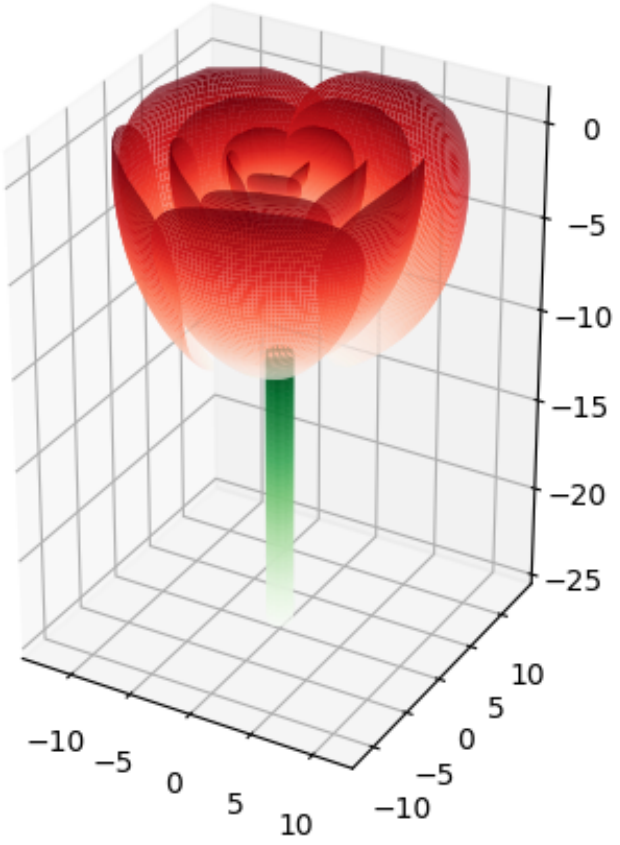
Г. В. Гренкин

ОСНОВЫ КОМПЬЮТЕРНОЙ ЛОГИКИ



$$\forall x\, \exists y\, P(x,y)$$

```
all([any([P[i][j] for j in J]) for i in I])
```

Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Владивостокский государственный университет»

Г. В. Гренкин

ОСНОВЫ КОМПЬЮТЕРНОЙ ЛОГИКИ

Электронное учебное пособие

Владивосток
Издательство ВВГУ
2024

УДК 004.021
ББК 22.12
Г80

Рецензенты: *А.А. Степанова*, д-р физ.-мат. наук, профессор департамента математики ДВФУ;
В.М. Гриняк, д-р техн. наук, доцент, профессор каф. ИТС ВВГУ.

Гренкин, Глеб Владимирович

Г80 **Основы компьютерной логики :** учебное пособие / Г.В. Гренкин ; Владивостокский государственный университет; Электрон. текст. дан. (1 файл: 1,29 Мб). — Владивосток : Изд-во ВВГУ, 2024. — 1 электрон. опт. диск (CD-ROM). — Систем. требования: Intel Pentium (или аналогичный процессор других производителей), 500 МГц; 512 Мб оперативной памяти; видеокарта SVGA, 1280×1024 High Color (32 bit); 5 Мб свободного дискового пространства; операц. система Windows XP и выше; Acrobat Reader, Foxit Reader либо другой их аналог.

ISBN 978-5-9736-0739-5

В пособии разбираются логические конструкции, помогающие делать умозаключения об информационных объектах. Рассматриваются разделы: булева алгебра, исчисление высказываний, логика предикатов, модели вычислений, алгоритмы. По каждому разделу приводится лекционный материал и практикум, включающий разбор типовых примеров, задачи и лабораторные работы.

Для студентов, обучающихся по направлениям подготовки в области информатики и вычислительной техники.

Электронное учебное издание

Минимальные системные требования:

Компьютер: Pentium 3 и выше, 500 МГц; 15,6 Мб; 5 Мб на жестком диске; видеокарта SVGA, 1280×1024 High Color (32 bit); привод CD-ROM. Операционная система: Windows 10.

Программное обеспечение: Internet Explorer 8 и выше или другой браузер; Acrobat Reader, Foxit Reader либо любой другой их аналог.

ISBN 978-5-9736-0739-5

Редактор И.Г. Шабунина
Компьютерная верстка: Г.В. Гренкин

690014, г. Владивосток, ул. Гоголя, 41
Тел./факс: (423)240-40-54

Объем 1,29 Мб. Усл.-печ. л. 12,1.
Подписано к использованию 2024 г.
Тираж 200 [50+1] экз.

Оглавление

Введение

Язык компьютера — это язык вычислений. На этом языке программист пишет инструкции, результатом выполнения которых является преобразование входной информации в выходную. Программа, применяемая к разным входам, должна всякий раз выдавать такой выход, который является искомым ответом на вопрос, заданный по отношению ко входу. Только программа, которая выдает требуемый выход для всех корректных входов, будет считаться верной. Достичь этого непросто, и главным инструментом программиста могут послужить логические утверждения, относящиеся к вычислениям.

Чтобы проиллюстрировать, насколько важна логика для эффективных вычислений, приведем пример из математического анализа. Одно и то же доказательство утверждения, что сумма двух непрерывных функций — непрерывная функция, можно записать как словами: *известно, что функции f и g непрерывны в точке x_0. Это означает, что предел функции f в точке x_0 равен $f(x_0)$ и предел функции g в точке x_0 равен $g(x_0)$. Пользуясь свойством пределов функций в точке, получаем, что предел суммы функций f и g в точке x_0 равен сумме пределов этих функций в точке x_0, то есть $f(x) + g(x) \to f(x_0) + g(x_0)$ при $x \to x_0$. А значит, функция $f + g$ непрерывна в точке x_0,* так и формулами, почти как программным кодом:

$$
\begin{gathered}
(f \text{ непрерывна в т. } x_0)\ \&\ (g \text{ непрерывна в т. } x_0) \Rightarrow \\
\Rightarrow \left(\lim_{x\to x_0} f(x) = f(x_0)\right)\ \&\ \left(\lim_{x\to x_0} g(x) = g(x_0)\right) \Rightarrow \\
\Rightarrow \lim_{x\to x_0} (f(x) + g(x)) = \lim_{x\to x_0} f(x) + \lim_{x\to x_0} g(x) = f(x_0) + g(x_0) \Rightarrow \\
\Rightarrow (f + g) \text{ непрерывна в т. } x_0,
\end{gathered}
$$

причем второй способ превращает рассуждения в вычисления.

Перевод идей из общего описания в вычислительную логику и обратно — это один из навыков, который необходим программисту. Этот навык хорошо тренируется математическими упражнениями. Компьютерная логика нужна для того, чтобы сопроводить программы логическими комментариями для объяснения назначения вычислений.

Глава 1

Булева алгебра

Тогда и только тогда...
В логике важно следить
За своими высказываниями.

1.1 Электронные схемы

Представление информации в компьютере, как хорошо известно, организуется в виде набора двоичных состояний. Если на контакт подается высокое напряжение, то это состояние 1, а если низкое, то 0. Двоичная логика выбрана из соображений надежности: так состояния легче различимы. Если же потребуется представить в компьютере одно из многих значений, это можно сделать с помощью комбинации из нескольких двоичных состояний.

Имея на входе набор двоичных входов, можно поставить задачу рассчитать набор двоичных выходов, который связан со входом по некоторому правилу. Каким бы сложным ни было это правило, всё вычисление можно свести к комбинации функций логического умножения AND, логического сложения OR и инверсии NOT.

Любую связь между входом и выходом, таким образом, можно записать в виде логической формулы, содержащей: конъюнкцию &, дизъюнкцию $\vee$, отрицание $\neg$. По этой формуле всегда можно сконструировать логическую схему, состоящую из вентилей трех упомянутых типов. В электронных схемах эти вентили реализуются с помощью транзисторов. Вентили лежат в основе аппаратного обеспечения, на котором строятся все цифровые компьютеры. Вся современная цифровая логика основывается на том, что транзистор может работать как очень быстрый двоичный переключатель[1].

1.2 Алгебра высказываний

1.2.1 Понятие высказывания

Логическим высказыванием называют утверждение, принимающее одно из двух значений: истина (1) или ложь (0). Например, “Владивосток является портом”. Примеры не высказываний: “в этом семестре у студентов нашей группы не будет задолженностей” (истинностное значение пока неизвестно), “это высказывание ложно” (истинностное значение не определено).

Придадим истинностное значение первого из названных высказываний логической переменной A. Точно так же введем еще две логические переменные: $B =$ “Уссурийск является портом” и $C =$ “Находка является портом”. Итак, $A = 1$, $B = 0$, $C = 1$.

Если бы мы выбрали другие города Приморского края, то логические переменные A, B, C, возможно, приняли бы другие значения. Допустим, нам известно, что если пер-

[1] Таненбаум Э., Остин Т. Архитектура компьютера. — Санкт-Петербург: Питер, 2013.

вый город является портом, то второй также порт. Запишем это утверждение в виде *составного высказывания*: $A \to C$. Если первый город — не порт, то высказывание $A \to C$ считается истинным, поскольку в рамках этого высказывания ложность A не накладывает ограничений на истинность C.

Утверждение о том, что хотя бы один из первых двух городов — порт, запишем в виде логического ИЛИ следующим образом: $A \vee B$. Утверждение о том, что A, B, C принимают конкретные истинностные значения, запишем в виде формулы $A \,\&\, \neg B \,\&\, C$, содержащей логическое И и логическое НЕ.

Итак, элементарные высказывания обозначаются латинскими буквами (возможно, с индексами). Составные высказывания выражаются логическими формулами, состоящими из элементарных высказываний, связанных между собой *логическими операциями* (табл. 1.1). Подставляя на место переменных конкретные значения, получаем выражение, вычисляемое согласно определениям логических операций (табл. 1.2). При записи формулы лишние скобки можно опустить, если считать, что операции имеют такой *приоритет*: $\neg$, $\&$, $\vee$, $\to$, $\leftrightarrow$.

Таблица 1.1

Логические операции

Операция	Знак	Равна 1	Равна 0
НЕ	$\neg A$	$A = 0$	$A = 1$
И	$A \,\&\, B$	$A = 1$ и $B = 1$	$A = 0$ или $B = 0$
ИЛИ	$A \vee B$	$A = 1$ или $B = 1$	$A = 0$ и $B = 0$
ЕСЛИ	$A \to B$	$B = 1$ или $A = 0$	$A = 1$ и $B = 0$
РАВНО	$A \leftrightarrow B$	$A = B$	$A \neq B$

Таблица 1.2

Значения логических операций

A	$\neg A$
0	1
1	0

A	B	$A \,\&\, B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

A	B	$A \to B$
0	0	1
0	1	1
1	0	0
1	1	1

Таблица истинности составного высказывания показывает его истинностные значения при всевозможных значениях входящих в него переменных.

К примеру, формализуем фразу из песни: “Если ты со мной, мир меняет цвет и других возможных вариантов нет”. Для этого введем элементарные высказывания: A = “ты со мной”, B = “мир меняет цвет”, C = “других вариантов нет”. Запишем формулу, правильно расставив скобки: $(A \to B) \,\&\, C$. Чтобы построить таблицу истинности, перечислим все *подформулы* и затем аккуратно вычислим их значения для каждого набора значений переменных (табл. 1.3). Наборы принято перечислять в лексикографическом порядке.

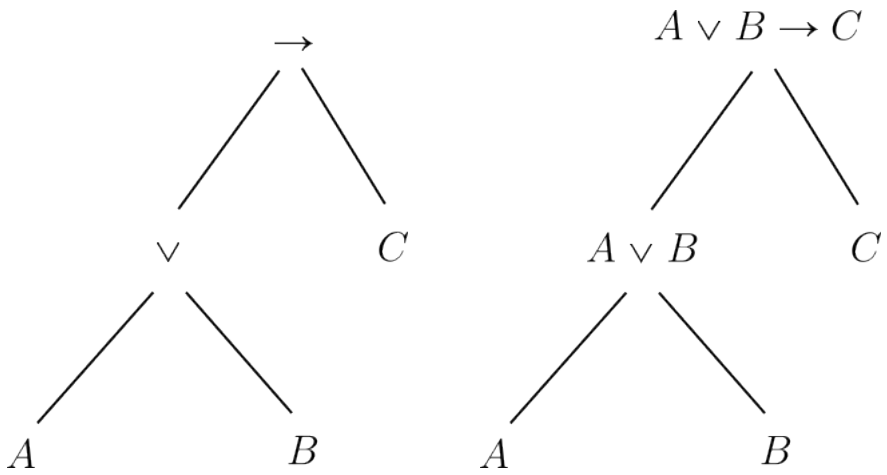
Формализуем высказывание “Собаки и кошки имеют хвост”. Введем элементарные высказывания: A = “это собака”, B = “это кошка”, C = “есть хвост”. Если это кошка или собака, то есть хвост: $A \vee B \to C$.

Операцию, которая выполняется последней, назовем *главной*, или *корневой*. В нашей формуле корневая операция — это логическое ЕСЛИ $\to$. Эта операция разбивает формулу на две подформулы: $A \vee B$ и C. Структуру формулы можно изобразить с помощью дерева выражения (рис. 1.1а) или дерева подформул (рис. 1.1б).

Таблица 1.3

Таблица истинности

A	B	C	$A \to B$	$(A \to B)\,\&\,C$
0	0	0	1	0
0	0	1	1	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1



а б

Рис. 1.1. а — дерево выражения; б — дерево подформул

Формализуем фразу из песни: “Полковнику никто не пишет”, чтобы сделать ее понятной компьютеру. Высказывание “Полковнику никто не пишет” утверждает, что есть один человек a, обладающий свойством P (пишут $P(a) = 1$), причем все остальные люди не связаны с ним отношением «писать» (пишут $R(x, a) = 0$). Как в прошлом примере, составим две подформулы: $P(a)$ и $\neg R(x, a)$. Первая утверждает, что a — полковник, вторая говорит, что x не пишет a. Чтобы указать, откуда брать значения переменной x, поставим x под знаком *квантора общности*: $\forall x\, \neg R(x, a)$. Это высказывание читается “для всех x неверно, что $R(x, a)$” и принимает значение 1 тогда и только тогда, когда формула под знаком квантора $\neg R(x, a)$ равна 1 для всех возможных x. Остается связать две подформулы операцией конъюнкции:

$$P(a)\ \&\ \big(\forall x\, \neg R(x, a)\big). \tag{1.1}$$

Как вычисляется формула (1.1)? Корневая операция & выполняется над результатами вычисления подформул $P(a)$ и $\forall x\, \neg R(x, a)$. Формула $P(a)$ атомарная. В формуле $\neg R(x, a)$ корневая операция — логическое НЕ, которое применяется к атомарной формуле $R(x, a)$. Но поскольку эта формула стоит под знаком квантора, то в ней переменная x будет пробегать значения из множества всех возможных значений, которое называют *универсумом*. Итак, вычислив значение выражения $\neg R(x, a)$ для всех x из множества людей, узнаем, верно ли, что это выражение всегда принимает значение 1. Получается, что формула (1.1) принимает значение 1 только в одном случае: когда $P(a) = 1$ и $R(x, a) = 0$ для всех x.

