МИНИСТЕРСТВО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
“МАТИ” — РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. К.Э. ЦИОЛКОВСКОГО

Кафедра “Моделирование систем и информационные технологии”

РАЗРАБОТКА И ИСПОЛЬЗОВАНИЕ ЭКСПЕРТНОЙ ПРОГРАММЫ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ ПРОЛОГ ДЛЯ РАСЧЕТА СОПРОТИВЛЕНИЯ ЭЛЕКТРИЧЕСКОЙ ЦЕПИ

Методические указания к лабораторной работе по предмету
“Методы искусственного интеллекта, базы знаний, экспертные системы”

Составитель В.В. Лидовский

Москва 2001

Владимир Викторович Лидовский

РАЗРАБОТКА И ИСПОЛЬЗОВАНИЕ ЭКСПЕРТНОЙ ПРОГРАММЫ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ ПРОЛОГ ДЛЯ РАСЧЕТА СОПРОТИВЛЕНИЯ ЭЛЕКТРИЧЕСКОЙ ЦЕПИ

Методические указания к лабораторной работе по предмету
“Методы искусственного интеллекта, базы знаний, экспертные системы”

Редактор М.А. Соколова

Оригинал-макет подготовлен в пакетах Plain-TEX и XY-pic

Под. в печ. 20.1.2001 Объем 1 п.л. Тираж 30 экз. Зак. 1

Ротапринт “МАТИ”—РГТУ, Берниковская наб. 14

ВВЕДЕНИЕ

Настоящие методические указания предназначены для обеспечения учебного процесса студентов четвертого курса дневной формы обучения специальности 220200 “АСОИиУ” при выполнении лабораторной работы по предмету “Методы искусственного интеллекта, базы знаний, экспертные системы”.

Цели лабораторной работы: изучить возможности средств языка программирования Пролог для создания экспертных систем и обработки реляционных данных, в частности, описывающих электрические схемы.

1. СИСТЕМНЫЕ ТРЕБОВАНИЯ

Для выполнения лабораторной работы необходимы следующие системные компоненты:

а) компьютер, работающий под управлением операционной системы Linux;

б) транслятор Пролога, соответствующий стандарту ISO/IEC 13211-1, например, GNU Prolog.

2. ПОДГОТОВКА ИСХОДНЫХ ДАННЫХ

Данные, описывающие электрическую схему, естественно представить в виде таблицы. Например, электрическую схему на рис. 1

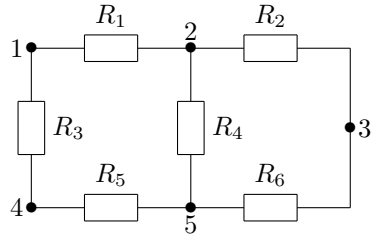


Рис. 1

естественно представить в виде следующей таблицы:

1-й узел	2-й узел	сопротивление
1	2	R_1
2	3	R_2
1	4	R_3
2	5	R_4
4	5	R_5
5	3	R_6

.

Очевидно, что подобной таблицей можно описать любую, сколь угодно сложную схему.

3. ОБРАБОТКА ТАБЛИЦ СРЕДСТВАМИ ЯЗЫКА ПРОГРАММИРОВАНИЯ ПРОЛОГ

Данные, представимые в виде таблиц, представляют собой частный случай более общих структур, называемых фразами Хорна. Язык программирования Пролог позволяет по заданной совокупности фраз Хорна (программе или, точнее, базе данных) автоматически получать ответы на заданный запрос, описываемый также фразой Хорна. Язык фраз Хорна является частью языка логики предикатов 1-го порядка. Предикатами называют функции, которые могут возвращать только логические значения. У предикатов 1-го порядка не может быть аргументов-предикатов. Вследствие того, что эффективная машинная обработка даже такой небольшой части языка логики, как язык фраз Хорна, чисто логическими средствами невозможна, Пролог использует другие (процедурные) методы поиска логических выводов. Выводы, получаемые Прологом, всегда являются частью всех возможных логически правильных ответов на заданный запрос. Из-за того, что поиск решений основан на последовательном вызове подпрограмм, соответствующих предикатам базы данных и запроса, а не на логическом выводе, Пролог имеет ряд средств для контроля за ходом поиска решений, которые не имеют к логике никакого отношения.

