab(2),                     %вывод сообщения об успешном удалении
    write('был успешно удален из БД'),nl,
    write('Введите любой символ'),nl,
```

```
    get0(C),                                %ожидание ввода символа
    !;                                      %отсечение альтернативы и завершение
    write('Такого сотрудника:'),tab(2),     %вывод сообщения о безуспешном удалении
    write('в базе данных нет'),nl,
    write('Введите любой символ'),nl,
    get0(C).                                %ожидание ввода символа
%2--------------------------------------------------------------------------

%3====== выборка записи из базы данных по критерию ============================
%------- выбираются сотрудники с фамилией на Б или Г  -------------------------
proc(3):-
    retractall(flag(_)),                            %удаление фактов - flag(_)
    сотрудник(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел),     %выбор записи о сотруднике
    фамилия_с_буквы(Фам,'бБгГ'),                       %проверка критерия
    assert(flag(1)),                                %запомнить флаг  - запись найдена
    nl,
    write('Номер: '),  write(N),nl,
    write('Фамилия: '), write(Фам), tab(2),
    write('Имя: '), write(Имя), tab(2),
    write('Отчество: '), write(Отч),nl,
    write('Отдел: '), write(Отд), tab(2),
    write('Должность: '), write(Должн),nl,
    write('Филиал: '),write(Филиал), tab(2),
    write('Телефон: '),write(Тел),nl,nl,
    write('Введите любой символ'),nl,
    get0(C1),get0(C2),
    fail;                       %возврат для выбора след. записи
    flag(1),!.                  %если записи были найдены, то завершить успешно

proc(3):-                       %сообщение, если  записи не найдены
    write('В базе нет сотрудников с фамилиями на Б или Г'),nl,
    write('Введите любой символ'),nl,
    get0(C1),get0(C2).

%проверка принадлежности первой буквы фамилии атому из букв: 'бБгГ'
фамилия_с_буквы(Фам,Буквы):-
    name(Фам,[H|T]),            %преобразование фамилии в список кодов букв
    name(Буквы,СписокКодов),    %преобразование атома из букв в список их кодов
    принадлежит(H,СписокКодов).%принадлежит ли первая буква фамилии списку кодов

%проверка принадлежности элемента списку
принадлежит(H,[H|T]).
принадлежит(H,[X|T]):-принадлежит(H,T).
%3--------------------------------------------------------------------------


%4================== просмотр базы данных ====================================
proc(4):-
    сотрудник(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел), %извлечение записи из БД
    nl,
    write('Номер: '),  write(N),nl,             %отображение на дисплее
    write('Фамилия: '), write(Фам), tab(2),     %элементов запаси
    write('Имя: '), write(Имя), tab(2),
    write('Отчество: '), write(Отч),nl,
    write('Отдел: '), write(Отд), tab(2),
    write('Должность: '), write(Должн),nl,
    write('Филиал: '),write(Филиал), tab(2),
    write('Телефон: '),write(Тел),nl,nl,
    write('Введите любой символ'),nl,
    get0(C1),get0(C2),                          %ожидание ввода символа
    fail;                                       %возврат к выбору записи
    true.                                       %завершение - записей больше нет
%4--------------------------------------------------------------------------

%5======== загрузка базы данных из файла ====================================
proc(5):-
      see('d:/db_sotrudnik.dat'),   текущий входной поток - d:/db_sotrudnik.dat
```

```
      retractall(сотрудник(_,_,_,_,_,_,_,_)),%очистка БД от фактов "сотрудник"
      db_load,                                %загрузка БД термами из файла
      seen,                                   %закрытие потока
      write('БД загружена из файла'),nl.

%загрузка термов в БД из открытого вх. потока
db_load:-
      read(Term),                  %чтение терма
      (Term == end_of_file,!;      %если конец файла, то завершение
       assertz(Term),              %иначе добавить терм в конец БД
       db_load).                   %рекурсивный вызов для чтения след. терма
%5-------------------------------------------------------------------------

%6========== сохранение БД в файле ==========================================
proc(6):-
 tell('d:/db_sotrudnik.dat'),                    %открытие вых. потока
 save_db(сотрудник(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел)), %сохранение терма
 told,                                           %закрытие вых. потока
 write('БД скопирована в файл d:/db_sotrudnik.dat'),nl.

%сохранение терма в открытом файле
 save_db(Term):-              %сохранение терма (факта!) Term в БД
      Term,                   %отождествление терма с термом в БД
      write(Term),            %запись терма
      write('.'),nl,          %запись точки в конце терма
      fail;              %неудача с целью поиска след. варианта
      true.              %завершение, если вариантов отождествления нет
%6-------------------------------------------------------------------------

%7============ реализация операций реляционной алгебры ======================
proc(7):-
    nl,
    write('Формирование отношения r1: сотрудники филиала "Ялта" '), nl,
    подмножество_сотрудников(ялта,R1),  %R1 - список сотрудников филиала 1
    список_в_бд(R1),                    %добавление элементов из R1 в базу данных
    вывод_списка(R1),nl,                %вывод списка R1 на экран

    write('Формирование отношения r2: сотрудники филиала "Керчь" '), nl,
    подмножество_сотрудников(керчь,R2),    %R2 - список сотрудников филиала 2
    список_в_бд(R2),                    %добавление элементов из R2 в базу данных
    вывод_списка(R2),nl,                %вывод списка R2 на экран

    write('Объединенное отношение r1_или_r2: '),  nl,
    объединение(Rez1),                  %Rez1 - список сотрудников филиала1 или 2
    вывод_списка(Rez1),nl,

    write('Пересечение отношений r1_и_r2: '),  nl,
    пересечение(Rez2),                  %Rez2 - список сотрудников 2-х филиалов
    вывод_списка(Rez2),nl,

    write('Разность отношений r1-r2: '),  nl,
    разность(Rez3),                     %Rez3-список сотрудников филиала 1 без фил.2
    вывод_списка(Rez3),nl,

    write('Введите любой символ'),nl,
    get0(C).                              %Ожидание ввода символа
%-------------------------------------------------------------------------

%формирование подмножества сотрудников R заданного Филиала
%подмножество R представляется в виде списка термов "сотрудник_ф(...)"
подмножество_сотрудников(Филиал,R):-
   bagof(сотрудник_ф(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел),
         сотрудник(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел), R).