Человек x	Он(а) полковник? $P(x)$
Аня	нет (0)
Боря	да (1)
Вова	нет (0)
Галя	да (1)

	А	Б	В	Г
А			V	
Б				V
В	V			
Г			V	

Рис. 1.2. Полковнику Боре никто не пишет: $R(x, \text{Б}) = 0$ для любого x

Чтобы вычислить $P(a)$, нужна таблица всех людей, в которой есть столбец с указанием того, полковник этот человек или нет (рис. 1.2). Чтобы вычислить $\forall x\, \neg R(x, a)$, понадобится квадратная таблица, в которой отмечено, кто кому пишет. Это будут исходные данные для вычислений.

1.2.2 Булевы функции

Пусть логические переменные A и B выражают состояние двух выключателей: вкл. (1) либо выкл. (0). Составим из элементарных высказываний A и B высказывание $F =$ "Свет выключен". Для этого определим булеву функцию $F(A, B)$, зависящую от переменных A, B.

Булевой функцией от n переменных $x_1, x_2, \ldots, x_n$ называют правило, по которому каждому набору значений переменных $x_1, x_2, \ldots, x_n$ ставится в соответствие значение 0 или 1.

Всего существует 4 различных набора значений переменных A и B:

1) оба выключателя выключены: $A = 0$, $B = 0$;
2) первый выключен, второй включен: $A = 0$, $B = 1$;
3) первый включен, второй выключен: $A = 1$, $B = 0$;
4) оба выключателя включены: $A = 1$, $B = 1$.

Если мы утверждаем, что свет выключен, то мы фиксируем одну возможную комбинацию: $A = 0$, $B = 0$. Если поставить рядом с каждой комбинацией 1 в случае, если она может осуществиться, или 0, если она осуществиться не может, то получим таблицу истинности (табл. 1.4).

Таблица 1.4

Таблица истинности высказывания "Свет выключен"

Выключатель A	Выключатель B	Свет выключен F
0	0	1
0	1	0
1	0	0
1	1	0

Ясно, что всякая таблица истинности высказывания определяет булеву функцию. Всего существует 2^n различных наборов значений n переменных. Столько же строк содержит

любая таблица истинности. Наборы значений переменных принято перечислять в таблице в лексикографическом порядке. Это означает, что сначала на первое место ставят 0, а затем перечисляют все наборы от $n-1$ переменных, после чего на первое место ставят 1 и повторяют процедуру.

Заметим, что булева функция двух переменных, выражающая составное высказывание "свет выключен", не равна ни одной из булевых функций, перечисленных в табл. 1.2. Однако нашу функцию можно выразить через основные логические операции.

В самом деле, утверждая, что "Свет выключен", мы подразумеваем следующее: первый и второй выключатели не включены. Разобьем это предложение на фрагменты:

1) первый выключатель не включен: $\neg A$;
2) второй выключатель не включен: $\neg B$;
3) первый и второй выключатели не включены: $\neg A \,\&\, \neg B$.

Таким образом, булева функция, выражающая составное высказывание "Свет выключен" в зависимости от значений простых высказываний A = "Первый выключатель включен" и B = "Второй выключатель включен", может быть вычислена по формуле $F(A,B) = \neg A \,\&\, \neg B$ (табл. 1.5).

Таблица 1.5

Вычисление булевой функции по таблице истинности

A	B	$\neg A$	$\neg B$	$\neg A \,\&\, \neg B$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Допустим, вы работаете в приемной комиссии вуза и вам дали задание написать скрипт "калькулятор абитуриента", который спрашивает, на каких образовательных программах человек учился, и выводит, на какие образовательные программы он может поступить на бюджетной основе.

Чтобы решить эту задачу, составим таблицу ответов для всех возможных входов. Для этого введем элементарные высказывания: $B(x)$ = "Человек x учился на бакалавриате", $M(x)$ = "Человек x учился в магистратуре", $A(x)$ = "Человек x учился в аспирантуре", $S(x)$ = "Человек x учился на специалитете", $P(x)$ = "Человек x учился на профессионалитете". Выходные данные выражают элементарные высказывания: $B'(x)$ = "Человек x может поступить на бакалавриат", $M'(x)$ = "Человек x может поступить в магистратуру", $A'(x)$ = "Человек x может поступить в аспирантуру", $S'(x)$ = "Человек x может поступить на специалитет", $P'(x)$ = "Человек x может поступить на профессионалитет".

Перечисленные булевы функции сведены в табл. 1.6. В пяти последних столбцах стоят прочерки, если в пяти первых столбцах указаны некорректные входные данные. Таким образом, булевы функции $B'(B,M,A,S,P)$, $M'(B,M,A,S,P)$, $A'(B,M,A,S,P)$, $S'(B,M,A,S,P)$, $P'(B,M,A,S,P)$ определены не всюду — говорят, что это *частичные функции*.

1.2.3 Формулы алгебры высказываний

Дадим формальное определение записи составного высказывания или формулы алгебры высказываний. Определение является индуктивным, то есть указывает правила построения новых формул из уже построенных:

Таблица 1.6

Таблица истинности

B	M	A	S	P	B'	M'	A'	S'	P'
0	0	0	0	0	1	0	0	1	1
0	0	0	0	1	1	1	1	1	0
0	0	0	1	0	0	1	1	0	0
0	0	0	1	1	0	1	1	0	0
0	0	1	0	0	–	–	–	–	–
0	0	1	0	1	–	–	–	–	–
0	0	1	1	0	0	1	0	0	0
0	0	1	1	1	0	1	0	0	0
0	1	0	0	0	–	–	–	–	–
0	1	0	0	1	–	–	–	–	–
0	1	0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	1	0	0
0	1	1	0	0	–	–	–	–	–
0	1	1	0	1	–	–	–	–	–
0	1	1	1	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0
1	0	0	0	1	0	1	0	0	0
1	0	0	1	0	0	1	1	0	0
1	0	0	1	1	0	1	1	0	0
1	0	1	0	0	–	–	–	–	–
1	0	1	0	1	–	–	–	–	–
1	0	1	1	0	0	0	0	0	0
1	0	1	1	1	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0
1	1	0	0	1	0	0	1	0	0
1	1	0	1	0	0	0	1	0	0
1	1	0	1	1	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0

1) если φ — логическая переменная, то φ — формула;
2) если φ и ψ — формулы, то $\neg\varphi$, $(\varphi \,\&\, \psi)$, $(\varphi \vee \psi)$, $(\varphi \to \psi)$ — формулы;
3) других формул нет.

Только что мы определили правила *синтаксического вывода*. Пункт 1 выражает базу индукции, то есть указывает стартовые формулы, из которых мы выводим все другие формулы. Пункт 2 выражает шаг индукции, который основан на гипотезе индукции, то есть допущении, что две строки φ и ψ уже выведены как формулы. Пункт 3 утверждает, что мы описали таким образом всё множество формул.

Итак, формула алгебры высказываний представляет собой упорядоченный набор логических переменных, логических операций и скобок. Согласно нашему определению результат применения любой бинарной операции берется в скобки, так что $A \,\&\, B \vee C$ — не формула, а $((A \,\&\, B) \vee C)$ и $(A \,\&\, (B \vee C))$ — формулы.

Подформулой называется всякая часть (подстрока) формулы, которая сама является формулой.

Оценкой (логических переменных) называется соответствие, согласно которому каждой переменной сопоставляется значение 0 или 1.

Например, если согласно оценке v переменная A имеет значение 0, а переменная B имеет значение 1, то пишут $v(A) = 0$, $v(B) = 1$.

Дадим определение истинности формулы на заданной оценке. Для этого введем оценку формул — соответствие, согласно которому каждой формуле сопоставляется значение 0 или 1.

Если φ — формула, то

$$v(\neg\varphi) = \begin{cases} 0, & \text{если } v(\varphi) = 1, \\ 1, & \text{если } v(\varphi) = 0. \end{cases}$$

Если φ, ψ — формулы, то

$$v(\varphi \,\&\, \psi) = \begin{cases} 1, & \text{если } v(\varphi) = 1 \text{ и } v(\psi) = 1, \\ 0, & \text{иначе,} \end{cases}$$

$$v(\varphi \vee \psi) = \begin{cases} 1, & \text{если } v(\varphi) = 1 \text{ или } v(\psi) = 1, \\ 0, & \text{иначе,} \end{cases}$$

$$v(\varphi \to \psi) = \begin{cases} 1, & \text{если } v(\varphi) = 0 \text{ или } v(\psi) = 1, \\ 0, & \text{иначе.} \end{cases}$$

Формулы φ и ψ *равносильны* ($\varphi = \psi$), если их оценки совпадают на любой оценке переменных.

Например, высказывание B' выражается равносильными формулами:

$$B' = \neg B \,\&\, \neg S \,\&\, \neg M = \neg(B \vee S \vee M).$$

При записи формул обычно пользуются соглашением не ставить скобки, которые однозначно восстанавливаются исходя из приоритета логических операций. Кроме этого, вводят сокращение $(\varphi \leftrightarrow \psi)$ для формулы $((\varphi \to \psi) \,\&\, (\psi \to \varphi))$.

1.2.4 Основные равносильности

Рассмотрим задачу упрощения произвольной формулы φ. Для этого применяют равносильные преобразования, в ходе которых формула последовательно заменяется на равносильную ей формулу.

Первое правило подстановки. Если в тождественно истинной формуле заменить все вхождения какой-нибудь переменной на произвольную формулу, то получится тождественно истинная формула.

Второе правило подстановки. Если заменить какую-нибудь подформулу формулы φ на равносильную ей формулу, то получится формула, равносильная формуле φ.

Перечислим основные равносильности алгебры высказываний. На основании первого правила подстановки входящие в эти равносильности переменные x, y, z могут быть заменены на любые формулы. Ниже $\mathbf{0}$ обозначает тождественно ложную формулу, $\mathbf{1}$ — тождественно истинную формулу.

Рассмотрим первую группу равносильностей (законов) алгебры логики — *свойства относительно констант.*

1. Если любое истинностное значение умножить на 1, то в результате получится это же значение. Если любое истинностное значение умножить на 0, то получится 0:

$$\boxed{x \,\&\, \mathbf{1} = x, \quad x \,\&\, \mathbf{0} = \mathbf{0}}$$

2. Если к любому значению прибавить 0, то получится это значение. Если к любому значению прибавить 1, то получится 1:

$$\boxed{x \vee \mathbf{0} = x, \quad x \vee \mathbf{1} = \mathbf{1}}$$

3. Из 0 следует любое значение. Из любого значения следует 1. Из значения следует 0 только тогда, когда это значение равно 0:

$$\boxed{\mathbf{0} \to x = \mathbf{1}, \quad x \to \mathbf{1} = \mathbf{1}, \quad x \to \mathbf{0} = \neg x}$$

Следующая группа — *законы идемпотентности*, то есть приведения подобных слагаемых или множителей.

4. При умножении двух одинаковых значений получается такое же значение:

$$\boxed{x \,\&\, x = x}$$

5. При сложении двух одинаковых значений получается такое же значение:

$$\boxed{x \vee x = x}$$

Классические законы с одной переменной.

6. *Закон исключенного третьего*:

$$\boxed{x \vee \neg x = \mathbf{1}}$$

7. *Закон непротиворечия*:

$$\boxed{x \,\&\, \neg x = \mathbf{0}}$$

8. *Закон двойного отрицания*:

$$\boxed{\neg\neg x = x}$$

Основные законы с двумя переменными.

9. *Формула замены импликации*: импликация ложна тогда и только тогда, когда слева — истина, а справа — ложь, поэтому истинна тогда и только тогда, когда слева ложь или справа истина:

$$\boxed{x \to y = \neg x \vee y}$$

10. *Коммутативность конъюнкции*: при перестановке множителей результат не меняется:

$$\boxed{x \,\&\, y = y \,\&\, x}$$

11. *Коммутативность дизъюнкции*: при перестановке слагаемых результат не меняется:

$$\boxed{x \vee y = y \vee x}$$

12. *Закон двойственности (де Моргана)*:

$$\boxed{\neg(x \,\&\, y) = \neg x \vee \neg y}$$

Левая часть равенства равна 1 всегда, за исключением случая $x = 1, y = 1$ (поскольку конъюнкция равна 1 только тогда, когда оба операнда 1). То же справедливо и

для правой части равенства (дизъюнкция равна 0 только тогда, когда оба операнда 0). Если левая часть всегда равна правой, то формулы равносильны.

13. *Закон двойственности (де Моргана)*:

$$\boxed{\neg(x \vee y) = \neg x \,\&\, \neg y}$$

Левая часть равенства равна 0 всегда, за исключением случая $x = 0, y = 0$ (поскольку дизъюнкция равна 0 только тогда, когда оба операнда 0). То же справедливо и для правой части равенства (конъюнкция равна 1 только тогда, когда оба операнда 1). Если левая часть всегда равна правой, то формулы равносильны.

Важные законы с тремя переменными.

14. *Ассоциативность конъюнкции*: если в длинной конъюнкции иначе расставить скобки, результат не изменится:

$$\boxed{(x \,\&\, y) \,\&\, z = x \,\&\, (y \,\&\, z)}$$

15. *Ассоциативность дизъюнкции*: если в длинной дизъюнкции иначе расставить скобки, результат не изменится:

$$\boxed{(x \vee y) \vee z = x \vee (y \vee z)}$$

16. *Дистрибутивность конъюнкции относительно дизъюнкции*: результат умножения общего множителя на сумму равен результату сложения двух умножений при раскрытии скобок:

$$\boxed{x \,\&\, (y \vee z) = (x \,\&\, y) \vee (x \,\&\, z)}$$

17. *Дистрибутивность дизъюнкции относительно конъюнкции*: результат суммирования общего слагаемого с умножением равен результату умножения двух суммирований при раскрытии скобок:

$$\boxed{x \vee (y \,\&\, z) = (x \vee y) \,\&\, (x \vee z)}$$

Докажем первый закон дистрибутивности. Рассмотрим случай $x = 0$ — левая часть равна 0, правая часть также равна 0, поэтому равенство выполняется. Рассмотрим случай $x = 1$ — левая часть равна $y \vee z$, правая часть также равна $y \vee z$.