Фразы Хорна, составляющие базу данных, делятся на два типа — факты и правила. Первые представляют собой отдельные предикаты-утверждения, а вторые соответствуют структуре предложения “если–то”, часть “если” которого состоит из одного или нескольких, соединенных союзом “и” предикатов-условий, а часть “то” из одного предиката-заключения. Факты можно рассматривать как частный случай правил, состоящих только из части “то”. Операция логическое “и” в Прологе обозначается запятой, а операция логического следования — последовательностью двух символов, двоеточия и дефиса. Следование в Прологе записывается справа-налево, т. е. сначала заключение, а затем посылка. Если одному заключению соответствуют разные посылки, то это означает наличие между этими посылками отношения, соответствующего операции логическое “или”, которая обозначается символом “точка с запятой”. Фраза-запрос состоит из одного или нескольких разделенных запятыми (связками “и”) предикатов. Часть “то” фактов и правил называется головой фразы, а часть “если” — телом. Факты не имеют тела, а запросы головы. Все фразы завершаются точкой.

Весь запрос называется целью, а предикаты, его составляющие, —

подцелями. Все переменные, входящие в предикаты цели, называются целевыми. При поиске решения ключевую роль играет поиск сопоставимых предикатов и их дальнейшая унификация.

Два предиката сопоставимы, если они имеют одинаковые имена и местности и, кроме того, ни одна пара их соответствующих аргументов не состоит из разных констант. Унификация определяется подстановкой, делающей оба предиката идентичными. В ней паре соответствующих различных аргументов соответствует замена либо переменной на константу, либо переменной на переменную.

Поиск решения Прологом проходит согласно следующей последовательности шагов:

1) Если цель пуста, то получено одно решение, которое либо состоит из значений целевых переменных, либо, если переменных в запросе нет, — из ответа "да". Значения целевых переменных получаются последовательным применением к ним унифицирующих подстановок;
2) Вычисляется первая слева подцель в запросе. Она может состоять из предопределенного предиката или из предиката, заданного в базе данных. В первом случае, если вычисленное значение предиката истинно, то происходит его отбрасывание, иначе поиск решения для заданной цели прекращается. Во втором случае, база данных просматривается последовательно от начала до конца (порядок фраз в базе данных имеет значение) до тех пор, пока голова очередной фразы не станет сопоставимой с подцелью. Если найденная фраза содержит переменные, встречавшиеся в целях, то перед унификацией их надо заменить на другие, в целях не встречавшиеся. После чего вычисляется унифицирующая подстановка, подцель заменяется на тело найденной фразы и к полученной новой цели применяется вычисленная унифицирующая подстановка. Переход к шагу 1. Если же не найдется ни одной фразы с головой, сопоставимой с подцелью, то поиск решения для заданной цели прекращается.

Состояние, при котором дальнейший поиск решения для заданной цели невозможен, называется "неуспехом". Таким образом, поиск решения состоит в замене цели до тех пор, пока она либо не станет пустой ("успех"), либо невозможной ("неуспех"). Если в запросе нет переменных, то поиск заканчивается после нахождения одного решения. Если же переменные в запросе есть, то в случае как "успеха", так и "неуспеха" происходит возврат назад по цепочке целей до тех пор, пока очередная предыдущая цель не будет допускать альтернативы при замене своей первой подцели, т.е. если за найденной при поиске решения

фразой в базе данных есть еще фраза, голова которой сопоставима с заданной подцелью. Для найденной цели-альтернативы поиск решения также проходит согласно приведенному двухшаговому алгоритму. Если же все альтернативы исчерпаны, то если ни разу не был достигнут "успех", то результат ответ "нет", иначе ответом являются один или более наборов значений целевых переменных. Описанный метод поиска решений называется "поиск с возвратом".