%правило объединения отношений - r1 или r2
%объединяются отношения сотрудник_ф(ялта) и сотрудник_ф(керчь)
объединение_r1_r2(X1,X2,X3,X4,X5,X6,X7,X8):-
      сотрудник_ф(X1,X2,X3,X4,X5,X6,ялта,X8),X7=ялта;
```

```
      сотрудник_ф(X1,X2,X3,X4,X5,X6,керчь,X8),X7=керчь.

%формирование списка Rez из фактов "сотрудник_ф1_или_ф2"
объединение(Rez):-
    bagof(сотрудник_ф1_или_ф2(X1,X2,X3,X4,X5,X6,X7,X8),
          объединение_r1_r2(X1,X2,X3,X4,X5,X6,X7,X8), %условие вкл. в список
          Rez).

%правило пересечения отношений - r1 и r2
%строится пересечение отношений сотрудник_ф(ялта) и сотрудник_ф(керчь)
%пересечение строится только по совпадению ф.и.о. сотрудников
%обозначения аргументов: первая цифра - номер филиала, вторая - номер аргумента
% например: X11 - филиал 1, аргумент 1 (т.е. номер)
%           X27 - филиал 2, аргумент 7 (т.е. название филиала)
%в конец списка аргументов отношения пересечения дописываются:
%филиал2 (X27),номер2(X21),отд2(X25),долж2 (X26)
пересечение_r1_r2(X11,X12,X13,X14,X15,X16,X17,X18,X27,X21,X25,X26):-
        сотрудник_ф(X11,X12,X13,X14,X15,X16,ялта,X18),X17=ялта,
        сотрудник_ф(X21,X12,X13,X14,X25,X26,керчь,X28),X27=керчь.

%формирование списка Rez из фактов "сотрудник_ф1_и_ф2"
пересечение(Rez):-
  bagof(сотрудник_ф1_и_ф2(X11,X12,X13,X14,X15,X16,X17,X18,X27,X21,X25,X26),
        пересечение_r1_r2(X11,X12,X13,X14,X15,X16,X17,X18,X27,X21,X25,X26),
        Rez).

%правило построения разности отношений: r1-r2
%находится разность сотрудники_ф(ялта)- сотрудники_ф(керчь)
%учитываются только сотрудники с совпадающими ф.и.о.
разность_r1_r2(X11,X12,X13,X14,X15,X16,X17,X18):-
  сотрудник_ф(X11,X12,X13,X14,X15,X16,ялта,X18),X17=ялта,
  not(сотрудник_ф(X21,X12,X13,X14,X25,X26,керчь,X28)),X27=керчь.

%построение списка Rez из фактов "сотрудник_ф1_и_не_ф2"
разность(Rez):-
    bagof(сотрудник_ф1_и_не_ф2(X11,X12,X13,X14,X15,X16,X17,X18),
          разность_r1_r2(X11,X12,X13,X14,X15,X16,X17,X18), %условие вкл. в список
          Rez).

%добавление термов из списка [H|T] в БД
список_в_бд([]).
список_в_бд([H|T]):-
      H=сотрудник_ф(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел),
      assertz(сотрудник_ф(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел)),
      список_в_бд(T).                    %Рекурсивный вызов для  след. терма

%вывод элементов списка [H|T] в каждой строке
вывод_списка([]).
вывод_списка([H|T]):-write(H),nl,вывод_списка(T).
%7-------------------------------------------------------------------------

%8============выход=====================================================
proc(8):-write('Досвидания'),nl.
%8-------------------------------------------------------------------------
```

Приложение Б
(справочное)
Примеры решения CSP задач

Б.1. Раскрашивание карты

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Решение задачи о раскрашивании карты
%--------------------------------------------------------------------------------
% Карта представляется в виде списка [A::[B,D], B::[C,D], C::[D,E], D::[E]],
% элементами которого являются структуры, образованные оператором «::».
% Каждая структура состоит из имени региона и списка имен смежных регионов.
% Например, A::[B,D] означает, что регион A имеет границы с регионами B и D.
% В предикате раскрасить_карту(Карта,Список_цветов) список Карта обрабатывается
% в порядке следования элементов. При этом цвет очередного региона выбирается
% с помощью предиката select (удалить) из Списка_цветов, а цвета сопряженных
% регионов назначаются из списка оставшихся цветов. Назначенные цвета действуют в
% пределах всего списка Карта, и тем самым ограничивают области определения
% сопряженных переменных. Что приводит в конечном итоге к сокращению ветвей
% дерева  поиска
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%объявление инфиксного оператора
:- op(100,xfy,'::').

раскрасить_карту([Регион::Список_смежных_регионов|Ост_Регионы],Список_цветов):-
  присвоить_цвет(Регион,Список_цветов,Оставшиеся_цвета),
  присвоить_цвета_смежным_регионам(Список_смежных_регионов,Оставшиеся_цвета),
  раскрасить_карту(Ост_Регионы,Список_цветов).
раскрасить_карту([],_).

присвоить_цвет(Регион,Список_цветов,Оставшиеся_цвета):-
      select(Регион,Список_цветов,Оставшиеся_цвета).
присвоить_цвета_смежным_регионам(Список_смежных_регионов,Оставшиеся_цвета):-
      members(Список_смежных_регионов,Оставшиеся_цвета).

% members(L1,L2)проверяет вхождение эл-тов из списка L1 в список L2
% Если L1 не конкретизирован, а L2 конкретизирован, то members будет
% наоборот назначать значения элементам списка L1, выбирая их из L2
members([X|Xs],Ys):-member(X,Ys), members(Xs,Ys).
members([],Ys).

 /*-----------------------------------------------------------------------------
Пример вызова:
:-time(повторять(раскрасить_карту([A::[B,D], B::[C,D], C::[D,E], D::[E]],
                 [red, green, blue]),1000)).
Результат:
% 72,105 inferences, 0.031 CPU in 0.023 seconds (136% CPU, 2311043 Lips)
A = C, C = red,
B = E, E = green,
D = blue
------------------------------------------------------------------------------*/
```