Доказательство второго закона дистрибутивности можно найти в [18, с. 21].

Полезно знать следующие законы, которые помогают упрощать логические формулы.

18. *Формула поглощения:*

$$\boxed{x \vee (x \,\&\, y) = x}$$

При умножении x на y значение $x \,\&\, y$ становится не больше, чем x. Значит, при сложении x и $x \,\&\, y$ получится большее значение, то есть x.

19. *Формула поглощения:*

$$\boxed{x \,\&\, (x \vee y) = x}$$

При сложении x и y значение $x \vee y$ становится не меньше, чем x. Значит, при умножении x на $x \vee y$ получится меньшее значение, то есть x.

20. *Формула склеивания:*

$$\boxed{(x \,\&\, y) \vee (x \,\&\, \neg y) = x}$$

Заметим, что из двух слагаемых одно всегда равно 0, а другое равно x. Поэтому их сумма равна x.

21. *Формула склеивания:*

$$\boxed{(x \vee y)\,\&\,(x \vee \neg y) = x}$$

Заметим, что из двух множителей одно всегда равно 1, а другое равно x. Поэтому их произведение равно x.

22. *Формула сокращения:*

$$\boxed{x \vee (\neg x\,\&\,y) = x \vee y}$$

Согласно второму закону дистрибутивности и закону исключенного третьего

$$x \vee (\neg x\,\&\,y) = (x \vee \neg x)\,\&\,(x \vee y) = \mathbf{1}\,\&\,(x \vee y) = x \vee y.$$

23. *Формула сокращения:*

$$\boxed{x\,\&\,(\neg x \vee y) = x\,\&\,y}$$

Согласно первому закону дистрибутивности и закону непротиворечия

$$x\,\&\,(\neg x \vee y) = (x\,\&\,\neg x) \vee (x\,\&\,y) = \mathbf{0} \vee (x\,\&\,y) = x\,\&\,y.$$

1.2.5 Нормальные формы

Абитуриент выбирает между компьютерными специальностями и психологией. В обоих случаях нужно сдать русский язык и математику, но на компьютерных специальностях нужно также сдать физику (A) или информатику, или английский язык (B), а для поступления на психологию нужно сдать биологию (C) или обществознание, или английский язык (B). Абитуриент хочет выбрать два экзамена, к которым он будет готовиться, помимо математики и русского языка, но кроме информатики и обществознания, чтобы можно было подать документы как на компьютерные специальности, так и на психологию. К каким экзаменам следует готовиться?

Если сдавать английский язык ($B = 1$), то в качестве второго экзамена можно выбрать физику или биологию, то есть должно выполняться $A \vee C = 1$ и $A\&C = 0$. Если не сдавать английский язык ($B = 0$), то нужно сдать как физику, так и биологию, то есть $A\,\&\,C = 1$.

Изобразим подходящие значения переменных A, B, C в виде таблицы истинности булевой функции $F(A, B, C)$ (табл. 1.7).

Таблица 1.7

Таблица истинности

A	B	C	$F(A,B,C)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Известно, что

$$F(A, 1, C) = (A \vee C)\,\&\,\neg(A\,\&\,C), \quad F(A, 0, C) = A\,\&\,C.$$

По теореме Шеннона о дизъюнктивном разложении булевой функции по одной переменной [9, с. 34] справедливо тождество

$$F(A,B,C) = (B \,\&\, F(A,1,C)) \vee (\neg B \,\&\, F(A,0,C)).$$

Поэтому

$$F(A,B,C) = \big(B \,\&\, (A \vee C) \,\&\, \neg(A \,\&\, C)\big) \vee \big(\neg B \,\&\, (A \,\&\, C)\big).$$

Применив закон де Моргана и раскрыв скобки (по первому закону дистрибутивности), получим дизъюнкцию элементарных конъюнкций:

$$f(A,B,C) = B \,\&\, (A \vee C) \,\&\, (\neg A \vee \neg C) \vee (A \,\&\, \neg B \,\&\, C) =$$
$$= B\&\big((A\&\neg C)\vee(\neg A\&C)\big)\vee(A\&\neg B\&C) = (A\&B\&\neg C)\vee(\neg A\&B\&C)\vee(A\&\neg B\&C).$$

Полученная формула так и читается: или сдавай физику и английский язык и не сдавай биологию ($AB\overline{C}$), или сдавай английский язык и биологию и не сдавай физику ($\overline{A}BC$), или сдавай физику и биологию и не сдавай английский язык ($A\overline{B}C$).

Элементарной конъюнкцией называется конъюнкция логических переменных или их отрицаний. Одна логическая переменная или ее отрицание тоже считаются элементарной конъюнкцией.

Формулу, записанную в виде дизъюнкции элементарных конъюнкций, называют *дизъюнктивной нормальной формой* (ДНФ).

Полной элементарной конъюнкцией называется элементарная конъюнкция, в которую входят все переменные по одному разу.

Дизъюнкция неповторяющихся полных элементарных конъюнкций носит название *совершенной ДНФ* (СДНФ).

Укажем абитуриенту, чего делать *не* нужно. Для этого применим теорему Шеннона о конъюнктивном разложении булевой функции по одной переменной [9, с. 35]:

$$F(A,B,C) = (\neg B \vee F(A,1,C)) \,\&\, (B \vee F(A,0,C)).$$

Отсюда

$$F(A,B,C) = \big(\neg B \vee ((A \vee C) \,\&\, \neg(A \,\&\, C))\big) \,\&\, \big(B \vee (A \,\&\, C)\big).$$

Применив второй закон дистрибутивности, получим конъюнкцию элементарных дизъюнкций:

$$F(A,B,C) = \big(\neg B \vee ((A \vee C) \,\&\, (\neg A \vee \neg C))\big) \,\&\, (B \vee A) \,\&\, (B \vee C) =$$
$$= (A \vee \neg B \vee C) \,\&\, (\neg A \vee \neg B \vee \neg C) \,\&\, (A \vee B) \,\&\, (B \vee C).$$

Запишем формулу, используя закон де Моргана:

$$F(A,B,C) = \neg(\neg A \,\&\, B \,\&\, \neg C) \,\&\, \neg(A \,\&\, B \,\&\, C) \,\&\, \neg(\neg A \,\&\, \neg B) \,\&\, \neg(\neg B \,\&\, \neg C).$$

Эту запись можно прочитать так: не надо сдавать только английский язык, не надо сдавать все три предмета, не надо пропускать одновременно физику и английский язык или одновременно английский язык и биологию.

Элементарной дизъюнкцией называется дизъюнкция логических переменных или их отрицаний. Одна логическая переменная или ее отрицание тоже считаются элементарной дизъюнкцией.

Формулу, записанную в виде конъюнкции элементарных дизъюнкций, называют *конъюнктивной нормальной формой* (КНФ).

Полной элементарной дизъюнкцией называется элементарная дизъюнкция, в которую входят все переменные по одному разу.

Конъюнкция неповторяющихся полных элементарных дизъюнкций носит название *совершенной КНФ* (СКНФ).

Чтобы получить СКНФ из нашей формулы, прибавим к «несовершенным» множителям КНФ ноль, который распишем по закону непротиворечия с участием недостающей переменной:

$$A \vee B = A \vee B \vee 0 = A \vee B \vee (C \,\&\, \neg C),$$

а затем воспользуемся вторым законом дистрибутивности:

$$A \vee B = A \vee B \vee (C \,\&\, \neg C) = (A \vee B \vee C) \,\&\, (A \vee B \vee \neg C).$$

Аналогично

$$B \vee C = (A \vee B \vee C) \,\&\, (\neg A \vee B \vee C).$$

Получим

$$F(A, B, C) = (A \vee \neg B \vee C) \,\&\, (\neg A \vee \neg B \vee \neg C) \,\&\, (A \vee B \vee C) \,\&\, (A \vee B \vee \neg C) \,\&\, (\neg A \vee B \vee C).$$

Пусть дана булева функция. Поставим задачу построения формулы, таблица истинности которой задает данную булеву функцию. Для примера рассмотрим операцию эквиваленции: $f(x, y) = x \leftrightarrow y$ (табл. 1.8).

Таблица 1.8

Нормальные формы для эквиваленции

x	y	$x \leftrightarrow y$	СДНФ	СКНФ	$\overline{x} \,\&\, \overline{y}$	$x \,\&\, y$	$x \vee \overline{y}$	$\overline{x} \vee y$
0	0	1	$x^0 \,\&\, y^0 = \overline{x} \,\&\, \overline{y}$		1	0	1	1
0	1	0		$x^1 \vee y^0 = x \vee \overline{y}$	0	0	0	1
1	0	0		$x^0 \vee y^1 = \overline{x} \vee y$	0	0	1	0
1	1	1	$x^1 \,\&\, y^1 = x \,\&\, y$		0	1	1	1

Каждой паре значений переменных x, y, которым соответствует значение $f(x, y) = 1$, припишем в колонке СДНФ полную элементарную конъюнкцию, равную 1 только при таких значениях переменных. Аналогично каждой паре значений переменных x, y, которым соответствует значение $f(x, y) = 0$, припишем в колонке СКНФ полную элементарную дизъюнкцию, равную 0 только при таких значениях переменных.

Полная элементарная конъюнкция, содержащая одну и ту же переменную с отрицанием и без отрицания, тождественно равна 0. Полная элементарная конъюнкция, содержащая все переменные по одному разу, равна 1 только при одном наборе значений переменных. Например, $\overline{x} \,\&\, \overline{y} = 1$ только при $\overline{x} = 1$, $\overline{y} = 1$, то есть $x = 0$, $y = 0$.

Полная элементарная дизъюнкция, содержащая одну и ту же переменную с отрицанием и без отрицания, тождественно равна 1. Полная элементарная дизъюнкция, содержащая все переменные по одному разу, равна 0 только при одном наборе значений переменных. Например, $\overline{x} \vee y = 0$ только при $\overline{x} = 0$, $y = 0$, то есть $x = 1$, $y = 0$.

Введем обозначения: $x^1 = x$, $x^0 = \overline{x}$. Чтобы записать полную элементарную конъюнкцию, равную 1 в заданной строке таблицы истинности, нужно возвести все переменные в степени, равные значению переменной в этой строке. Чтобы записать полную элементар-

ную дизъюнкцию, равную 0 в заданной строке таблицы истинности, нужно возвести все переменные в степени, противоположные значению переменной в этой строке.

Если построить таблицы истинности выражений $\overline{x}\,\&\,\overline{y}$ и $x\,\&\,y$, а затем взять их дизъюнкцию, то получим в точности таблицу истинности $f(x,y)$. Если построить таблицы истинности выражений $x \vee \overline{y}$ и $\overline{x} \vee y$ и перемножить их, то также получим $f(x,y)$.

Итак, мы представили булеву функцию $f(x,y)$ в СДНФ:

$$x \leftrightarrow y = (\overline{x}\,\&\,\overline{y}) \vee (x\,\&\,y)$$

и СКНФ:

$$x \leftrightarrow y = (x \vee \overline{y})\,\&\,(\overline{x} \vee y).$$

1.2.6 Логическое следствие

Обозначим A = "человек является студентом", B = "у него есть пропуск для входа в университет", C = "ему оформлен временный пропуск". Возможность войти в здание университета определяется истинностью логической формулы $A\,\&\,B \vee C$. В то же время можно заметить, что всякий раз, когда человеку разрешен вход, справедлива истинность высказывания $\neg A \to C$: если это не студент (и ему разрешен вход), то ему оформлен временный пропуск.

Формула ψ является *логическим следствием* формул $\varphi_1, \varphi_2, \ldots, \varphi_n$ (обозначение: $\varphi_1, \varphi_2, \ldots, \varphi_n \models \psi$), если всякий раз, когда истинны одновременно все посылки $\varphi_1 = 1$, $\varphi_2 = 1$, …, $\varphi_n = 1$, истинно и заключение $\psi = 1$.

В нашем примере

$$A\,\&\,B \vee C \models \neg A \to C.$$

Может быть, эти высказывания равносильны? При $A = 1, B = 0, C = 0$ (студент не имеет пропуска) левая часть ложна (вход не разрешен), а правая истинна (если он не студент, то у него есть временный пропуск). Значит, равносильность не выполняется.

Итак, логическое следствие позволяет получить информацию об истинности одной логической формулы (заключения) при владении информацией об истинности другой формулы (посылки).

Для проверки логического следствия можно пользоваться следующими критериями.

1-й критерий логического следствия:

$$\boxed{\varphi \models \psi \quad \Leftrightarrow \quad \varphi \to \psi \equiv 1}$$

На самом деле всякий раз, когда $\varphi = 1$, должно быть выполнено $\psi = 1$. В случае же $\varphi = 0$ допускается любое значение ψ.

2-й критерий логического следствия:

$$\boxed{\varphi \models \psi \quad \Leftrightarrow \quad \varphi\,\&\,\neg\psi \equiv 0}$$

Второй критерий проверяет невозможность нарушения логического следствия. А когда оно может нарушиться? Только когда $\varphi = 1$ и $\psi = 0$.

В свою очередь, чтобы установить тождественную истинность или ложность соответствующей формулы, достаточно вычислить ее значения на всевозможных оценках, построив таблицу истинности. Другой способ — используя равносильные преобразования, упростить формулу до константы либо до элементарной дизъюнкции или элементарной конъюнкции, про которые известно, что они не тождественно истинны/ложны.

Для проверки верности утверждения $\varphi_1, \varphi_2, \ldots, \varphi_n \models \psi$ можно использовать *метод от противного*, или метод редукции. Допустим, что ψ не является логическим следствием $\varphi_1, \ldots, \varphi_n$. Тогда существует набор значений логических переменных, от которых зависят формулы $\varphi_1, \ldots, \varphi_n, \psi$, такой, что $\psi = 0$ на этом наборе, несмотря на то что $\varphi_1 = 1, \ldots, \varphi_n = 1$ на этом наборе.

Составим систему логических уравнений и попробуем ее решить. Если окажется, что система не имеет решения, то логическое следствие выполнено. В противном случае все решения системы — это контрпримеры, на которых логическое следствие нарушается.