Если установить резисторам на рис. 1 конкретные значения, например, $R_1 = 100$ Ом, $R_2 = R_4 = 300$ Ом, $R_3 = R_5 = R_6 = 500$ Ом, то таблицу, описывающую эту схему, можно представить следующим далее набором фактов базы данных Пролога.

```
connection(100, 1, 2).
connection(300, 2, 3).
connection(500, 1, 4).
connection(300, 2, 5).
connection(500, 4, 5).
connection(500, 5, 3).
```

Эта база данных, как и эквивалентная ей таблица, имеет два существенных недостатка. Во-первых, она фиксирует направление от контакта к контакту в каждом соединении. Информация о том, что сопротивление резистора не зависит от выбранного направления между контактами, в базе данных никак не отражена. Во-вторых, на схеме могут быть несколько идентичных соединений, которые было бы желательно четко различать. С другой стороны подобная база данных наиболее естественна для ввода в качестве исходной. Таким образом, получается, что перед собственно расчетами нужно пополнить базу данных таблицей, которая будет как содержать всю исходную информацию, так и информацию, избавляющую от двух означенных недостатков. В этой новой таблице одному соединению будут соответствовать две строки, задающих оба возможных направления при соединении, с идентичными полями сопротивления и дополнительного идентифицирующего номера соединения. Эта новая таблица должна создаваться из исходной автоматически в начале расчетов.

Расчет сопротивления между двумя заданными контактами на схеме будет проходить через серию преобразований базы данных, цель которых свести всю схему к эквивалентному ей по сопротивлению одному соединению. Означенные преобразования базы данных соответствуют простейшим электротехническим эквивалентным расчетным заменам:

1) два последовательно соединенных резистора, заменяются одним с сопротивлением, равным сумме сопротивлений исходных резисторов;

2) два параллельно соединенных резистора, заменяются одним с величиной обратной сопротивлению, равной сумме величин, обратных сопротивлениям исходных резисторов;
3) изолированная ветвь или петля (примеры таких ветви и петли, возникающих при измерении сопротивления между контактами 1 и 3, показаны соответственно на рис. 2 и рис. 3) отбрасывается;

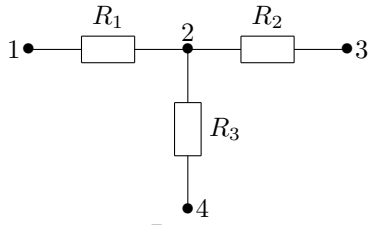


Рис. 2

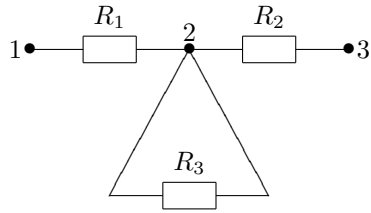
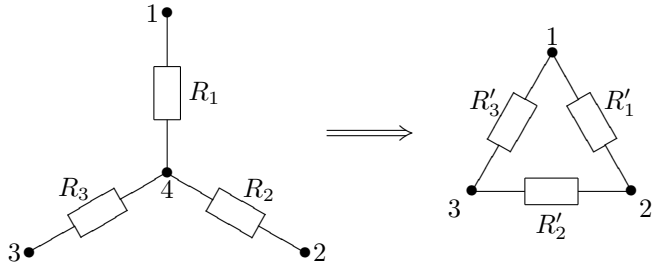


Рис. 3

4) трехлучевая звезда резисторов, заменяется на треугольник согласно рис. 4.



$$R'_1 = \frac{R_1 * R_2 + R_1 * R_3 + R_2 * R_3}{R_3}$$

$$R'_2 = \frac{R_1 * R_2 + R_1 * R_3 + R_2 * R_3}{R_1}$$

$$R'_3 = \frac{R_1 * R_2 + R_1 * R_3 + R_2 * R_3}{R_2}$$

Рис. 4

Первые два преобразования сокращают базу данных на одно соединение каждое. Четвертое исключает из базы данных один контакт. Третье сокращает базу данных на одно соединение, а в случае с изолированной ветвью еще и исключает один контакт.

Сначала производятся все возможные простейшие преобразования 1, 2 и 3. Если в результате не удастся получить одно соединение, то производится 4-е преобразование и затем все возможные простейшие

упрощения 1–3. Если в результате последних получить одно соединение опять не удастся, то цикл повторяется с 4-го преобразования. Если же когда-нибудь после 4-го преобразования не удастся выполнить ни одного упрощения 1–3, то это будет означать неспособность данного метода поиска решения найти ответ на запрос.