Б.2. Сравнение решений логической задачи

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Поиск решения логической задачи с ограничения типа Alldiff
%--------------------------------------------------------------------------------
%Три друга заняли 1-ое, 2-ое и 3-е места на чемпионате университета.
%Они имеют разные имена: Майкл,Ричард,Саймон.
%Они разных национальностей: американец,австралиец,израильтянин.
%Любят разные виды спорта: баскетбол,крикет,теннис.

%Ограничения:
%   1)Майкл предпочитает баскетбол и играет лучше, чем американец.
%   2)Саймон - израильтянин и играет лучше теннисиста.
%   3)Игрок в крикет занял первое место.
%Решение представляется в виде списка, упорядоченного в соответствии с местами.
%Каждый элемент списка - структура из трех компонент: Имя::Национальность::Спорт,
%где "::" инфиксный оператор, объявляемый в программе.
%Задача: Определить место, имя, национальность, вид спорта для каждого из друзей.
%--------------------------------------------------------------------------------
%Для сравнения вариантов решений следует оценить время выполнения целей:
% 1) :-time(повторять(решить1(X),1000)).
% 2) :-time(повторять(решить2(X),1000)).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%объявление инфиксного оператора
:- op(100,xfy,'::').

%Вариант 1: генерация решения , а потом проверка выполнения ограничений
решить1(X) :-генерировать_решение(X), проверить_ограничения(X).

%Вариант 2: проверка выполнения ограничений, а потом генерация решения
решить2(X) :-проверить_ограничения(X), генерировать_решение(X).

%формирование очередного кандидата в решения
%в виде списка [P1::N1::S1, P2::N2::S2, P3::N3::S3]
%Решение-кандидат формируется с помощью  перестановок
%имен P, национальностей N и видов спорта S. Всего  6*6*6=218 сочетаний
генерировать_решение([P1::N1::S1, P2::N2::S2, P3::N3::S3]):-
  перестановка([P1,P2,P3],[майкл,ричард,саймон]),            %перестановки имен
  перестановка([N1,N2,N3],[американец,австралиец,израильтянин]),  %национальностей
  перестановка([S1,S2,S3],[баскетбол,крикет,теннис]).         %видов спорта

%проверка выполнения ограничений
проверить_ограничения(Решение) :-
      Решение = [X1,X2,X3],
      member(майкл::_::баскетбол,Решение),                %ограничение 1
      предшествует(майкл::_::_,_::американец::_,Решение),  %ограничение 1
      member(саймон::израильтянин::_,Решение),             %ограничение 2
      предшествует(саймон::_::_,_::_::теннис,Решение),     %ограничение 2
      X1 = _::_::крикет.                                  %ограничение 3

%========предикаты обработки списков========================================

%отношение предшествует(X,Y,L) верно, если  X следует в списке L  раньше  Y
предшествует(X,Y,[X|T]):- member(Y,T).
предшествует(X,Y,[_|T]):- предшествует(X,Y,T).

%перестановка элементов списка
%в начале находим перестановку L1 для хвоста списка L, а
%затем выполняем вставку головы списка H в произвольную позицию L1
перестановка([],[]).
перестановка([X|L],P):-перестановка(L,L1),вставить(X,L1,P).

%вставка элемента X в список L1
%реализована через удаление X из рез. списка L2
вставить(X,L1,L2):-удалить(X,L2,L1).

%удаление элемента списка: удалить(X,L,L1), где L1- это L без X
```

```
удалить(X,[X|T],T).
удалить(X,[H|T],[H|T1]):-удалить(X,T,T1).

%==============вспомогательный предикат=======================================
% цикл повторения выполнения Цели заданное число раз (N)
повторять(Цель,1):-Цель.
повторять(Цель,N):-
      not(not(Цель)),               %стирание предыдущих подстановок
      M is N-1,повторять(Цель,M).
/*-----------------------------------------------------------------------------
 Примеры вызовов:

:-time(повторять(решить1(X),1000)).
% 1,179,051 inferences, 0.312 CPU in 0.324 seconds (96% CPU, 3778985 Lips)
X = [саймон::израильтянин::крикет,
     майкл::австралиец::баскетбол,
     ричард::американец::теннис]

:- time(повторять(решить2(X),1000)).
% 110,051 inferences, 0.047 CPU in 0.049 seconds (96% CPU, 2351502 Lips)
X = [саймон::израильтянин::крикет,
     майкл::австралиец::баскетбол,
     ричард::американец::теннис]