В примере с пропуском система уравнений имеет следующий вид:

$$\begin{cases} A \,\&\, B \vee C = 1, \\ \quad \neg A \to C = 0. \end{cases}$$

Из второго уравнения следует, что импликация ложна только в одном случае: $\neg A = 1$, $C = 0$, или $A = 0$, $C = 0$. Подставляя найденные значения переменных в первое уравнение, получим $0 \,\&\, B \vee 0 = 1$, или $0 = 1$, что никогда не выполняется. Значит, два уравнения системы противоречат друг другу и система решений не имеет. Поэтому логическое следствие не может быть нарушено ни при какой оценке.

1.2.7 Метод резолюций

Допустим, что у нас есть формула в конъюнктивной нормальной форме — это конъюнкция элементарных дизъюнкций. Метод резолюций позволяет проверить эту формулу на тождественную ложность. В частности, пользуясь вторым критерием логического следствия, можно ответить на вопрос, является ли формула следствием другого набора формул.

Метод резолюций основан на следующем правиле резолюции:

$$\varphi \vee x_i,\ \psi \vee \neg x_i \ \models\ \varphi \vee \psi,$$

где φ, ψ — формулы; x_i — переменная. В этом случае $\varphi \vee \psi$ будем обозначать через $\mathrm{res}_{x_i}(\varphi, \psi)$.

Например, A = “Кипятили воду”, B = “Не готовили обед”, C = “Плита выключена”. Допустим, мы знаем, что, во-первых, кипятили воду или плита выключена $(A \vee C)$, то есть если плита не выключена, то кипятили воду, и, во-вторых, не готовили обед или плита не выключена $(B \vee \neg C)$, то есть если плита выключена, то не готовили обед. Значит, по правилу резолюций получаем, что кипятили воду или не готовили обед $(A \vee B)$, то есть ситуация, когда не кипятили воду и готовили обед, невозможна.

Покажем на примере, как работает метод резолюций. В примере с пропуском (см. подразд. 1.2.6) воспользуемся вторым критерием логического следствия. Составим формулу, для которой нужно проверить тождественную ложность:

$$A \,\&\, B \vee C \models \neg A \to C \quad \Leftrightarrow \quad (A \,\&\, B \vee C) \,\&\, \neg(\neg A \to C) \equiv 0.$$

Приведем эту формулу к КНФ. Воспользуемся вторым законом дистрибутивности, формулой замены импликации и законом де Моргана:

$$(A \,\&\, B \vee C) \,\&\, \neg(\neg A \to C) = (A \vee C) \,\&\, (B \vee C) \,\&\, \neg A \,\&\, \neg C.$$

Выпишем все дизъюнкты:

1) $A \vee C$;

2) $B \vee C$;
3) $\neg A$;
4) $\neg C$.

Применим правило резолюции к *резольвирующим* дизъюнктам, то есть содержащим ровно одну пару противоположных литер. *Резольвента* двух таких дизъюнктов будет их логическим следствием, и ее просто добавим к имеющимся дизъюнктам.

Тогда
5) $\mathrm{res}_A(A \vee C, \neg A) = C$;
6) $\mathrm{res}_C(\neg C, C) = 0$.

Метод резолюций позволяет вывести из исходного множества дизъюнктов всевозможные логические следствия. Если исходное множество дизъюнктов противоречиво (то есть не обращается одновременно в 1 ни при одной оценке), то его логическим следствием будет константа 0. Если исходное множество дизъюнктов совместно, то метод резолюций зайдет в тупик — остановится, не выведя 0 (*пустую резольвенту*).

Теорема (полнота метода резолюций). Если множество дизъюнктов S противоречиво, то из него выводима по правилу резолюции константа 0. Это означает, что существует последовательность дизъюнктов $\varphi_1, \ldots, \varphi_n$, такая, что для любого i или $\varphi_i \in S$, или $\varphi_i = \mathrm{res}(\varphi_j, \varphi_k)$ для некоторых $j < i$, $k < i$ и $\varphi_n = 0$.

Доказательство см. в [2, с. 355].

1.3 Практикум

1.3.1 Типовые примеры

Пример 1.1. Формализуем высказывание: “Диоген бил отца, если сын сквернословил”.

Введем элементарные высказывания: A = “Диоген бил чьего-то отца” и B = “Сын сквернословил”. Нам известно, что всякий раз, когда истинно B, истинно и A, и больше никаких ограничений на возможные значения переменных A и B у нас нет. Полученная информация означает в точности истинность импликации $B \to A$.

Пример 1.2. Дано высказывание: “Не бывает глупых девушек-программистов”. Построим таблицу истинности для этого высказывания относительно элементарных высказываний A = “Этот человек — девушка”, B = “Этот человек глуп”, C = “Этот человек — программист”.

A девушка	B глуп	C программист	Бывает?
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Таблица истинности сообщает, какие из возможных комбинаций осуществимы. Запишем формулу в СДНФ, выражающую данную таблицу. Для этого просто перечислим все осуществимые комбинации, выразим их в виде конъюнктов и сложим.

Напомним, что мы перечисляем все истинные строки таблицы истинности, для каждой из них берем значения переменных и возводим переменные в такие степени:

$$A^0B^0C^0 \vee A^0B^0C^1 \vee A^0B^1C^0 \vee A^0B^1C^1 \vee A^1B^0C^0 \vee A^1B^0C^1 \vee A^1B^1C^0 =$$
$$= (\overline{A}\&\overline{B}\&\overline{C}) \vee (\overline{A}\&\overline{B}\&C) \vee (\overline{A}\&B\&\overline{C}) \vee (\overline{A}\&B\&C) \vee (A\&\overline{B}\&\overline{C}) \vee (A\&\overline{B}\&C) \vee (A\&B\&\overline{C}).$$

Чтобы записать формулу в СКНФ, рассмотрим ложную строку таблицы истинности и составим полную элементарную дизъюнкцию, возведя переменные в степени, противоположные значениям переменных в данной строке:

$$A^0 \vee B^0 \vee C^0 = \overline{A} \vee \overline{B} \vee \overline{C}.$$

Вспомним закон де Моргана:

$$\overline{A} \vee \overline{B} \vee \overline{C} = \overline{A \,\&\, B \,\&\, C}.$$

Полученная формула ближе всего к тексту: неправда, что человек одновременно девушка, глуп и программист.

На самом деле не всё “логичное” — правда. Логика дает нам возможность делать правильные умозаключения и формулировать свои суждения на общепринятом языке. Но это не значит, что всё, что мы можем записать математически, обязательно окажется истиной! Можно вспомнить высказывание русского физиолога И. П. Павлова: “Если я рассуждаю логично, это значит только то, что я не сумасшедший, но вовсе не доказывает, что я прав”.

Пример 1.3. Формализуем высказывание: “Чтобы оставаться здоровым, нужно не курить и не пить”.

Чтобы не обмануть себя при составлении формулы, сначала построим таблицу, в которой для каждой возможной комбинации укажем её осуществимость.

A здоров	B курит	C пьет	Бывает?
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Составим СДНФ, перечислив все истинные возможности:

$$(\overline{A} \,\&\, \overline{B} \,\&\, \overline{C}) \vee (\overline{A} \,\&\, \overline{B} \,\&\, C) \vee (\overline{A} \,\&\, B \,\&\, \overline{C}) \vee (\overline{A} \,\&\, B \,\&\, C) \vee (A \,\&\, \overline{B} \,\&\, \overline{C}).$$

Сократим ДНФ. Для этого добавим все возможные результаты склеиваний:

$$(\overline{A} \,\&\, \overline{B} \,\&\, \overline{C}) \vee (\overline{A} \,\&\, \overline{B} \,\&\, C) \vee (\overline{A} \,\&\, B \,\&\, \overline{C}) \vee (\overline{A} \,\&\, B \,\&\, C) \vee (A \,\&\, \overline{B} \,\&\, \overline{C}) =$$
$$= (\overline{A} \,\&\, \overline{B} \,\&\, \overline{C}) \vee (\overline{A} \,\&\, \overline{B} \,\&\, C) \vee (\overline{A} \,\&\, B \,\&\, \overline{C}) \vee (\overline{A} \,\&\, B \,\&\, C) \vee (A \,\&\, \overline{B} \,\&\, \overline{C}) \vee$$
$$\vee (\overline{A} \,\&\, \overline{B}) \vee (\overline{A} \,\&\, \overline{C}) \vee (\overline{B} \,\&\, \overline{C}) \vee (\overline{A} \,\&\, C) \vee (\overline{A} \,\&\, B) \vee \overline{A},$$

после чего оставим только результаты всех возможных поглощений:

$$(\overline{A}\,\&\,\overline{B}\,\&\,\overline{C}) \vee (\overline{A}\,\&\,\overline{B}\,\&\,C) \vee (\overline{A}\,\&\,B\,\&\,\overline{C}) \vee (\overline{A}\,\&\,B\,\&\,C) \vee (A\,\&\,\overline{B}\,\&\,\overline{C}) = \overline{A} \vee (\overline{B}\,\&\,\overline{C}).$$

Полученную формулу перепишем, используя формулу замены импликации и закон де Моргана:

$$\overline{A} \vee (\overline{B}\,\&\,\overline{C}) = A \to \overline{B \vee C}$$

или

$$\overline{A} \vee (\overline{B}\,\&\,\overline{C}) = B \vee C \to \overline{A}.$$

В последней формуле отчетливо читается смысл высказывания: если человек курит или пьет, то он не будет здоров.

Наконец, составим СКНФ, для чего перечислим все ложные возможности:

$$(\overline{A} \vee B \vee \overline{C})\,\&\,(\overline{A} \vee \overline{B} \vee C)\,\&\,(\overline{A} \vee \overline{B} \vee \overline{C}).$$

Каждый множитель равен 0 только в одной из строк таблицы истинности, где стоит 0.

Физику Л. Д. Ландау приписывают такое высказывание: “Если ты чего-то не понял, прочти еще раз. Если не понял после пяти раз, значит, ты — дурак”.

Вывод: не сдавайтесь, если что-нибудь не получается понять с самого начала!

Пример 1.4. Запишем предложение на языке логики: “Выходила Катюша на высокий и крутой берег”.

Цель перевода высказывания на язык логики — вычисление его значения. Чтобы вычисление стало возможным, нужно указать исходные данные. В предложении говорится о двух объектах: человеке по имени Катюша и береге, обладающем двумя свойствами. Допустим, что наличие этих свойств у всех берегов сведено в таблицы T_1, T_2. Говорится также о соответствии между человеком и берегом. Эта информация сведена в таблицу R. Тогда формула $R(k,b)\,\&\,T_1(b)\,\&\,T_2(b)$ вычисляется так: берется значение “выходить” из таблицы R для человека k и берега b, затем берутся значения из таблиц T_1 “высокий” и T_2 “крутой” для берега b и эти три значения перемножаются. Таким образом, для любого человека k и любого берега b будет вычислена истинность сказанного. Формула примет значение 1, только если $R(k,b) = 1$, $T_1(b) = 1$ и $T_2(b) = 1$.

Пример 1.5. Докажем формулы поглощения и склеивания, пользуясь законами дистрибутивности:

$$x \vee (x\,\&\,y) = (x\,\&\,1) \vee (x\,\&\,y) = x\,\&\,(1 \vee y) = x\,\&\,1 = x,$$
$$x\,\&\,(x \vee y) = (x \vee 0)\,\&\,(x \vee y) = x \vee (0\,\&\,y) = x \vee 0 = x;$$
$$(x\,\&\,y) \vee (x\,\&\,\neg y) = x\,\&\,(y \vee \neg y) = x\,\&\,1 = x,$$
$$(x \vee y)\,\&\,(x \vee \neg y) = x \vee (y\,\&\,\neg y) = x \vee 0 = x.$$

— Обнять его или обнять и поцеловать?

— По формуле поглощения — только обнять.

— Подарить ей букет и шоколадку или подарить букет, а шоколадку не дарить?

— По формуле склеивания — остается подарить букет.

Пример 1.6. Запишем формулу составного высказывания “Не хочу учиться, хочу жениться” относительно элементарных высказываний A = “Хочу учиться”, B = “Хочу жениться”, C = “Хочу есть”.

Утверждается истинность формулы на всех оценках $v(A), v(B), v(C)$, на которых $v(A) = 0$ и $v(B) = 1$, то есть $A = 0$, $B = 1$, C любое. Можно составить таблицу истинности и выделить истинные строки, после чего записать СДНФ и сократить. А можно вспомнить связь между формулами и их оценками (см. подразд. 1.2.3).

$$v(\neg A) = \begin{cases} 0, & \text{если } v(A) = 1, \\ 1, & \text{если } v(A) = 0, \end{cases}$$

$$v(\neg A \,\&\, B) = \begin{cases} 1, & \text{если } v(\neg A) = 1 \text{ и } v(B) = 1, \text{ т.е. } v(A) = 0,\ v(B) = 1, \\ 0, & \text{иначе.} \end{cases}$$

Приведем полученную формулу $\neg A \,\&\, B$ к СДНФ:

$$\neg A \,\&\, B = \overline{A} \,\&\, B \,\&\, (C \vee \overline{C}) = \overline{A} \,\&\, B \,\&\, C \vee \overline{A} \,\&\, B \,\&\, \overline{C}.$$

Пример 1.7. Профессор экономики сформулировал студентам три правила:
- если инвестиции не увеличатся, то возрастут правительственные расходы или возникнет безработица;
- если правительственные расходы не возрастут, то налоги снизятся;
- если налоги снизятся и инвестиции увеличатся, то не возникнет безработица.

Оказалось, что в данный момент инвестиции не увеличились. Могут ли студенты сделать вывод о том, что правительственные расходы возрастут?