4. ФОРМАЛЬНАЯ ЛОГИКА ПОИСКА РЕШЕНИЙ И ВСПОМОГАТЕЛЬНЫЕ ОПИСАНИЯ

Пролог не позволяет изменять фразы, заданные в тексте программы. Данные из текста программы называются поэтому статическими, в отличие от динамических данных, которые могут произвольно меняться по ходу поиска решений. Пример статических данных — это предикат connection, пример динамических — это создаваемая по информации из него новая таблица, описывающая предикат branch.

Все динамические предикаты должны быть объявлены до их первого использования стандартным предикатом dynamic. Кроме новой таблицы, branch, полностью описывающей схему, нужны еще будут таблица из двух строк и одного столбца, extra_point, для хранения двух точек измерения, а также одностолбцовая таблица, t, для временных величин.

```
:- dynamic(branch/4, extra_point/1, t/1).
```

В приведенном объявлении после имени предиката через наклонную черту следует его местность. Это объявление оформлено как запрос. Он выполняется до ввода запроса пользователем.

Затем в текст программы заносится описание схемы для расчета предикатом connection.

Затем следует описание метода поиска решения на языке фраз Хорна. Поиск решения инициируется запросом, 3-местным предикатом query, 1-й аргумент которого целевая переменная, искомое сопротивление, а два оставшихся — номера точек измерения.

После получения запроса нужно уничтожить все оставшиеся от возможных предыдущих расчетов динамические данные и затем правильно заполнить динамические таблицы branch и extra_point.

```
query(R,X,Y) :-
  retractall(extra_point(_)),
    assertz(extra_point(X)), assertz(extra_point(Y)),
    fill_branch;              % инициализация динамических данных
  simplify, branch(R,X,Y,_), !;                          % расчет
  write('I can''t get solution...'), nl.
fill_branch :-                                % заполнение branch
```

```
  retractall(branch(_,_,_,_)), retractall(t(_)),
    assertz(t(0)), fail;
  connection(R,X,Y), t(N), N1 is N+1, retract(t(N)),
    assertz(t(N1)), assertz(branch(R,X,Y,N)),
    assertz(branch(R,Y,X,N)), fail.
```

Всегда истинный предикат assertz добавляет утверждение-аргумент к концу динамической базы данных, а всегда истинный предикат retractall удаляет из динамической базы данных все фразы, голова которых сопоставима с его аргументом. Предикат retract отличается от retractall тем, что он за один раз удаляет только одну фразу и создает точку разветвления для поиска решений. Если retract не сможет удалить ни одной фразы, то его результат — ложь. Переменные в Прологе должны начинаться с заглавной буквы или с подчеркивания. Переменная, имя которой состоит из одного знака подчеркивания, называется анонимной. Все анонимные переменные считаются различными и их значения не выводятся в ответах. Всегда истинные предикаты write и nl печатают соответственно значение своего аргумента и конец строки. "Предикат" fail всегда ложен — это не логическое средство, для описания логики он бессмыслен. Однако, в Прологе, поставленный в конце фразы, он прекратит ее вычисление только после вычисления всех предшествующих ему предикатов. Это и связка "или" позволяют организовывать вспомогательные расчеты. Всегда истинный "предикат" отсечения, обозначаемый восклицательным знаком, также не является логическим средством — он используется из-за своего побочного эффекта. После выполнения отсечения поиск решения прекращается по целям-альтернативам, возникающим по ходу выполнения предиката, вызвавшего предикат с отсечением. В частности, выполнение отсечения в предикате означает отказ от вычисления следующих далее через связку "или" его частей.

Предикат fill_branch всегда ложен — после него через связку "или" следует главная часть программы — попытка упрощения схемы до одного эквивалентного соединения. Предикат simplify истинен, если такое упрощение удастся сделать и ложен в противном случае. Если упрощение удалось, то печатается сопротивление полученного соединения; если нет, то — сообщение о том, что программа не может найти решение.

```
simplify :-
  simplify1;
  simplify2;
  calcnbr(2).
simplify1 :-
```

```
  retractall(t(1)), parallel;
  sequential;
  cut;
  t(1), simplify1.
simplify2 :-
  retractall(t(2)), star, simplify1;
  t(2), simplify2.
calcnbr(X) :-
  retractall(t(_)), assertz(t(0)), fail;
  branch(_,_,_,_), t(N), N1 is N+1,
    retract(t(N)), assertz(t(N1)), fail;
  t(N), X = N.
```