Второй вариант выполняется примерно в 6 раз быстрее!
 -----------------------------------------------------------------------------*/
```

Б.3. Решение задачи «Загадка Эйнштейна»

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Загадка Эйнштейна — известная логическая задача,
% по легенде созданная Альбертом Эйнштейном в годы его детства
%-------------------------------------------------------------------------------
%  На улице стоят пять домов.
%  Каждый из пяти домов окрашен в свой цвет,
%  а их жители — люди разных национальностей,
%  владеют разными животными, пьют разные напитки
%  и курят разные марки американских сигарет.
%-------------------------------------------------------------------------------
%  Ограничения:
%  1. Англичанин живёт в красном доме;
%  2. У испанца есть собака;
%  3. В зелёном доме пьют кофе;
%  4. Русский пьёт чай;
%  5. Зелёный дом стоит сразу справа от белого дома;
%  6. Тот, кто курит Old Gold, разводит улиток;
%  7. В жёлтом доме курят Kools;
%  8. В центральном доме пьют молоко;
%  9. Норвежец живёт в первом доме;
%  10. Сосед того, кто курит Chesterfield, держит лису;
%  11. В доме по соседству с тем, в котором держат лошадь, курят Kools;
%  12. Тот, кто курит Lucky Strike, пьёт апельсиновый сок;
%  13. Японец курит Parliament;
%  14. Норвежец живёт рядом с синим домом.
%  Вопрос:
%  Кто пьёт воду? Кто держит зебру?
%-------------------------------------------------------------------------------
%  Будем представлять решение в виде списка, состоящего из 5 подсписков
%  (по количеству домов): Реш =[[1,N1,C1,P1,D1,S1],…,[5,N5,C5,P5,D5,S5]],
%  где цифра обозначает номер дома, N - национальность,  C- цвет дома,
%  P - животное, D - напиток, S - сигареты
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

эйнштейн:-
      эйнштейн(Решение),
      вывод_табл(Решение).

эйнштейн(Реш) :-
      Реш =[[1,N1,C1,P1,D1,S1],
            [2,N2,C2,P2,D2,S2],
            [3,N3,C3,P3,D3,S3],
            [4,N4,C4,P4,D4,S4],
            [5,N5,C5,P5,D5,S5]],

% 1. Англичанин живет в красном доме
  member([_,англичанин,красный,_,_,_], Реш),
% 2. У испанца есть собака
  member([_,испанец,_,собака,_,_], Реш),
% 3. В зеленом доме пьют кофе
  member([_,_,зеленый,_,кофе,_], Реш),
% 4. Русский пьет чай
  member([_,русский,_,_,чай,_], Реш),
% 5. Зеленый дом стоит сразу справа от белого дома
  member([БД,_,белый,_,_,_], Реш), member([ЗД,_,зеленый,_,_,_], Реш),
  ЗД =:= БД + 1,
% 6. Тот, кто курит Old Gold, разводит улиток
  member([_,_,_,улитки,_,old_gold], Реш),
% 7. В желтом доме курят Kools
  member([_,_,желтый,_,_,kools], Реш),
% 8. В центральном доме пьют молоко
  member([3,_,_,_,молоко,_], Реш),
% 9. Норвежец живет в первом доме
  member([1,норвежец,_,_,_,_], Реш),
% 10.Сосед того, кто курит Chesterfield, держит лису
  member([СД,_,_,_,_,chesterfield], Реш), member([ЛД,_,_,лиса,_,_], Реш),
```

```
  (ЛД =:= СД + 1; ЛД =:= СД - 1),
% 11.В доме по соседству с тем, в котором держат лошадь, курят Kools
  member([КД,_,_,_,_,kools], Реш), member([ЛоД,_,_,лошадь,_,_], Реш),
  (ЛоД =:= КД + 1; ЛоД =:= КД - 1),
% 12.Тот, кто курит Lucky Strike, пьёт апельсиновый сок
  member([_,_,_,_,сок,luckystrike], Реш),
% 13.Японец курит Parliament
  member([_,японец,_,_,_,parliament], Реш),
% 14.Норвежец живёт рядом с синим домом
  member([НД,норвежец,_,_,_,_], Реш), member([СД,_,синий,_,_,_], Реш),
  (НД =:= СД + 1; НД =:= СД - 1),
перестановка([норвежец,русский,англичанин,японец,испанец],[N1,N2,N3,N4,N5]),
перестановка([желтый,синий,красный,зеленый,белый],[C1,C2,C3,C4,C5]),
перестановка([лиса,лошадь,улитки,зебра,собака],[P1,P2,P3,P4,P5]),
перестановка([сок,чай,молоко,кофе,вода],[D1,D2,D3,D4,D5]),
перестановка([kools,old_gold,chesterfield,luckystrike,parliament],[S1,S2,S3,S4,S5]).

% вывод решения в форме таблицы
вывод_табл([A,B,C,D,E]) :-
  H=['N','Национальн.','Цвет дома','Животное','Напиток','Сигареты'],
  write('+--+------------+----------+----------+----------+-------------+'),nl,
  writef('|%2L|%12L|%10L|%10L|%10L|%13L|',H),nl,
  write('+--+------------+----------+----------+----------+-------------+'),nl,
  writef('|%2L|%12L|%10L|%10L|%10L|%13L|',A),nl,
  writef('|%2L|%12L|%10L|%10L|%10L|%13L|',B),nl,
  writef('|%2L|%12L|%10L|%10L|%10L|%13L|',C),nl,
  writef('|%2L|%12L|%10L|%10L|%10L|%13L|',D),nl,
  writef('|%2L|%12L|%10L|%10L|%10L|%13L|',E),nl,
  write('+--+------------+----------+----------+----------+-------------+'),nl.

/* ----------------Решение-----------------------------------------------------
Вызов:
  :-эйнштейн.
Результат:
+--+------------+----------+----------+----------+-------------+
|N |Национальн. |Цвет дома |Животное  |Напиток   |Сигареты     |
+--+------------+----------+----------+----------+-------------+
|1 |норвежец    |желтый    |лиса      |вода      |kools        |
|2 |русский     |синий     |лошадь    |чай       |chesterfield |
|3 |англичанин  |красный   |улитки    |молоко    |old_gold     |
|4 |испанец     |белый     |собака    |сок       |luckystrike  |
|5 |японец      |зеленый   |зебра     |кофе      |parliament   |
+--+------------+----------+----------+----------+-------------+

-------------------------------------------------------------------------------*/
```