Определим элементарные высказывания: $A =$ “Инвестиции увеличились”, $B =$ “Правительственные расходы возросли”, $C =$ “Возникла безработица”, $D =$ “Налоги снизились”. Формализуем рассуждения:

$$\neg A \to B \vee C,\ \neg B \to D,\ D \,\&\, A \to \neg C,\ \neg A \models B. \tag{1.2}$$

По второму критерию логического следствия данное логическое следствие будет выполнено, если

$$(\neg A \to B \vee C) \,\&\, (\neg B \to D) \,\&\, (D \,\&\, A \to \neg C) \,\&\, \neg A \,\&\, \neg B \equiv 0.$$

Приведем эту формулу к КНФ, чтобы затем применить метод резолюций:

$$\begin{aligned} (\neg A \to B \vee C) \,\&\, (\neg B \to D) \,\&\, (D \,\&\, A \to \neg C) \,\&\, \neg A \,\&\, \neg B = \\ = (A \vee B \vee C) \,\&\, (B \vee D) \,\&\, (\overline{D \,\&\, A} \vee \overline{C}) \,\&\, \overline{A} \,\&\, \overline{B} = \\ = (A \vee B \vee C) \,\&\, (B \vee D) \,\&\, (\overline{D} \vee \overline{A} \vee \overline{C}) \,\&\, \overline{A} \,\&\, \overline{B}. \end{aligned}$$

Напомним идею метода резолюций: на каждом шаге нужно найти два дизъюнкта, содержащих ровно одну противоположную переменную (которая входит в один дизъюнкт с отрицанием, а в другой без отрицания). Если в конце получаем противоречие, значит, наша формула невыполнима (тождественно ложна). Если получить противоречие не получается и метод зашел в тупик, значит, формула равна 1 при некоторых оценках:

1) $A \vee B \vee C$;
2) $B \vee D$;
3) $\overline{D} \vee \overline{A} \vee \overline{C}$;
4) $\overline{A}$;
5) $\overline{B}$;
6) $B \vee C$ — из 1, 4;
7) C — из 5, 6;

8) D — из 2, 5;
9) $\overline{D} \vee \overline{A}$ — из 3, 7;
10) $A \vee C$ — из 1, 5;
11) $C \vee \overline{D}$ — из 9, 10;
12) $B \vee C \vee \overline{D}$ — из 1, 9;
13) и так далее...

Видим, что вывести противоречие у нас пока не получается. Перебор всевозможных резольвент мы предоставим компьютеру. Для себя мы уже получили полезную информацию. Поскольку все резольвенты являются логическими следствиями нашей формулы, то, чтобы отыскать, когда она равна 1, возьмем найденные значения переменных A, B, C и D: $A = 0$, $B = 0$, $C = 1$, $D = 1$ (см. формулы 4, 5, 7, 8). Если подставить эти значения в формулу (1.2), легко убедиться в том, что логическое следствие нарушается на этой оценке: все посылки истинны, а заключение ложно.

Значит, в случае, когда инвестиции не увеличились, возникла безработица и налоги снизились; возможен случай, когда правительственные расходы не возросли, и это не будет противоречить ни одному из озвученных профессором правил.

Пример 1.8. Установим правильность рассуждения: “Если данное явление психическое, то оно обусловлено внешним воздействием на организм. Если оно физиологическое, то тоже обусловлено внешним воздействием на организм. Данное явление не психическое и не физиологическое. Следовательно, оно не обусловлено внешним воздействием на организм.”

Формализуем высказывания и проверим логическое следствие. Элементарные высказывания: A = “Явление психическое”, B = “Явление физиологическое”, C = “Явление обусловлено внешним воздействием”. Посылки: $A \to C$, $B \to C$, $\neg A \,\&\, \neg B$. Заключение: $\neg C$.

Применяя метод от противного, найдем все такие оценки A, B, C, на которых все посылки истинны, а заключение ложно. Если таких оценок нет, то логическое следствие никогда не опровергается. Составим систему логических уравнений:

$$\begin{cases} A \to C = 1, \\ B \to C = 1, \\ \neg A \,\&\, \neg B = 1, \\ \neg C = 0. \end{cases}$$

Из последнего уравнения находим $C = 1$. Подставляя найденное значение C в первое и второе уравнения, получаем $A \to 1 = 1$ и $B \to 1 = 1$. Оба эти равенства верны при любых значениях A и B. Остается разобрать третье уравнение, которое раскладывется на два: $\neg A = 1$ и $\neg B = 1$, поскольку конъюнкция равна 1 только в одном случае, когда оба ее аргумента равны 1. Отсюда выводим $A = 0$, $B = 0$.

Итак, система имеет решение $A = 0,\ B = 0$, $C = 1$. Значит, в случае, когда явление не психическое, не физиологическое и обусловлено внешним воздействием, логическое следствие нарушается. Значит, рассуждения, вообще говоря, ошибочны.

Если, однако, изменить формализацию рассуждений и считать, что все явления бывают только психическими, физиологическими и никакими другими, то мы придем к противоречию: найденное нами решение уже не будет решением расширенной системы уравнений. В этом случае логическое следствие будет выполняться.

Упражнение. Добавьте к уравнениям $\neg(A \leftrightarrow B) = 1$ и проведите рассуждения.

Без известного самостоятельного труда ни в одном серьезном вопросе истины не найти, и кто боится труда, тот сам себя лишает возможности найти истину (В.И. Ленин).

Пример 1.9. Есть три коробки: на первой написано “Кошка в этой коробке”; на второй — “Кошка не в этой коробке”; на третьей — “Кошка не в первой коробке”. Известно, что правда написана только на одной коробке, а две остальные надписи лгут. В какой коробке находится кошка?

Пусть A = “Кошка в первой коробке”, B = “Кошка во второй коробке”, C = “Кошка в третьей коробке”. Тогда надписи на коробках можно записать так: A, $\neg B$, $\neg A$. Запишем формулу, утверждающую истинность ровно одного из этих высказываний:

$$(A \,\&\, \neg\neg B \,\&\, \neg\neg A) \vee (\neg A \,\&\, \neg B \,\&\, \neg\neg A) \vee (\neg A \,\&\, \neg\neg B \,\&\, \neg A).$$

Здесь первое слагаемое означает, что на первой коробке написана правда, а на второй и третьей — ложь; второе слагаемое означает, что на второй коробке написана правда, а на первой и третьей — ложь; третье слагаемое означает, что правда написана только на третьей коробке.

Преобразуем нашу формулу. Сначала снимем двойные отрицания:

$$(A \,\&\, B \,\&\, A) \vee (\neg A \,\&\, \neg B \,\&\, A) \vee (\neg A \,\&\, B \,\&\, \neg A).$$

Воспользуемся законом идемпотентности и законом непротиворечия:

$$(A \,\&\, B) \vee 0 \vee (\neg A \,\&\, B).$$

Получаем формулу, в которой вынесем общий множитель за скобку или сразу применим формулу склеивания:

$$(A \,\&\, B) \vee (\neg A \,\&\, B) = B.$$

Ответ: кошка во второй коробке.

Пример 1.10. Докажем логическое следствие:

$$(A \to B) \,\&\, (C \to D) \models A \,\&\, C \to B \,\&\, D.$$

Воспользуемся первым критерием логического следствия, согласно которому логическое следствие выполнено тогда и только тогда, когда следующая формула тождественно истинна:

$$((A \to B) \,\&\, (C \to D)) \to (A \,\&\, C \to B \,\&\, D)\,.$$

Займемся равносильными преобразованиями, применяя формулу замены импликации и законы де Моргана:

$$\begin{aligned}((A \to B) \,\&\, (C \to D)) \to (A \,\&\, C \to B \,\&\, D) &= \neg\,((A \to B) \,\&\, (C \to D)) \vee (A \,\&\, C \to B \,\&\, D) = \\ &= \neg\,((\neg A \vee B) \,\&\, (\neg C \vee D)) \vee (\neg(A \,\&\, C) \vee (B \,\&\, D)) = \\ &= \neg\,\big((\overline{A} \vee B) \,\&\, (\overline{C} \vee D)\big) \vee \big(\overline{A} \vee \overline{C} \vee (B \,\&\, D)\big) = \\ &= \big(\neg(\overline{A} \vee B) \vee \neg(\overline{C} \vee D)\big) \vee \big(\overline{A} \vee \overline{C} \vee (B \,\&\, D)\big) = \\ &= \big((A \,\&\, \overline{B}) \vee (C \,\&\, \overline{D})\big) \vee \big(\overline{A} \vee \overline{C} \vee (B \,\&\, D)\big)\,.\end{aligned}$$

Обратим внимание на структуру формулы и уберем лишние скобки:

$$(A \,\&\, \overline{B}) \vee (C \,\&\, \overline{D}) \vee \overline{A} \vee \overline{C} \vee (B \,\&\, D).$$

Переставим и сгруппируем слагаемые:

$$\big((A \,\&\, \overline{B}) \vee \overline{A}\big) \vee \big((C \,\&\, \overline{D}) \vee \overline{C}\big) \vee (B \,\&\, D).$$

Применим формулу сокращения: $xy \vee \overline{x} = y \vee \overline{x}$, которую запоминать не обязательно, а достаточно уметь ее выводить из второго закона дистрибутивности. Получим

$$(\overline{B} \vee \overline{A}) \vee (\overline{D} \vee \overline{C}) \vee (B \,\&\, D).$$

Снова уберем лишние скобки и перегруппируем слагаемые:

$$\overline{B} \vee (B \,\&\, D) \vee \overline{A} \vee \overline{D} \vee \overline{C} = \overline{B} \vee D \vee \overline{A} \vee \overline{D} \vee \overline{C}.$$

Полученный дизъюнкт содержит два противоположных литерала D и $\overline{D}$, а потому тождественно равен 1.

Итак, мы доказали тождественную истинность нашей формулы, что означает справедливость логического следствия.

Настоящий программист на вопрос «можешь ли ты это сделать?» ответит «да», а потом подумает, как.

Пример 1.11. Формализуем высказывание: “Не бывает такой работы, на которой много платят, а работать нужно мало”.

Определим множество X всех возможных мест работы. На этом множестве определим два предиката: $P(x)$ = “На работе x много платят”, $Q(x)$ = “На работе x нужно много работать”. Переменная x пробегает по множеству X.

Запишем формулу

$$\neg\exists x \left(P(x) \,\&\, \overline{Q(x)}\right).$$

Формула вычисляется так. Сначала для каждого x вычисляется область действия квантора, то есть выражение в скобках. И если найдется хотя бы одно значение x, для которого это выражение равно 1, то квантор равен 1, иначе 0. Таким образом, истинность отрицания квантора сообщает нам информацию о том, что такого x не существует.

Преобразуем формулу по закону де Моргана:

$$\forall x \left(\overline{P(x)} \vee Q(x)\right).$$

Так что на любой работе или мало платят, или нужно много работать, или и то, и другое.

Пример 1.12. Молодая пара пришла в китайский ресторан. Они решили, что закажут суп с морепродуктами или суп с помидорами, также салат харбинский или салат из одуванчиков и на горячее — мясо в кисло-сладком соусе или картошку с баклажанами. Девушка сказала, что картошка с баклажанами подходит только к супу с морепродуктами или харбинскому салату. Молодой человек предложил взять не только овощи, но и суп с морепродуктами или мясо в кисло-сладком соусе. Девушка ответила, что если они возьмут мясо в кисло-сладком соусе, то к нему подойдут только суп с помидорами и салат из одуванчиков (потому что харбинский салат кислый, а суп с помидорами диетический). Оказалось, что найдется три возможных варианта заказа. Но официант сказал, что одуванчики закончились, поэтому молодые люди выбрали один оставшийся вариант. Что они заказали?

Придадим логическим переменным A, B, C такой смысл: A = “Суп с морепродуктами”, $\neg A$ = “Суп с помидорами”, B = “Салат харбинский”, $\neg B$ = “Салат из одуванчиков”, C = “Мясо в кисло-сладком соусе”, $\neg C$ = “Картошка с баклажанами”.

Запишем высказывания в виде формул:

$$\neg C \to A \vee B, \;\; A \vee C, \;\; C \to \neg A \,\&\, \neg B.$$

Чтобы определить, на каких оценках истинны все три формулы, составим их конъюнкцию и приведем ее к СДНФ с помощью равносильных преобразований:

$$(\neg C \to A \vee B) \,\&\, (A \vee C) \,\&\, (C \to \neg A \,\&\, \neg B) = (C \vee A \vee B) \,\&\, (\overline{C} \vee \overline{A} \,\&\, \overline{B}) \,\&\, (A \vee C) =$$
$$= (C \vee A \vee B) \,\&\, (A \,\&\, \overline{C} \vee \overline{A} \,\&\, \overline{B} \,\&\, C) = \overline{A} \,\&\, \overline{B} \,\&\, C \vee A \,\&\, \overline{C} \vee A \,\&\, B \,\&\, \overline{C} =$$
$$= \overline{A}\&\overline{B}\&C \vee A\&(B \vee \overline{B})\&\overline{C} \vee A\&B\&\overline{C} = \overline{A}\&\overline{B}\&C \vee A\&B\&\overline{C} \vee A\&\overline{B}\&\overline{C} \vee A\&B\&\overline{C} =$$
$$= \overline{A} \,\&\, \overline{B} \,\&\, C \vee A \,\&\, B \,\&\, \overline{C} \vee A \,\&\, \overline{B} \,\&\, \overline{C}.$$

Таким образом, формула истинна на трех оценках из восьми. Поскольку по условию задачи $B = 1$ (салат из одуванчиков выбрать не получится), то остается такой заказ: $A = 1$ (суп с морепродуктами), $B = 1$ (салат харбинский), $C = 0$ (картошка с баклажанами).

1.3.2 Задачи и упражнения

1. Запишите предложение на языке логики: а) “Полковник никому не пишет”; б) “Среди гор барахла казенного есть приятное обстоятельство”; в) “Если ты со мной, мир меняет цвет и других возможных вариантов нет”; г) «Чтоб она была серьёзной и весёлой, чтоб она была неглупой и красивой — эти качества несовместимы”; д) “Не ошибается только тот, кто ничего не делает”.
2. Составьте логическую формулу и перечислите наборы значений переменных, при которых формула принимает значение 1:
   - Тезисы доклада должны быть представлены на двух языках: на русском или белорусском и на английском.
   - Кто только работает и не веселится, из того вырастет дурак и тупица.
   - Мужик сказал — мужик сделал, из этого следует, что молчание — золото.
   - Можно учиться и подрабатывать, но не работать и подучиваться.
3. По словесной формулировке составного высказывания составьте таблицу истинности. Руководствуясь таблицей, запишите формулу, выражающую данное высказывание. Проверьте себя, вычислив значения формулы на всех возможных оценках переменных.
   - Чтобы жениться, нужно уметь готовить или хотя бы зарабатывать.
   - Если одежда не подошла и была заказана в Интернете, значит, покупатель указал не свой размер.
   - Когда ты один — не страшно, а страшно, когда ты ноль.
   - Бывает, что идет дождь и светит солнце, но не бывает, что идет дождь и на улице сухо.
4. Напишите формулу, выражающую данное словесное высказывание, с заданным количеством элементарных высказываний.
   - Летом жители Владивостока работают или отдыхают на море (4 элементарных высказывания).
   - Стыда нет — иди в мед, ума нет — иди в пед, не пройдешь ни там, ни там, в техникум (5 элементарных высказываний).
   - Чтобы успешно сдать экзамен, необходимо к нему подготовиться, чтобы подготовиться к экзамену, достаточно изучить методичку (3 элементарных высказывания).
5. Даны высказывания: A = “Ему нужен доктор”, B = “Ему нужен адвокат”, C = “С ним произошел несчастный случай”, D = “Он болен”, E = “Он ранен”. Запишите следующие выражения формулами:

- Если он болен, то ему нужен доктор, и если с ним произошел несчастный случай, то ему нужен адвокат.
- Если ему нужен доктор, то он болен или ранен.
- Если ему нужны доктор и адвокат, то с ним произошел несчастный случай.
- Ему нужны доктор и адвокат тогда и только тогда, когда он болен или ранен.
- Если он и не болен и не ранен, то ему не нужен доктор.
- Он не ранен и не болен, если с ним не произошел несчастный случай.