Предикат simplify1 пытается упростить схему всеми возможными простейшими преобразованиями 1–3, осуществляемые предикатами sequential, parallel и cut, которые, кроме того, устанавливают истинными t(1) и t(2) при осуществлении преобразований. Предикат simplify2 перед тем как перейти к простейшим преобразованиям выполняет преобразование 4, предикатом star. И simplify1, и simplify2 — всегда ложны, каждый из них выполняется до тех пор, пока ни одно из простейших преобразований не сможет быть выполнено. После их выполнения производится расчет количества соединений, оставшихся после преобразований схемы, предикатом calcnbr. Если это число равно 2, то simplify устанавливается истинным; если нет, то — ложным.

```
sequential :-
  branch(R1,X,Y,N1), branch(R2,Y,Z,N2), N1 \== N2, simple2(Y),
  retract(branch(R1,X,Y,N1)), retract(branch(R2,Y,Z,N2)),
  retract(branch(R1,Y,X,N1)), retract(branch(R2,Z,Y,N2)),
  R is R1+R2,
  assertz(branch(R,X,Z,N1)), assertz(branch(R,Z,X,N1)),
  assertz(t(1)), assertz(t(2)), fail.
simple2(X) :-
  branch(_,X,_,N1), branch(_,X,_,N2), N1 \== N2,
    branch(_,X,_,N3), N1 \== N3, N2 \== N3, !, fail;
  extra_point(X), !, fail;
  true.
parallel :-
  branch(R1,X,Y,N1), branch(R2,X,Y,N2), N1 \== N2,
  retract(branch(R1,X,Y,N1)), retract(branch(R2,X,Y,N2)),
  retract(branch(R1,Y,X,N1)), retract(branch(R2,Y,X,N2)),
  R is R1*R2/(R1+R2),
  assertz(branch(R,X,Y,N1)), assertz(branch(R,Y,X,N1)),
```

```
  assertz(t(1)), assertz(t(2)), fail.
cut :-
  branch(_,X,X,_), retractall(branch(_,X,X,_)),
    assertz(t(1)), assertz(t(2)), !, fail;       % удаление петель
  branch(_,X,_,_), simple1(X),
    retractall(branch(_,X,_,_)), retractall(branch(_,_,X,_)),
    assertz(t(1)), assertz(t(2)),
    fail.                        % удаление изолированных соединений
simple1(X) :-
  branch(_,X,_,N1), branch(_,X,_,N2), N1 \== N2, !, fail;
  extra_point(X), !, fail;
  true.
star :-
  branch(R1,X1,Y,N1), branch(R2,X2,Y,N2), N1 \== N2,
  branch(R3,X3,Y,N3), N1 \== N3, N2 \== N3, simple3(Y),
  retract(branch(R1,X1,Y,N1)), retract(branch(R1,Y,X1,N1)),
  retract(branch(R2,Y,X2,N2)), retract(branch(R2,X2,Y,N2)),
  retract(branch(R3,X3,Y,N3)), retract(branch(R3,Y,X3,N3)),
  RT is R1*R2+R2*R3+R1*R3,
  R12 is RT/R3, R23 is RT/R1, R13 is RT/R2,
  assertz(branch(R12,X1,X2,N1)), assertz(branch(R12,X2,X1,N1)),
  assertz(branch(R13,X1,X3,N2)), assertz(branch(R13,X3,X1,N2)),
  assertz(branch(R23,X2,X3,N3)), assertz(branch(R23,X3,X2,N3)).
simple3(X) :-
  branch(_,X,_,N1), branch(_,X,_,N2), N1 \== N2,
    branch(_,X,_,N3), N1 \== N3, N2 \== N3,
    branch(_,X,_,N4), N1 \== N4, N2 \== N4, N3 \== N4, !, fail;
  extra_point(X), !, fail;
  true.
```

Предикаты simple1(X), simple2(X) и simple3(X) истины в случае, если X — это узел на схеме, в котором пересекаются не более соответственно одного, двух и трех соединений.

5. ПОРЯДОК ВЫПОЛНЕНИЯ И ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

5.1. Порядок выполнения лабораторной работы

Лабораторная работа выполняется за следующие четыре этапа:

1) внесение в базу знаний описания логики поиска решения;
2) дополнение базы знаний фактами, описывающими заданную индивидуальным вариантом схему;

3) в Пролог-системе ввести запросы к введенной на предыдущих этапах базе знаний для получения ответов о сопротивлении заданных вариантом задания участков;
4) проанализировать ход выполнения Пролог-программы по каждому из запросов.