Приложение В
(справочное)
Примеры реализации ЭС продукционного типа

В.1. Пример диагностической ЭС

В.1. 1. База знаний диагностической ЭС (файл nb.pl.)

```
info:-
      nl,
      write('*******************************'),nl,
      write('*     Экспертная система      *'),nl,
      write('*      для диагностики        *'),nl,
      write('*       неисправностей        *'),nl,
      write('*---------------------------- *'),nl,
      write('*   Отвечайте на вопросы:     *'),nl,
      write('*    да, нет, почему          *'),nl,
      write('*   Для объяснения решения    *'),nl,
      write('*     введите цель            *'),nl,
      write('*******************************'), nl ,
      write('Введите любой символ'),nl,   %Ожидание ввода литеры
      get0(_).

% тестовая база продукционных правил для диагностики электрической плиты
правило1 :: если лампа(светится) и плита(холодная)
            то нагреватель(неисправен).
правило2 :: если тока(нет)
            то выключатель(не_включен).
правило3 :: если тока(нет)
            то напряжения(нет).
правило4 :: если плита(холодная) и лампа(не_светится)
            то тока(нет).
правило5 :: если лампа(не_светится) и плита(горячая)
            то лампа(неисправна).

% гипотезы неисправности (цели)
h1 :: гипотеза(нагреватель(неисправен)).
h2 :: гипотеза(выключатель(не_включен)).
h3 :: гипотеза(напряжения(нет)).
h4 :: гипотеза(лампа(неисправна)).

% признаки, истинность которых можно выяснить у пользователя
q1 :: признак(лампа(светится)).
q2 :: признак(плита(холодная)).
q3 :: признак(лампа(не_светится)).
q4 :: признак(плита(горячая)).
```

В.1.2. Интерпретатор ЭС для диагностической базы знаний

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Интерпретатор (машина вывода) для ЭС продукционного типа
% Метод вывода: обратный вывод
% Вариант 1: интерпретатор обрабатывает правила, в  которых
% предпосылки задаются в виде условий (не более 2-х), соединенных оператором "и".
% -------------------------------------------------------------------------------
% Примеры правил см. в загружаемой тестовой  базе знаний - nb.pl
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:-dynamic
сообщено/2.
% объявление операторов
определить_операторы:-
           op(920, xfy, и),
           op(950, xfx, то),
           op(960, fx, если),
           op(970, xfx, '::').
:-определить_операторы.

%============обратный вывод=====================================================
% реализуется предикатом найти(Н,Стек,Д), где Н - проверяемая гипотеза (цель),
% Стек - стек из имен доказываемых гипотез и правил (используется при ответе на
% вопросы "почему"), Д - дерево вывода целевого утверждения (используется при отве-
% те на вопросы "как"). Предикат получает на вход Н и Стек=[Н] и в процессе обрат-
% ного вывода строит дерево вывода Д.
%-------------------------------------------------------------------------------

% случай1:если цель Н была подтверждена пользователем,
% то дерево вывода Д=сообщено(Н).
найти(Н,Стек,сообщено(Н)):-сообщено(Н,да).
% если цель - это признак, то спроси его значение
найти(Н,Стек,сообщено(Н)):-запрашиваемая(Н),
                           not(сообщено(Н,_)),спроси(Н,Стек).

% случай2:если цель Н подтверждается фактом, уже известным системе,
% то дерево вывода Д=Факт :: Н
найти(Н,Стек,Факт :: Н):-Факт :: Н.

% случай3: если цель Н соответствует следствию одного из
% правил -> Правило :: если Н1 то Н
% и если Д1 дерево вывода для подцели Н1,
% то Д= Правило :: если Д1 то Н и номер правила добавить в Стек
найти(Н,Стек,Правило :: если Д1 то Н):-
                    Правило :: если Н1 то Н,
                    найти(Н1,[Правило | Стек],Д1).

% случай4: если доказывается конъюнкция гипотез Н=Н1 и Н2,
% то дерево вывода Д=Д1 и Д2, где Д1,Д2 - деревья вывода гипотез Н1 и Н2
найти(Н1 и Н2,Стек,Д1 и Д2):-
            найти(Н1,Стек,Д1),найти(Н2,Стек,Д2).

% проверка: является ли гипотеза признаком, значение которого можно спросить
запрашиваемая(Н):-Факт :: признак(Н).

%=========вывод вопросов и обработка ответов "да, нет, почему" =================
% вывод вопроса и ввод ответа
спроси(Н,Стек):-write(Н),write('?'),nl,
                read(О),ответ(Н,О,Стек).

% обработка ответов: да, нет.
ответ(Н,да,Стек):-assert(сообщено(Н,да)),!.
ответ(Н,нет,Стек):-assert(сообщено(Н,нет)),!,fail.

% обработка ответов - "почему"
% случай1: стек целей пустой
ответ(Н,почему,[]):-!,write(' Вы задаете слишком много вопросов'),nl,
                    спроси(Н,[]).
%случай2: в стеке только первая введенная цель, т.е. доказываемая гипотеза
```

```
ответ(H,почему,[H1]):-!,write('моя гипотеза: '),
                       write(H1),nl,спроси(H,[]).
% случай3: если в стеке несколько элементов, то вывод заключения (т.е. подцели)
% и номера текущего применяемого правила
ответ(H,почему,[Правило | Стек]):-!,
            Правило :: если H1 то H2,
            write('пытаюсь доказать '),
            write(H2),nl,
            write('с помощью правила: '),
            write(Правило),nl,
            спроси(H,Стек).

% неправильный ответ: повторяем вопрос
ответ(H,_,Стек):-write(' правильный ответ: да, нет, почему'),nl,
                 спроси(H,Стек).

%==============обработка ответов на вопросы "как?"==============================
% предикат как(H,Д)- выполняет поиск подцели H в построенном
% с помощью предиката "найти" дереве вывода Д и отображает соответствующий
% фрагмент дерева вывода, объясняя, как было получено доказательство H.
% Дерево вывода Д представляет собой последовательность вложенных правил, напри-
% мер:
% правило2::если (правило4::если сообщено(плита(холодная))и
%                                 сообщено(лампа(не_светится))
%                              то тока(нет))
%           то выключатель(не_включен)
% ------------------------------------------------------------------------------

% поиск целевого утверждения H в дереве
как(H,Дерево):-как1(H,Дерево),!.