6. Найдите логические значения высказываний A и B, при которых выполняются равенства:
   - $(1 \to A) \to B = 0$;
   - $A \vee B \leftrightarrow \overline{A} = 1$;
   - $(A \leftrightarrow B) \leftrightarrow B = 0$;
   - $A \,\&\, B \leftrightarrow \overline{A} = 1$;
   - $(A \to B) \leftrightarrow \overline{B} \,\&\, \overline{A} = 1$;
   - $(B \to \overline{A}) \leftrightarrow A = 1$.
7. Постройте дерево выражения:
   - $(p \to q) \,\&\, (\neg r \to (q \vee (\neg p \,\&\, r)))$;
   - $\neg((q \,\&\, (p \to r)) \,\&\, (r \to q))$;
   - $\neg(s \to (\neg(p \to (q \vee \neg s))))$.
8. Определите, кто из трех студентов сдал экзамен, если известно:
   а) если сдал первый, то и второй сдал;
   б) если сдал второй, то и третий сдал;
   в) если сдал третий, то второй сдал, а первый нет.
9. На вопрос, какая погода будет завтра, синоптик ответил:
   - Если будет мороз, то снег выпадет только при пасмурной погоде.
   - Если не будет мороза, но пойдет снег, то погода будет пасмурной.
   - Не будет ни снега, ни дождя, если небо будет ясным.
   - Неверно, что если не будет мороза, то для выпадения снега или дождя достаточно наличие пасмурного неба.

   Какую погоду предсказал синоптик?
10. Антон, Борис и Вадим подозреваются в угоне автомобиля. На допросе они показали следующее:
    Антон: Борис лжет.
    Борис: Вадим лжет.
    Вадим: Антон и Борис оба лгут.
    Допустим, что виновный лжет, а невиновный говорит правду. Кто из троих виновен в угоне автомобиля?
11. Приведите формулу к ДНФ и КНФ:
    - $\neg((X \to Y) \,\&\, (Y \to X))$;
    - $X \vee Y \to X \,\&\, Y$;
    - $X \,\&\, Y \to \neg(X \vee Y)$.

    Приведите также формулу к СДНФ и СКНФ, не строя таблицу истинности.
12. Даны высказывания: A = “Иван не может быть хорошим студентом, если неверно, что он способный и его родители помогают ему”, B = “Иван хороший студент тогда и только тогда, когда ему помогают родители”. Является ли B логическим следствием A?
13. Выясните, выполняются ли логические следствия:
    - $A \to B,\ C \to D,\ \overline{A} \vee \overline{C} \models \overline{B} \vee \overline{D}$;
    - $A \to B,\ C \to D,\ \overline{B} \,\&\, \overline{D} \models \overline{A} \,\&\, \overline{B}$;
    - $(A \vee B) \to (C \,\&\, D),\ \overline{C} \models \overline{A} \,\&\, \overline{B}$;

– $(A \vee B) \to (C \to D)$, $\overline{A}\,\&\,\overline{B} \models C\,\&\,\overline{D}$;
– $A \to B$, $C \to D$, $A \vee D \models B \vee C$;
– $(A\,\&\,B) \to C$, $\overline{C} \models \overline{B}$.

14. Решите логическую задачу двумя способами: построив таблицу истинности и преобразовав формулу.
Было совершено ограбление. Мегрэ сообщили, что подозреваются трое бродяг: Луи, Франсуа и Этьен. Бродяги дали следующие показания.
Луи: Чтобы обвинить меня, достаточно доказать, что Франсуа участвует в ограблении только тогда, когда в нем участвует Этьен, но я невиновен.
Франсуа: Если Луи невиновен, то, чтобы обвинить меня, достаточно признать Этьена тоже невиновным. Но Этьен виновен тогда и только тогда, когда виновен Луи. А если Этьен виновен, то я невиновен.
Этьен: Виновен либо я, либо Франсуа и Луи.
Мегрэ знал, что Этьен всегда лжет, а Луи и Франсуа говорят правду. Это помогло ему распутать дело. Кто был причастен к ограблению?

Индивидуальное домашнее задание № 1[2]

Выполните 4 задания, соответствующие вашему варианту.

Задание 1. Дайте исчерпывающий ответ на поставленный вопрос, построив таблицу истинности.

1. Следствием установлено:
1) A виновен или B виновен, или C виновен;
2) если A виновен, то и C виновен;
3) если C невиновен, то и B невиновен.
Виновен ли C?
2. Следствием установлено:
1) если A виновен, то виновен и хотя бы один из B, C;
2) если C не виновен, то и A невиновен;
3) если B виновен, то и C виновен.
Виновен ли C?
3. Следствием установлено:
1) если A виновен, то виновны также и B, и C;
2) если C виновен, то хотя бы один из A и B невиновен.
Виновен ли A?
4. Следствием установлено:
1) если A невиновен или B виновен, то C виновен;
2) если A невиновен, то C невиновен.
Виновен ли A?
5. Если завтра будет холодно (A), то я надену теплую куртку (B), если рукав будет починен (C). Завтра будет холодно, а рукав не будет починен. Следует ли отсюда, что я не надену теплую куртку?
6. Андрей или переутомился (A), или болен (B). Если он переутомился, он раздражается (C). Он не раздражается. Следует ли отсюда, что он не болен?

[2]Фролов И.С. Задачи по математической логике. — Самара, 2000.
Назиев А.Х., Моисеев С.А. Математическая логика: задачник-практикум. — Рязань, 2011.
Методические указания к лабораторным работам по дисциплине «Математическая логика и теория алгоритмов». — Чебоксары, 2019.
Беккер И.А. Логика и теория алгоритмов: методические рекомендации к лабораторным работам. — Могилев, 2021.
Попова Л.А. Математическая логика: методические рекомендации по выполнению расчетной работы. — Рубцовск, 2021.

7. Если цех II не будет участвовать в выпуске нового образца продукции, то не будет участвовать и цех I. Если же цех II будет участвовать в выпуске нового образца, то в этой работе непременно должны быть задействованы цеха I и III. Необходимо ли участие цеха III, если в выпуске нового образца будет участвовать цех I?
8. Если 2 — простое число (A), то 2 — наименьшее простое число (B). Если 2 — наименьшее простое число, то 1 не является простым числом (C). Число 1 не является простым числом. Следует ли отсюда, что 2 — наименьшее простое число? Следует ли отсюда, что 2 — простое число?
9. Следствием установлено:
   1) если B виновен, то и C виновен;
   2) кто-то из пары A, C виновен, но не оба вместе;
   3) если C невиновен, то и A невиновен.
   Виновен ли C? Можно ли достоверно утверждать, что он действовал в одиночку?
10. Подозреваемые A, B, C дали следующие показания:
    A: если B виновен, то C невиновен;
    B: если A виновен, то и C виновен;
    C: A виновен.
    Следствие установило, что: а) если A сказал правду, то B соврал; б) показания C ложны. Виновен ли B? Мог ли он действовать в одиночку?

Задание 2. Пользуясь методом от противного, определите, являются ли формулы тавтологиями:

1) $(A \to B) \to ((C \to B) \to (A \,\&\, C \to B))$;
2) $(A \to B) \to ((B \to C) \to (A \to C))$;
3) $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$;
4) $(A \to C) \to (A \to B \vee C)$;
5) $(B \to C) \to (B \to (A \to C))$;
6) $((B \to A) \to C) \to (A \to C)$;
7) $(A \vee B \to C) \to (A \to C) \vee (B \to C)$;
8) $(A \to (B \to C)) \to (A \vee B \to C)$;
9) $(A \to (B \to C)) \to (B \to (A \to C))$;
10) $(B \to C) \to (A \vee C \to (B \to C))$.

Задание 3. Приведите формулу к СДНФ и СКНФ:

1) $\overline{A} \vee \overline{C} \to A \,\&\, B \,\&\, \overline{C}$;
2) $(A \leftrightarrow C) \vee A \,\&\, B \,\&\, \overline{C}$;
3) $\overline{A} \,\&\, C \vee \overline{A \vee B} \vee C$;
4) $\overline{A} \vee C \to (B \to C)$;
5) $A \,\&\, C \vee (A \to B) \,\&\, \overline{C}$;
6) $\overline{A \to C} \vee A \,\&\, \overline{C} \,\&\, B$;
7) $\overline{A} \vee (C \vee A \,\&\, B \to C)$;
8) $A \vee \overline{B} \to B \,\&\, C$;
9) $A \,\&\, C \vee (A \vee B \to \overline{C})$;
10) $A \vee C \vee \overline{A \to B} \,\&\, C$.

Задание 4. Формализуйте высказывания, сведя задачу к проверке логического следствия. Установите или опровергните логическое следствие при помощи метода резолюций.

1. Если философ дуалист, то он не материалист. Если он не материалист, то он метафизик. Этот философ дуалист. Следовательно, он метафизик.
2. Если не было дождей или были заморозки, то урожай плохой. Известно, что урожай хороший, а заморозков не было. Значит, дожди были.

3. Если студент занимается не систематически, то он не имеет прочных знаний. Если он не имеет прочных знаний, то он не будет хорошим специалистом. Следовательно, если студент занимается не систематически, то он не будет хорошим специалистом.
4. Если человек занимается спортом, то он здоров. Если человек здоров, то он счастлив. Этот человек занимается спортом. Значит, он счастлив.
5. Если знать язык программирования, то можно написать работающую программу. Работающую программу можно также получить при наличии знакомого программиста. Овладеть языком программирования можно, обучаясь в техническом вузе. Если программа работает, то её написал выпускник технического вуза. Следовательно, если программа не работает, значит, человек, написавший её, не знает язык программирования и у него нет знакомых программистов.
6. Если Иванов работает, то он получает зарплату. Если же Иванов учится, то он получает стипендию. Но Иванов не получает зарплату или не получает стипендию. Следовательно, он не работает или не учится.
7. Если лес дешевый или транспорт в дефиците, то либо растут рубки, либо уменьшаются посадки. Если растут рубки и уменьшаются посадки, то транспорт в дефиците. Известно, что лес дешев и транспорт не в дефиците. Следовательно, рубки растут тогда и только тогда, когда посадки не уменьшаются.
8. На двери деканата злоумышленники масляной краской нарисовали несколько карикатур на преподавателей. Подозрение пало на известных хулиганов и вольнодумцев Пашу и Сашу. Кроме того, обнаружились три свидетеля, которые заявили:
   Первый: Это они сделали вместе.
   Второй: Рисовал на двери только Саша, а Паша в этом не участвовал.
   Третий: Если Паша рисовал на двери, то Саша тоже принимал в этом участие.
   Известно, что все свидетели лгали, то есть говорили прямо противоположное тому, что было на самом деле. Виноват ли Паша? Кто виноват?
9. Если конгресс отказывается принять новые законы, то забастовка не будет окончена, кроме, может быть, случая, когда она длится более года и президент фирмы уйдет в отставку. Допустим, что конгресс отказывается действовать, забастовка оканчивается и президент фирмы не уходит. Длилась ли забастовка более года?
10. Если Сергей интересуется логикой, то он посещает лекции по дискретной математике и не пропускает семинарские занятия. Если Сергей посещает лекции, то он пропускает семинарские занятия, и наоборот. Верно ли, что Сергей не интересуется логикой?

1.3.3 Лабораторные работы

Все материалы к лабораторным работам доступны по ссылке https://github.com/lapkin25/comp-logic

Лабораторная работа № 1 “Таблицы истинности”

Во-первых, вам необходимо вывести таблицу истинности бинарных логических операций: конъюнкции &, дизъюнкции $\vee$, импликации $\to$, эквиваленции $\leftrightarrow$, а также еще двух любых других булевых функций от двух переменных на ваш выбор.

Пример вывода:

```
Функция: конъюнкция
Таблица истинности:
+---+---+---------+
| A | B | f(A, B) |
+---+---+---------+
| 0 | 0 |    0    |
| 0 | 1 |    0    |
| 1 | 0 |    0    |
| 1 | 1 |    1    |
+---+---+---------+
```

Во-вторых, требуется выбрать любую равносильность от одной переменной, еще одну равносильность от двух переменных и еще одну равносильность от трех переменных; их необходимо проверить, построив таблицу истинности для левой и правой частей логического тождества.

Пример вывода:

```
Равносильность: закон де Моргана
Таблица истинности:
+---+---+---------+---------+
| A | B | f(A, B) | g(A, B) |
+---+---+---------+---------+
| 0 | 0 |    1    |    1    |
| 0 | 1 |    1    |    1    |
| 1 | 0 |    1    |    1    |
| 1 | 1 |    0    |    0    |
+---+---+---------+---------+
```

Чтобы сократить повторяющийся код, можно реализовать функцию со следующей сигнатурой:

```
def print_truth_table(f, f_name=""):
    print("Таблица истинности для функции", f_name)
    # вставьте сюда свой код
    print(1, 1, "->", f(1, 1))
```

Лабораторная работа № 2 “Дизъюнктивные нормальные формы”

Вам дана программа, которая выводит таблицу истинности заданной булевой функции трех переменных, а также строит ее СДНФ.

Примените эту программу, задав булеву функцию таблицей значений. Еще раз примените программу, задав булеву функцию с помощью вычисления заданной формулы, записанной на языке программирования Python.

Протестируйте программу. Для этого возьмите СДНФ, полученную с помощью программы, и введите ее в качестве входных данных. Построенная таблица истинности должна совпасть с исходной.

Лабораторная работа № 3 "Логическое следствие"

Иногда можно утверждать, что некоторые комбинации значений определенных высказываний неосуществимы. Это утверждение можно сформулировать в форме логического следствия.