Ход преобразований при выполнении запроса можно отслеживать добавлением к определениям упрощающих предикатов cut, parallel, sequential и star части, печатающей соответствующую информацию при выполнении преобразования. Например, в описание предиката sequential, перед fail можно поставить

```
write('sequential('), write(X), write(Y), write(Z), write(')').
```

5.2. **Отчет по лабораторной работе**

Отчет по работе выполняется на отдельных листах или в отдельной тетради и должен содержать:
1) цель исследования;
2) спецификацию задания;
3) полученные ответы и описание хода их получения Пролог-системой;
4) выводы.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назначение языка программирования Пролог, его сильные и слабые стороны.
2. Связь логических и табличных данных.
3. Недостатки и преимущества использования Пролога при расчетах характеристик сложных электрических цепей по сравнению с методом, опирающемся на построение системы линейных уравнения по законам Кирхгофа.
4. Пути совершенствования приведенной в настоящем указании экспертной программы.
5. Логическое описание реализации методов упрощения электрической схемы, описанных в базе знаний.
6. Методы формального описания электрических схем, их достоинства и недостатки.
7. Оптимальной ли была стратегия поиска решения при поиске ответов на запросы?

7. ВАРИАНТЫ РАБОТ

Лабораторная работа имеет 25 вариантов заданий на основе пяти принципиальных электрических схем, приведенных на рис. 5–9. Каж-

дый индивидуальный вариант задается номером рисунка и двумя парами номеров контактов с этих рисунков, сопротивления между которыми нужно рассчитать. Для всех вариантов $R_1 = R_3 = 100$ Ом, $R_2 = R_7 = R_8 = 200$ Ом, $R_4 = R_5 = R_6 = 300$ Ом, $R_9 = R_{12} = R_{14} = 400$ Ом, $R_{10} = R_{11} = 700$ Ом, $R_{13} = 600$ Ом.

Вариант	*№ рисунка*	*1-я пара*	*2-я пара*
1	5	3, 8	4, 5
2	5	4, 6	5, 7
3	5	1, 7	1, 6
4	5	9, 10	1, 9
5	5	2, 6	4, 8
6	6	3, 4	3, 6
7	6	1, 2	4, 5
8	6	1, 8	1, 6
9	6	3, 7	1, 7
10	6	2, 6	2, 4
11	7	1, 7	1, 6
12	7	1, 2	1, 5
13	7	5, 6	3, 9
14	7	3, 6	3, 7
15	7	4, 5	8, 9
16	8	1, 2	1, 5
17	8	1, 6	7, 8
18	8	2, 4	2, 7
19	8	3, 9	1, 3
20	8	3, 4	3, 5
21	9	1, 6	1, 7
22	9	1, 5	6, 11
23	9	2, 8	2, 9
24	9	5, 11	1, 8
25	9	4, 6	4, 5