% вывод сообщения, если H не найдено в дереве
как(H,_):-write(H),tab(2),write('не доказано'),nl.

% случай1: если H сообщено пользователем,
% то вывести "H было введено"
как1(H,_):-сообщено(H,_),!,
           write(H),write('было введено'),nl.

% случай2: если дерево вывода Д представлено фактом, подтверждающим H
как1(H,Факт :: H):-!,
         write(H), write( 'является фактом'), write(Факт),nl.

% случай3: если дерево вывода Д - правило в заключение, которого есть H,
% то отобразить это правило
как1(H,Правило :: если _ то H):-!,
         write(H),write('было доказано с помощью'),nl,
         Правило :: если H1 то H,
         отобрази_правило(Правило :: если H1 то H).

% случай4: если в дереве Д нет правила с заключением H,
%то поиск H надо выполнять в дереве предпосылок, т.е. в Дерево
как1(H,Правило :: если Дерево то _):-как(H,Дерево).

% случай5: если предпосылки образуют конъюнкцию,
% то выполнить поиск в поддеревьях Дерево1 и Дерево2
как1(H,Правило :: если Дерево1 и Дерево2 то _):-как(H,Дерево1),как(H,Дерево2).

%вывод правила на экран
отобрази_правило(Правило :: если H1 то H):-
         write(Правило), write( ':'),nl,
         write('если '), write(H1), nl,
         write('то '), write(H),nl.

/* ====================Вызов интерпретатора=================================*/
инициализация:-retractall(сообщено(_,_)).
start:-
      /* Загрузка базы знаний из файла */
      reconsult('D:/Eclipse_prolog_w/exp_sys/src/nb.pl'),
      info,                    %отображение информации о базе знаний*
```

```
      go_exp_sys.

go_exp_sys:-  инициализация,
              Факт :: гипотеза(H),
              найти(H,[H],Дерево),
              write('решение:'),write(H),nl,
              объясни(Дерево),
              возврат.


%объяснение вывода утверждения
объясни(Дерево):-write( 'объяснить ? [цель/нет]:'), nl,read(H),
                  (H\=нет,!,как(H,Дерево),объясни(Дерево));!.

%поиск следующих решений
возврат:-nl,write('Искать ещё решение [да/нет]?: '),nl, read(нет).
```

В.1.3. Протокол работы диагностической ЭС

```
:-start.
*****************************
*     Экспертная система    *
*      для диагностики      *
*       неисправностей      *
*-------------------------- *
*   Отвечайте на вопросы:   *
*    да, нет, почему        *
*   Для объяснения решения  *
*     введите цель          *
*****************************
Введите любой символ

лампа(светится)?
:-нет.
плита(холодная)?
:-да.
лампа(не_светится)?
:-да.
решение:выключатель(не_включен)
объяснить ? [цель/нет]:
:-выключатель(не_включен).
выключатель(не_включен)было доказано с помощью
правило2:
если тока(нет)
то выключатель(не_включен)
объяснить ? [цель/нет]:
:-тока(нет).
тока(нет)было доказано с помощью
правило4:
если плита(холодная)и лампа(не_светится)
то тока(нет)
объяснить ? [цель/нет]:
:-плита(холодная).
плита(холодная)было введено
объяснить ? [цель/нет]:
:-нет.
Искать ещё решение [да/нет]?:
:-да.
решение:напряжения(нет)
объяснить ? [цель/нет]:
:-нет.
Искать ещё решение [да/нет]?:
:-да.
плита(горячая)?
:-почему.
пытаюсь доказать лампа(неисправна)
с помощью правила: правило5
плита(горячая)?
:-почему.
моя гипотеза: лампа(неисправна)
плита(горячая)?
:-почему.
 Вы задаете слишком много вопросов
плита(горячая)?
:-нет.
false.
```

В.2 . Пример классифицирующей ЭС

В.2.1. База знаний классифицирующей ЭС (файл new_anim.pl)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% База знаний ЭС игры "Отгадай животное".
% Суть игры состоит в том, чтобы пользователь задумал какое-либо животное,
% а ЭС, задавая вопросы, пытается отгадать его. ЭС используя введенные признаки
% животного, по сути, выполняет классификацию и относит описываемое животное к
% соответствующему виду.
% База знаний, необходимая для проведения игры, содержит сведения о признаках
% различных видов животных
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
info:-
      nl,
      write('*****************************'),nl,
      write('*     Экспертная система      *'),nl,
      write('*   Игра "Отгадай животное    *'),nl,
      write('*                             *'),nl,
      write('*---------------------------- *'),nl,
      write('*   Отвечайте на вопросы:     *'),nl,
      write('*    да, нет, почему          *'),nl,
      write('*   Для объяснения решения    *'),nl,
      write('*     введите цель            *'),nl,
      write('*****************************'), nl ,
      write('Введите любой символ'),nl,     %Ожидание ввода литеры
      get0(_).

% база продукционных правил игры "отгадай животное"
% предпосылки правил задаются в виде списка условий
% [условие1,условие2,...] означает - условие1 и условие2 и...
правило1  ::  если  [имеет(шерсть)]
              то   млекопитающее.
правило2  ::  если [кормит_детенышей(молоком)]
              то   млекопитающее.
правило3  ::  если [имеет(перья)]
              то  птица.
правило4  ::  если  [летает, откладывает_яйца]
              то птица.
правило5  ::  если [млекопитающее,ест_мясо]
              то хищник.
правило6  ::  если [млекопитающее, имеет(острые_зубы),
                    имеет(острые_когти), имеет(прямо_посаженные_глаза)]
              то  хищник.
правило7  ::  если [млекопитающее,имеет(копыта)]
              то  жвачное.
правило8  ::  если [млекопитающее, жует(жвачку)]
              то   жвачное.
правило9  ::  если [хищник, желто-коричневое, имеет(темные_пятна)]
              то гепард.
правило10  :: если [хищник,желто-коричневое,полосатое,имеет(черные_полосы)]
              то тигр.
правило11  :: если [жвачное, длинношеее, длинноногое,
                    желто-коричневое, имеет(темные_пятна)]
              то   жираф.
правило12  :: если [жвачное,полосатое,имеет(черные_полосы)]
              то  зебра.
правило13  :: если [птица,не_летает,длинношеее,длинноногое,черно-белое]
              то  страус.
правило14  :: если [птица,не_летает,плавает,черно-белое]
              то  пингвин.
правило15  :: если  [птица, хорошо_летает]
              то   альбатрос.

% гипотезы
h1  :: гипотеза(гепард).
h2  :: гипотеза(тигр).
```

```
h3 :: гипотеза(жираф).
h4 :: гипотеза(зебра).
h5 :: гипотеза(страус).
h6 :: гипотеза(пингвин).
h7 :: гипотеза(альбатрос).

% признаки животных, истинность которых можно выяснить у пользователя
q1 :: признак(имеет(шерсть)).
q2 :: признак(кормит_детенышей(молоком)).
q3 :: признак(имеет(перья)).
q4 :: признак(летает).
q5 :: признак(откладывает_яйца).
q6 :: признак(ест_мясо).
q7 :: признак(имеет(острые_зубы)).
q8 :: признак(имеет(острые_когти)).
q9 :: признак(имеет(прямо_посаженные_глаза)).
q11 :: признак(имеет(копыта)).
q12 :: признак(жует(жвачку)).
q13 :: признак(желто-коричневое).
q14 :: признак(имеет(темные_пятна)).
q15 :: признак(полосатое).
q16 :: признак(имеет(черные_полосы)).
q17 :: признак(длинношеее).
q18 :: признак(длинноногое).
q19 :: признак(не_летает).
q20 :: признак(плавает).
q21 :: признак(черно-белое).
q22 :: признак(хорошо_летает).
```