Например, не бывает такой ситуации, что человек учится в университете ($A = 1$), не сдал сессию ($B = 0$) и был переведен на следующий курс ($C = 1$). Указанная комбинация значений логических переменных невозможна, что означает истинность формулы $\neg(A \,\&\, \neg B \,\&\, C)$. Упростим ее, пользуясь законом де Моргана и формулой замены импликации:

$$\neg(A \,\&\, \neg B \,\&\, C) = \neg A \vee B \vee \neg C = B \vee (\neg A \vee \neg C) = \neg B \to (\neg A \vee \neg C).$$

Другими словами, если человек не сдал сессию, то он не учится в университете или не был переведен на следующий курс.

Отметим, что некоторые высказывания, которые мы обозначили в формулах как элементарные, в свою очередь могут зависеть от других логических переменных. В этом случае логическое следствие возникает естественным образом без дополнительных формул. Например, F = "Все три выключателя выключены", G = "Все три выключателя включены". Тогда справедливо логическое следствие $F \models \neg G$ или, если ввести функции $F = F(A_1, A_2, A_3)$, $G = G(A_1, A_2, A_3)$, получим $\neg A_1 \,\&\, \neg A_2 \,\&\, \neg A_3 \models \neg(A_1 \,\&\, A_2 \,\&\, A_3)$. Запись со знаком $\models$ читается так: если значения переменных таковы, что справедливо то, что слева, то тогда справедливо то, что справа.

По данным обследования пациентов процедурой ОФЭКТ необходимо выявить, какие сочетания (пары и тройки) факторов являются определяющими для наличия или отсутствия заболевания. Данные по 267 пациентам включают 22 входных признака (`F1`, `F2`, ..., `F22`) и один выходной признак (`target`). Все признаки бинарные. Каждый входной признак получен методами машинного обучения по томографическим снимкам и обозначает выявление патологии (1 или 0). Выходной признак обозначает наличие заболевания (1 или 0). Данные заданы в формате TSV (значения, разделенные символами табуляции).

Вам дана заготовка решения с реализацией поиска пар входных признаков, взаимосвязанных между собой. Программа выявила отсутствие однозначных взаимосвязей. Необходимо:

1) реализовать поиск однозначных взаимосвязей между тройками входных признаков в форме логического следствия;
2) реализовать поиск однозначных взаимосвязей между одним входным признаком и выходным признаком;
3) реализовать поиск однозначных взаимосвязей между парами входных признаков и выходным признаком.

Например, если программа выявила отсутствие комбинаций значений признаков `F1 = 1`, `F4 = 1`, `Y = 0`, то это можно сформулировать в виде логической формулы:

$$\neg(F_1 \,\&\, F_4 \,\&\, \neg Y) = \neg(F_1 \,\&\, F_4) \vee Y = (F_1 \,\&\, F_4) \to Y.$$

Таким образом, можно записать логическое следствие: $F_1, F_4 \models Y$. Другими словами, сочетание 1-го и 4-го признаков существенно для постановки диагноза.

Ваша программа должна вывести найденные закономерности в таком виде:

```
F1, F4 -> Y
F15, F18 -> F21
```

```
~F2, F18 -> Y
```

Символ ~ обозначает отрицание.

Лабораторная работа № 4 “Метод резолюций”

Вам дана программа, реализующая метод резолюций. Программа берет заданную КНФ и затем в цикле находит пары резольвирующих дизъюнктов и добавляет к множеству дизъюнктов их резольвенту.

Структура данных “КНФ” в этой программе основана на представлении каждого множителя КНФ в виде списка из n значений $-1, 0, 1$, где 1 на i-м месте означает наличие в дизъюнкте литерала x_i, -1 на i-м месте означает наличие в дизъюнкте литерала $\neg x_i$ и 0 на i-м месте означает отсутствие в дизъюнкте переменной x_i.

Например, входные данные могут быть заданы таким образом: `cnf = [[1, 0, -1], [0, 1, 1], [0, -1, 0], [-1, 0, 0]]`, что означает формулу $(x \vee \overline{z}) \,\&\, (y \vee z) \,\&\, \overline{y} \,\&\, \overline{x}$.

Пример вывода программы:

```
Вход: (A + B + ∼C)(B + C)(∼B)(∼A)
Логические следствия:
A + B
A
0
Результат: невыполнима
```

Примените эту программу к примеру 1.7. Поскольку в этом примере 4 переменных, а программа работает только с тремя, то запишите КНФ дважды, рассмотрев два случая: положите переменную D равной сначала 0, а затем 1.

Приведите свой пример, в котором формула невыполнима, а вывод пустой резольвенты достаточно длинный.

Глава 2

Логика предикатов

Плохо, когда ты один,
А других множество!
Пустое...

2.1 Множества

Теория множеств служит не только одним из основных средств выражения математической мысли, но и способом формализации предметных областей в информатике. Абстракция множества подразумевает некоторую совокупность объектов, объединенных некоторым общим свойством. С помощью множеств вводят понятия функции, бинарного и n-арного отношения, графа.

Допустим, вы собираетесь посетить супермаркет, чтобы купить товары по списку. Список — это *множество* товаров. Чтобы как можно быстрее найти нужные товары, вы собираетесь воспользоваться картой торгового зала, на которой отмечены категории товаров. Вы знаете, какие товары к каким категориям относятся. Можно сказать, что у вас есть *отношение* между элементами двух множеств — множество пар (`товар`, `категория`), причем каждому товару соответствует ровно одна категория. Значит, `категория` — это *функция*: `категория` = `категория`(`товар`). Между множеством категорий товаров и множеством пунктов торгового зала установлено соответствие: вам известно, в каких пунктах может оказаться каждая категория. Таким образом, вам задана таблица (или отношение) `КатегорияПункт`, в которой перечислены пары (`категория`, `пункт`). Вы знаете также координаты каждого пункта и, значит, можете вычислить кратчайшее расстояние между любыми двумя пунктами. Пункты образуют *граф* — это множество плюс отношение между его элементами.

Итак, имея на входе список товаров, можно найти *образ* этого множества — множество интересующих категорий товаров. А зная множество категорий, мы узнаем множество пунктов из таблицы `КатегорияПункт`. Остается найти кратчайший маршрут, проходящий через все выбранные пункты. Это классическая задача на графах, которая при небольшом числе вершин может быть решена перебором всех возможных порядков, в которых вы посетите выбранные пункты. Для этого применяется алгоритм генерации *перестановок*.

2.2 Высказывания с кванторами

В алгебре высказываний буквы обозначали элементарные высказывания, каждое из которых истинно либо ложно. Алгебра предикатов позволяет учесть структуру элементарных высказываний. Теперь элементарные высказывания будут относиться к совокупностям объектов, относительно которых утверждается наличие или отсутствие у них определенных свойств.

Рассмотрим пример высказывания: “У кошки родились несколько котят”. Обозначим через K множество котят, находящихся в определенном месте. Каждый элемент x множе-

ства K — это котенок, то есть для этого x выполняется свойство $\texttt{Kitten}(x)$. А также про всех x мы знаем истинность отношения $\texttt{Born}(x, m)$, где m — конкретная кошка.

Независимо от количества котят мы можем записать утверждение о том, что все эти котята родились у конкретной кошки, с помощью формулы

$$\texttt{Cat}(m) \,\&\, \forall x \; (\texttt{Kitten}(x) \,\&\, \texttt{Born}(x, m)) \,.$$

В случае, когда множество K состоит из элементов $\{a, b, c\}$, эта формула означает то же самое, что и формула

$$\texttt{Cat}(m) \,\&\, \texttt{Kitten}(a) \,\&\, \texttt{Kitten}(b) \,\&\, \texttt{Kitten}(c) \,\&\, \texttt{Born}(a, m) \,\&\, \texttt{Born}(b, m) \,\&\, \texttt{Born}(c, m).$$

А как записать, что кто-то из котят хочет есть или пить? Для этого вместо квантора общности $\forall$ понадобится квантор существования $\exists$:

$$\exists x \, \big(\texttt{Kitten}(x) \,\&\, (\texttt{Hungry}(x) \vee \texttt{Thirsty}(x))\big).$$

Полезно уметь находить отрицания выражений с кванторами. Например, чтобы сказать, что никто из котят не хочет есть или пить, навесим на формулу отрицание и воспользуемся законом двойственности:

$$\neg\exists x \, \big(\texttt{Kitten}(x) \,\&\, (\texttt{Hungry}(x) \vee \texttt{Thirsty}(x))\big) = \\ = \forall x \; (\neg\texttt{Kitten}(x) \vee (\neg\texttt{Hungry}(x) \,\&\, \neg\texttt{Thirsty}(x))) \,.$$

Упражнение. Прочитайте полученную формулу. Насколько точно она соответствует словесной формулировке?

Как с помощью кванторов выразить такое высказывание: “Всем студентам необходим отдых”? Квантор общности будет пробегать всех людей, и нужно как-то отфильтровать только студентов. Для этого запишем формулу, истинную для людей, не являющихся студентами. А для студентов эта формула будет отвечать на вопрос, нужен ли этому человеку отдых.

Введем элементарные высказывания $\texttt{Student}(x)$, $\texttt{NeedsRest}(x)$ и запишем формулу

$$\forall x \; (\texttt{Student}(x) \to \texttt{NeedsRest}(x)) \,.$$

Выразим на языке логики предикатов высказывание “Некоторые птицы умеют летать”. Если записать под знаком квантора существования импликацию, то мы получим высказывание, которое точно будет истинным, если посылка импликации хотя бы раз обращается в 0. А нам нужна формула, которая станет истинной, только если найдется существо, которое одновременно и птица, и умеет летать:

$$\exists x \; (\texttt{Bird}(x) \,\&\, \texttt{CanFly}(x)) \,.$$

С другими примерами формализации подобных высказываний можно ознакомиться в [4].

2.3 Алгебраические системы

Алгебраической системой $\mathcal{A}$ называется множество (*носитель* алгебраической системы $\mathcal{A}$), на котором определены операции и предикаты. Совокупность символов, обозначающих операции и предикаты, называется *сигнатурой* алгебраической системы $\mathcal{A}$.

Рассмотрим несколько примеров алгебраических систем. Числовое множество $\mathbb{R}$ вместе с арифметическими операциями $+$, $-$, $\cdot$ и предикатом $<$ образует алгебраическую систему $\langle \mathbb{R}; +, -, \cdot, < \rangle$. Если добавить к операциям операцию деления, то такое множество не будет алгебраической системой: деление на 0 не определено. Но если исключить из $\mathbb{R}$ число 0 и оставить только умножение и деление, то получится алгебраическая система $\langle \mathbb{R} \backslash \{0\}; \cdot, /; < \rangle$.

Пусть множество (непустых) последовательностей символов русского алфавита — это возможные хештеги. На этом множестве определим двухместную операцию и одноместный предикат. Операция конкатенации двух последовательностей символов — это приписывание к одной последовательности справа всех символов другой последовательности. Одноместный предикат принимает истинное значение на словах русского языка и ложное на всех других последовательностях.

Обозначим всё множество возможных хештегов через A, а множество русских слов — через B. Образует ли множество B алгебраическую систему относительно операции конкатенации $+$? Нет, потому что если склеить два слова, то обычно получается строка, которая не является словом.

Подсистемой системы $\mathcal{A}$, порожденной множеством, называется минимальная алгебраическая система, включающая в себя это множество и содержащаяся в $\mathcal{A}$.

В нашем примере подсистемой алгебраической системы $\langle A, + \rangle$, порожденной множеством B, будет новая алгебраическая система $\langle A(B), + \rangle$, где множество $A(B)$ состоит только из склеенных русских слов. Например, `явуниверситете` принадлежит $A(B)$, но `фывафыва` не принадлежит. Таким образом, здесь $A(B)$ — это множество всех осмысленных хештегов.

Пусть A — это множество городов России. Определим на этом множестве унарную операцию f. Пусть $f(x)$ — это административный центр того региона, в котором находится город x. Можно определить предикат $P(x)$, принимающий значение 1 для тех городов x, которые являются административными центрами, и 0 для всех остальных городов.

Пусть $B = \{\texttt{Находка}, \texttt{Уссурийск}\}$. Тогда $P(\texttt{Владивосток}) = 1$, $P(\texttt{Уссурийск}) = 0$, $f(\texttt{Уссурийск}) = \texttt{Владивосток}$. В подсистему алгебраической системы $\langle A; f; P \rangle$, порожденную множеством B, войдут города: Находка, Уссурийск, Владивосток.

Пусть A — множество студентов в аудитории. На этом множестве можно определить двуместный предикат R. Пусть $R(x, y) = 1$, если студенты x и y сидят рядом, и $R(x, y) = 0$ — иначе. $\langle A; R \rangle$ — это алгебраическая система.

Теорема (о подсистеме, порожденной множеством). Рассмотрим алгебраическую систему $\mathcal{A}$ с носителем A, алгебраическими операциями Func и предикатами Pred. Пусть B — непустое подмножество A. Тогда можно построить, притом единственным образом, такую алгебраическую систему, которая: а) содержит в себе множество B; б) содержится в любой другой подсистеме алгебраической системы $\mathcal{A}$, содержащей множество B.

Доказательство. Проведем итерационный процесс. Обозначим $B_0 = B$, далее построим множество B_{k+1} с помощью применения операций алгебраической системы $\mathcal{A}$ к всевозможным наборам аргументов из множества B_k для $k = 1, 2, \ldots$ и так далее до бесконечности. В результате объединения всех множеств B_k получится носитель подсистемы, порожденной множеством B.

1. Это множество, обозначаемое $A(B)$, содержит в себе множество B.

2. Возьмем любое множество C, которое, во-первых, содержит в себе все элементы множества B и, во-вторых, замкнуто относительно операций из Func. Это значит, что в результате применения операций к любым элементам множества C мы не выйдем за его пределы. Нетрудно видеть, что все элементы множества $A(B)$ содержатся в множестве C. На самом деле, формируя всевозможные алгебраические выражения с операциями из Func и аргументами из множества B и вычисляя значения этих выражений, мы полу-

чим множество $A(B)$. По условию результат применения операций из Func к элементам множества B принадлежит множеству C.

3. Множество $A(B)$ замкнуто относительно операций из Func, а потому построенная конструкция действительно является алгебраической системой.

Упражнение. Почему множество $A(B)$ замкнуто относительно операций из Func?

Итак, чтобы найти подсистему алгебраической системы $\mathcal{A}$, порожденную множеством X, необходимо указать такое множество $A(X)$, что:

1) $X \subset A(X)$;
2) $A(X)$ замкнуто относительно операций алгебраической системы $\mathcal{A}$;
3) Все элементы множества $A(X)$ можно выразить через операции из Func и элементы множества X.