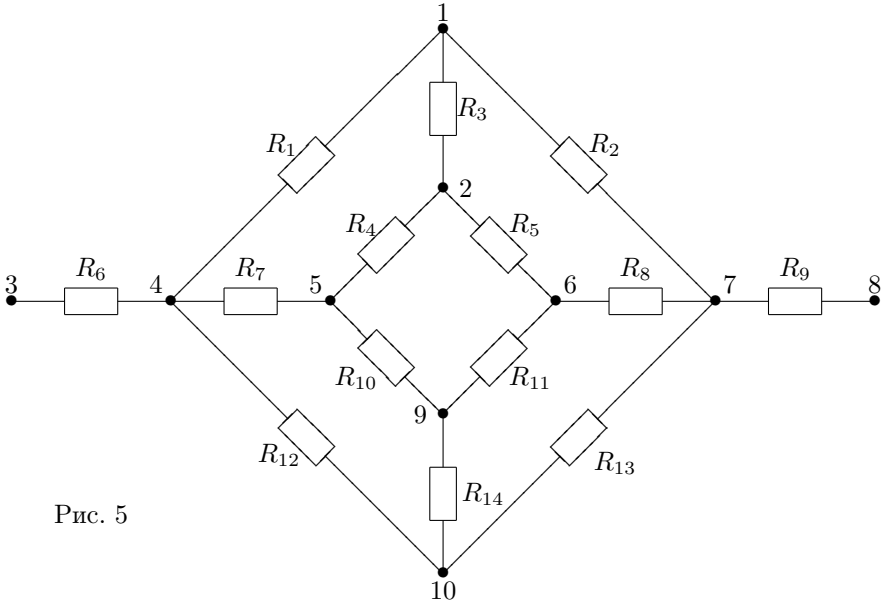


Рис. 5

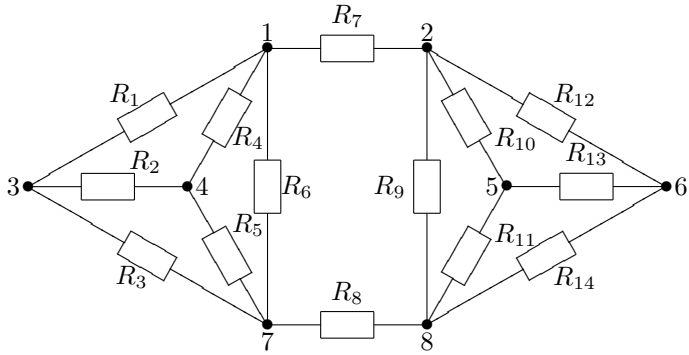


Рис. 6

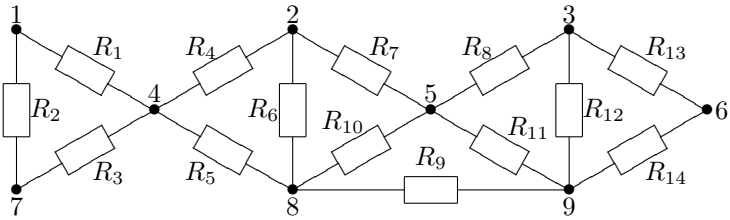


Рис. 7

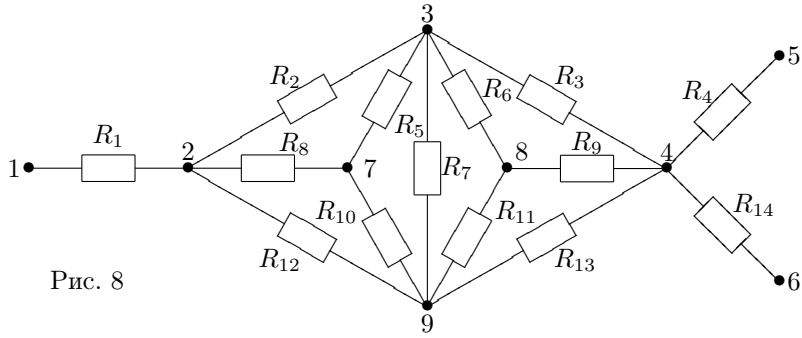


Рис. 8

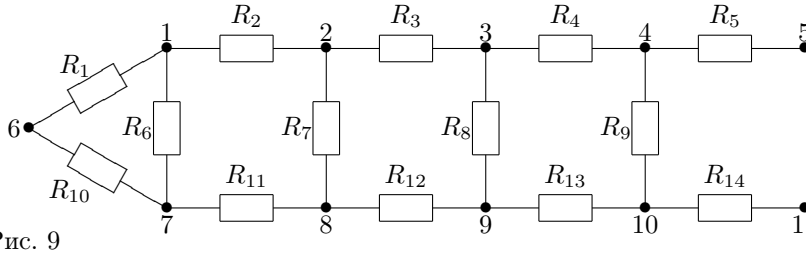


Рис. 9

ЛИТЕРАТУРА

1. Братко И. *Программирование на языке ПРОЛОГ для искусственного интеллекта* — М.: Мир, 1990.
2. Волынский Б.А., Зейн Е.Н., Шатерников В.Е. *Электротехника* — М.: Энергоатомиздат, 1987.
3. Лорьер Жан-Луис *Системы искусственного интеллекта* — М.: Мир, 1991.
4. Тей А., Грибомон П., Луи Ж. и др. *Логический подход к искусственному интеллекту* — М.: Мир, 1990.

ОГЛАВЛЕНИЕ