В.2.2. Интерпретатор классифицирующей ЭС

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Интерпретатор (машина вывода) для ЭС продукционного типа
% Метод вывода: обратный вывод
% Вариант 2: интерпретатор обрабатывает правила, в  которых
% предпосылки задаются в виде списка условий.
% Это позволяет в условной части правила, задавать произвольное
% количество условий.
% -------------------------------------------------------------------------------------
% Примеры правил см. в загружаемой тестовой  базе знаний - new_anim.pl
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-dynamic
сообщено/2.
определить_операторы:-
            op(950, xfx, то),
            op(960, fx, если),
            op(970, xfx, '::').
:-определить_операторы.

%============обратный вывод=================================================================
% реализуется предикатом найти(S,Стек,Д),где S - список проверяемых гипотез,
% Стек - стек из имен доказываемых гипотез и правил (используется при ответе на
% вопросы "почему"), Д - дерево вывода целевого утверждения (используется при отве-
% те на вопросы "как"). Предикат получает на вход список [H] и Стек=[H] и в про-
% цессе обратного вывода строит дерево вывода Д.
% Предикат "найти" для доказательства отдельных гипотез из списка S
% использует предикат найти1(H,Стек,Дерево).
%-------------------------------------------------------------------------------------

% случай1:если цель H была подтверждена пользователем,
% то дерево вывода Д=сообщено(H).
найти1(H,Стек,сообщено(H)):-сообщено(H,да).
найти1(H,Стек,сообщено(H)):-запрашиваемая(H),
                              not(сообщено(H,_)),спроси(H,Стек).

% случай2:если цель H подтверждается фактом, уже известным системе,
% то дерево вывода Д=Факт :: H
найти1(H,Стек,Факт :: H):-Факт :: H.

% случай3: если цель H соответствует следствию одного из
% правил -> Правило :: если H1 то H
% и если Д1 дерево вывода для подцели H1,
% то Д= Правило :: если Д1 то H и добавить № правила в Стек
найти1(H,Стек,Правило :: если Д1 то H):-
                      Правило :: если H1 то H,
                      найти(H1,[Правило | Стек],Д1).

% случай4: если доказывается конъюнкция гипотез, заданная списком гипотез,
% то найти доказательство первой гипотезы H1 из списка
% с помощью найти1(H1,Стек,Дерево1),  а затем найти доказательство оставшихся
% гипотез T с помощью найти(T,Стек,Дерево) и
% объединить деревья вывода в общий список [Дерево1 | Дерево].
найти([],Стек,Дерево):-Дерево=[].
найти([H1|T],Стек,[Дерево1 | Дерево]):-
                    найти1(H1,Стек,Дерево1),найти(T,Стек,Дерево).

% проверка: является ли гипотеза признаком, значение которого можно спросить
запрашиваемая(H):-Факт :: признак(H).

%=========вывод вопросов и обработка ответов "да, нет, почему" =================
%вывод вопроса и ввод ответа
спроси(H,Стек):-write(H),write('?'),nl,
                 read(O),ответ(H,O,Стек).

%обработка ответов: да, нет
ответ(H,да,Стек):-assert(сообщено(H,да)),!.
ответ(H,нет,Стек):-assert(сообщено(H,нет)),!,fail.
```

```
%обработка ответов - "почему"
% случай1: стек целей пустой
ответ(Н,почему,[]):-!,write(' Вы задаете слишком много вопросов'),nl,
                    спроси(Н,[]).
%случай2: в стеке осталась только первая введенная цель, т.е доказываемая гипотеза
ответ(Р,почему,[Н]):-!,write('моя гипотеза: '),
                      write(Н),nl,спроси(Р,[]).

%случай3: вывод заключения и номера правила для доказываемой текущей подцели Н
ответ(Н,почему,[Правило | Стек]):-!,
            Правило :: если Н1 то Н2,
            write('пытаюсь доказать '),
            write(Н2),nl,
            write('с помощью правила: '),
            write(Правило),nl,
            спроси(Н,Стек).

%неправильный ответ: повторяем вопрос
ответ(Н,_,Стек):-write(' правильный ответ: да, нет, почему'),nl,
   спроси(Н,Стек).

%==============обработка ответов на вопросы "как?"==============================
% предикат как(Н,Д)- выполняет поиск подцели Н в построенном
% с помощью предиката "найти" дереве вывода Д и отображает соответствующий
% фрагмент дерева вывода, объясняя, как было получено доказательство Н.
% Дерево вывода Д представляет собой последовательность вложенных правил
% в виде списка, например:
% [правило5::если[правило1::если[сообщено(имеет(шерсть))]то млекопитающее,
%                 сообщено(ест_мясо)]то хищник,...]
%-------------------------------------------------------------------------------

% поиск целевого утверждения Н в дереве
как(Н,Дерево):-как1(Н,Дерево),!.

% вывод сообщения, если Н не найдено
как(Н,_):-write(Н),tab(2),write('не доказано'),nl.

% случай1: если Н сообщено пользователем,
% то вывести "Н было введено"
как1(Н,_):-сообщено(Н,_),!,
           write(Н),write('было введено'),nl.

% случай2: если дерево вывода Д представлено фактом, подтверждающим Н
как1(Н,Факт :: Н):-!,
         write(Н), write( 'является фактом'), write(Факт),nl.

% случай3: если дерево вывода Д - правило в заключение, которого есть Н,
% то отобразить это правило
как1(Н,[Правило :: если _ то Н]):-!,
         write(Н),write(' было доказано с помощью'),nl,
         Правило :: если Н1 то Н,
         отобрази_правило(Правило :: если Н1 то Н).

% случай4: если в дереве Д нет правила с заключением Н,
%то поиск Н надо выполнять в дереве вывода предпосылок, т.е. в Дерево
как1(Н,[Правило :: если Дерево то _]):-как(Н,Дерево).


% случай5: если дерево вывода - список поддеревьев вывода
% каждой  конъюнктивной подцели правила из БЗ,
% то поиск Н следует выполнять в каждом из поддеревьев;
% поиск Н следует выполнять сначала в  поддереве [Д1], а
% если Н не найдено, то продолжить поиск  в оставшихся поддеревьях
как1(Н,[]):-!.
как1(Н,[Д1|Д2]):-как(Н,[Д1]),!;
                 как1(Н,Д2).

%вывод правила на экран
отобрази_правило(Правило :: если Н1 то Н):-
```

```
          write(Правило), write( ':'),nl,
          write('если '), write(Н1), nl,
          write('то '), write(Н),nl.

/* Вызов интерпретатора*/
инициализация:-retractall(сообщено(_,_)).
start:-
        /* Загрузка базы знаний из файла*/
      reconsult('D:/Eclipse_prolog_w/exp_sys/src/new_anim.pl'),
      info,                %отображение информации о базе знаний*
      go_exp_sys.

go_exp_sys:-  инициализация,
              Факт :: гипотеза(Н),
              найти([Н],[Н],Дерево),
              write('решение:'),write(Н),nl,
              объясни(Дерево),
              возврат.

%объяснение вывода утверждения
объясни(Дерево):-write( 'объяснить ? [цель/нет]:'), nl,read(Н),
                 (Н\=нет,!,как(Н,Дерево),объясни(Дерево));!.

%поиск следующих решений
возврат:-write('Искать ещё решение [да/нет] ?: '),nl, read(нет).
```

В.2.2. Протокол работы классифицирующей ЭС

```
:-start.
*****************************
*     Экспертная система    *
*   Игра "Отгадай животное  *
*                           *
*-------------------------- *
*   Отвечайте на вопросы:   *
*    да, нет, почему        *
*   Для объяснения решения  *
*     введите цель          *
*****************************
Введите любой символ
имеет(шерсть)?
:-да.
ест_мясо?
:-да.
желто-коричневое?
:-почему.
пытаюсь доказать гепард
с помощью правила: правило9
желто-коричневое?
:-да.
имеет(темные_пятна)?
:-да.
решение:гепард
объяснить ? [цель/нет]:
:-гепард.
гепард было доказано с помощью
правило9:
если [хищник,желто-коричневое,имеет(темные_пятна)]
то гепард
объяснить ? [цель/нет]:
:-хищник.
хищник было доказано с помощью
правило5:
если [млекопитающее,ест_мясо]
то хищник
объяснить ? [цель/нет]:
:-млекопитающее.
млекопитающее было доказано с помощью
правило1:
если [имеет(шерсть)]
то млекопитающее
объяснить ? [цель/нет]:
:-ест_мясо.
ест_мясобыло введено
объяснить ? [цель/нет]:
:-нет.
Искать ещё решение [да/нет] ?:
:-да.
кормит_детенышей(молоком)?
:-да.
решение:гепард
объяснить ? [цель/нет]:
:-нет.
Искать ещё решение [да/нет] ?:
:-да.
имеет(острые_зубы)?
:-да.
имеет(острые_когти)?
:-да.
имеет(прямо_посаженные_глаза)?
:-да.
решение:гепард
объяснить ? [цель/нет]:
:-нет.
Искать ещё решение [да/нет] ?:
:-да.
```

```
решение:гепард
объяснить ? [цель/нет]:
:-гепард.
гепард было доказано с помощью
правило9:
если [хищник,желто-коричневое,имеет(темные_пятна)]
то гепард
объяснить ? [цель/нет]:
:-хищник.
хищник было доказано с помощью
правило6:
если [млекопитающее,имеет(острые_зубы), имеет(острые_когти),
      имеет(прямо_посаженные_глаза)]
то хищник
объяснить ? [цель/нет]:
:-нет.
Искать ещё решение [да/нет] ?:
:-нет.
true
```

Заказ № _____ от « _____ » _____202 г. Тираж _____экз.
Изд-во СевГУ