2.4 Формулы логики предикатов

С алгебраическими выражениями мы знакомы из школьной алгебры, где использовались такие операции, как сложение, вычитание, умножение, деление, логарифмирование и т.д. Можно обобщить понятие выражения на алгебраические операции с любым количеством аргументов. Алгебраическое выражение еще называют *термом*:

- если φ — предметная переменная, то φ — терм;
- если φ — предметная константа, то φ — терм;
- если $\varphi_1, \varphi_2, \ldots, \varphi_k$ — термы и f — k-местная алгебраическая операция, то $f(\varphi_1, \varphi_2, \ldots, \varphi_k)$ — терм;
- других термов нет.

Если к термам применить предикат, то получится *атомарная формула*. Если $t_1, t_2, \ldots, t_m$ — термы, а P — m-местный предикат, то $P(t_1, t_2, \ldots, t_m)$ — атомарная формула. Таким образом, атомарная формула имеет логический тип.

Формулы логики предикатов строятся по аналогии с формулами алгебры высказываний:

- если φ — атомарная формула, то φ — формула;
- если φ и ψ — формулы, то $\neg\varphi$, $(\varphi \,\&\, \psi)$, $(\varphi \vee \psi)$, $(\varphi \to \psi)$ — формулы;
- если φ — формула, в которой переменная x не стоит под знаком квантора, то $\forall x\, \varphi$ и $\exists x\, \varphi$ — формулы.

Приоритет логических операций и соглашения о расстановке скобок здесь сохраняются, при этом кванторы имеют наивысший приоритет.

Вхождение переменной в формулу называется *связанным*, если данная переменная находится под знаком квантора, иначе вхождение называется *свободным*.

Интерпретацией языка в алгебраической системе называется функция, которая каждому знаку k-местной алгебраической операции ставит в соответствие конкретную функцию k аргументов и каждому знаку m-местного предиката ставит в соответствие конкретный m-местный предикат.

Оценкой называется функция, указывающая каждой предметной переменной ее значение.

Рассмотрим формулу логики предикатов $\varphi(x_1, x_2, \ldots, x_n)$, зависящую от n свободных переменных. Скажем, что формула φ истинна в интерпретации $\mathcal{I}$ на оценке v, если результат вычисления этой формулы при значениях переменных, указываемых оценкой, равен 1. Обозначение: $\mathcal{I} \models \varphi(a_1, a_2, \ldots, a_n)$, где $v(x_1) = a_1, \ldots, v(x_n) = a_n$.

Формула $\varphi(x_1, x_2, \ldots, x_n)$ называется *общезначимой*, если она истинна в любой интерпретации и на любой оценке.

Говорят, что формула $\varphi(x_1, x_2, \ldots, x_n)$ *выполнима*, если существуют интерпретация $\mathcal{I}$ и оценка v, для которых $\mathcal{I} \models \varphi(a_1, a_2, \ldots, a_n)$.

Пусть A — множество книг в библиотеке. Определим унарную алгебраическую операцию next, сопоставляющую каждой книге x книгу $\text{next}(x)$, которая следует за ней по списку. Определим двухместный предикат $R(x, y)$, равный 1, если книги x и y относятся к одной тематической категории, и одноместный предикат $P(x)$, равный 1, если книга x относится к компьютерным наукам.

Следующая формула истинна в указанной интерпретации:

$$\forall x\, \forall y\ (R(x, y)\ \&\ P(x) \to P(y))\,.$$

В этом случае говорят, что алгебраическая система $\langle A; \text{next}; R, P\rangle$ — модель этой формулы. Это означает, что алгебраическая система подходит под это описание.

Следующая формула также истинна:

$$\exists x\, \neg R(x, \text{next}(x)).$$

Первая формула утверждает, что какие бы мы ни взяли две книги x и y, которые относятся к одной тематической категории, если первая из них относится к компьютерным наукам, то и вторая тоже. Вторая формула говорит, что найдется книга (хотя бы одна), такая, что следующая за ней относится к другой тематической категории.

Как сказать, что любые две книги по компьютерным наукам относятся к одной тематической категории?

$$\forall x\, \forall y\ (P(x)\ \&\ P(y) \to R(x, y))\,.$$

2.5 Основные равносильности

Определим логическое следствие: $\varphi \models \psi$, если в любой алгебраической системе $\mathcal{A}$ истинность φ в $\mathcal{A}$ влечет истинность ψ в $\mathcal{A}$.

Рассмотрим две формулы логики предикатов. Формулы называются *равносильными*, если при любой интерпретации предикатных символов и символов операций, входящих в эти формулы, в произвольной алгебраической системе $\mathcal{A}$ и при всяком наборе значений входящих в них предметных переменных эти формулы принимают одинаковые значения истинности. Другими словами, истинностные значения равносильных формул совпадают при любой интерпретации в любой алгебраической системе и на любой оценке.

Заметим, что равносильность $\varphi = \psi$ выполняется тогда и только тогда, когда $\varphi \models \psi$ и $\psi \models \varphi$.

Ниже $\varphi(x)$ или $\varphi(x,y)$ обозначает формулу, содержащую свободные вхождения указанных переменных.

1. Закон двойственности:

$$\boxed{\neg\forall x\, \varphi(x) = \exists x\, \neg\varphi(x)}$$

   Чтобы проверить эту равносильность, достаточно показать, что при любой интерпретации истинностные значения левой и правой частей совпадают. На самом деле, когда левая часть истинна, неверно, что для всех x формула $\varphi(x)$ истинна. Значит, переменной x можно придать такое значение, при котором $\varphi(x)$ окажется ложной, следовательно, правая часть истинна. И наоборот, допустим, что правая часть истинна. В этом случае можно указать некоторое значение x, при котором $\varphi(x) = 0$. Но тогда $\forall x\, \varphi(x) = 0$, а значит, левая часть равносильности истинна.
2. Закон двойственности:

$$\boxed{\neg\exists x\, \varphi(x) = \forall x\, \neg\varphi(x)}$$

Распишите рассуждения в качестве упражнения.

3. Вынесение логической константы за знак квантора:

$$\boxed{\forall x\, \varphi(x, y)\ \&\ \psi(y) = \forall x\, (\varphi(x, y)\ \&\ \psi(y))}$$

В этой формуле y — свободная переменная, которая станет константой при задании оценки. Пусть $y = a$. Тогда $\psi(a)$ — логическая константа. В случае, когда $\psi(a) = 0$, обе части равносильности равны 0. Если $\psi(a) = 1$, то левая и правая части равносильны $\forall x\, \varphi(x, y)$.

4. Вынесение логической константы за знак квантора:

$$\boxed{\exists x\, \varphi(x, y) \vee \psi(y) = \exists x\, (\varphi(x, y) \vee \psi(y))}$$

Пусть $y = a$, тогда $\psi(a)$ — логическая константа. Если $\psi(a) = 1$, то обе части равносильности равны 1. В случае, когда $\psi(a) = 0$, левая и правая части равносильны $\exists x\, \varphi(x, y)$.

5. Вынесение логической константы за знак квантора:

$$\boxed{\exists x\, \varphi(x, y)\ \&\ \psi(y) = \exists x\, (\varphi(x, y)\ \&\ \psi(y))}$$

6. Вынесение логической константы за знак квантора:

$$\boxed{\forall x\, \varphi(x, y) \vee \psi(y) = \forall x\, (\varphi(x, y) \vee \psi(y))}$$

7. Преобразование конъюнкции кванторов общности:

$$\boxed{\forall x\, \varphi(x)\ \&\ \forall x\, \psi(x) = \forall x\, (\varphi(x)\ \&\ \psi(x))}$$

Истинность левой части означает тождественную истинность $\varphi(x)$ и $\psi(x)$. Истинность правой части означает то же самое.

8. Преобразование дизъюнкции кванторов существования:

$$\boxed{\exists x\, \varphi(x) \vee \exists x\, \psi(x) = \exists x\, (\varphi(x) \vee \psi(x))}$$

Ложность левой части означает тождественную ложность $\varphi(x)$ и $\psi(x)$. Ложность правой части означает то же самое.

9. Перестановка кванторов общности:

$$\boxed{\forall x\, \forall y\, \varphi(x, y) = \forall y\, \forall x\, \varphi(x, y)}$$

10. Перестановка кванторов существования:

$$\boxed{\exists x\, \exists y\, \varphi(x, y) = \exists y\, \exists x\, \varphi(x, y)}$$

2.6 Нормальные формы

С помощью основных равносильностей формулу можно привести к *предваренной нормальной форме* (ПНФ), в которой все кванторы вынесены в начало формулы, а после них следует бескванторная часть.

Отметим, что, используя основные равносильности, можно привести любую формулу к равносильной ПНФ.

Затем, просматривая кванторы слева направо, можно исключить кванторы существования, заменив соответствующие переменные на сколемовские функции.

Для примера рассмотрим высказывание: "В любом вузе есть студенты, которые знают все предметы". Введем предикаты: $S(x, y)$ = "Студент y учится в вузе x", $K(x, y)$ = "Студент x знает предмет y". Запишем формулу логики предикатов:

$$\forall x\, \exists y\, (S(x, y)\, \&\, \forall z\, K(y, z)).$$

Преобразуем формулу к ПНФ:

$$\forall x\, \exists y\, (S(x, y)\, \&\, \forall z\, K(y, z)) = \forall x\, \exists y\, \forall z\, (S(x, y)\, \&\, K(y, z)).$$

Заменим y на сколемовскую функцию $y = f(x)$ — некоторый студент, который учится в вузе x:

$$\forall x\, \exists y\, \forall z\, (S(x, y)\, \&\, K(y, z)) = \forall x\, \forall z\, (S(x, f(x))\, \&\, K(f(x), z)).$$

Еще пример: "Бывают магазины, в которых у всех товаров есть уникальная характеристика". Введем предикаты: $S(x, y)$ = "Товар x продается в магазине y", $P(x, y)$ = "Товар x обладает характеристикой y", $Q(x)$ = "Характеристика x уникальна". Запишем формулу:

$$\exists x\, \forall y\, (S(y, x) \to \exists z\, (P(y, z)\, \&\, Q(z))).$$

Приведем формулу к ПНФ:

$$\exists x\, \forall y\, (S(y, x) \to \exists z\, (P(y, z)\, \&\, Q(z))) = \exists x\, \forall y\, (\neg S(y, x) \vee \exists z\, (P(y, z)\, \&\, Q(z))) = \\ = \exists x\, \forall y\, \exists z\, (\neg S(y, x) \vee P(y, z)\, \&\, Q(z)).$$

Теперь можно определить сколемовские функции $x = c$ и $z = f(y)$ и преобразовать формулу к сколемовской форме:

$$\forall y(\neg S(y, c) \vee P(y, f(y))\, \&\, Q(f(y))).$$

Подробнее см. [1, с. 28], [7, с. 72].

2.7 Метод резолюций

Как и в логике высказываний, метод резолюций в логике предикатов проверяет формулу на невыполнимость. Однако в логике предикатов не существует универсального алгоритма, проверяющего выполнимость или общезначимость формулы. Поэтому здесь метод резолюций не универсален[1].

Задачу проверки логического следствия $\varphi_1, \ldots, \varphi_n \models \psi$ в логике предикатов, как и в алгебре высказываний, можно свести к проверке выполнимости формулы $\varphi_1 \& \ldots \& \varphi_n \& \neg\psi$. Логическое следствие справедливо тогда и только тогда, когда эта формула невыполнима. Запись $\dfrac{\varphi, \psi}{\chi}$ означает, что $\chi = \mathrm{res}\,(\varphi, \psi)$.

В качестве примера проверим методом резолюций следующее умозаключение. "Некоторые преподаватели с уважением относятся ко всем студентам. Ни один преподаватель не уважает бездельников. Следовательно, ни один студент не является бездельником".

[1] Михальченко Г.Е. Математическая логика и теория алгоритмов: учеб. пособие. — Красноярск: Сиб. федер. ун-т, 2018.

Введем предикаты: $P(x)$ = "x — преподаватель", $S(x)$ = "x — студент", $B(x)$ = "x — бездельник", $R(x,y)$ = "x уважает y". Требуется проверить логическое следствие:

$$\exists x\, P(x)\,\&\,\forall y\,(S(y)\to R(x,y)),\ \neg\exists x(\,P(x)\,\&\,\exists y\, B(y)\,\&\,R(x,y))\ \models\ \forall x\,(S(x)\to\neg B(x)).$$

Составим формулу, выражающую нарушение логического следствия:

$$\big(\exists x\,(P(x)\&\forall y\,(S(y)\to R(x,y)))\big)\,\&\,\big(\neg\exists x(\,P(x)\&\exists y\, B(y)\&R(x,y))\big)\,\&\,\neg\big(\forall x\,(S(x)\to\neg B(x))\big)$$

и проверим, что она невыполнима.

Вначале преобразуем формулу к ПНФ. Для этого сначала избавимся от импликации и применим законы де Моргана:

$$\begin{aligned}&\big(\exists x\,(P(x)\&\forall y\,(S(y)\to R(x,y)))\big)\&\big(\neg\exists x(\,P(x)\&\exists y\,(B(y)\&R(x,y)))\big)\&\neg\big(\forall x\,(S(x)\to\neg B(x))\big)\ =\\ &=\big(\exists x\,(P(x)\&\forall y\,(\neg S(y)\vee R(x,y)))\big)\&\big(\forall x\,\neg(P(x)\&\exists y\,(B(y)\&R(x,y)))\big)\&\neg\big(\forall x\,(\neg S(x)\vee\neg B(x))\big)\ =\\ &=\big(\exists x\,(P(x)\&\forall y\,(\neg S(y)\vee R(x,y)))\big)\&\big(\forall x\,(\neg P(x)\vee\forall y\,(\neg B(y)\vee\neg R(x,y)))\big)\&\big(\exists x\,(S(x)\&B(x))\big).\end{aligned}$$

Внесем логические константы под кванторы:

$$\begin{aligned}&\big(\exists x\,(P(x)\&\forall y\,(\neg S(y)\vee R(x,y)))\big)\&\big(\forall x\,(\neg P(x)\vee\forall y\,(\neg B(y)\vee\neg R(x,y)))\big)\&\big(\exists x\,(S(x)\&B(x))\big)\ =\\ &=\exists x\,\forall y\,(P(x)\,\&\,(\neg S(y)\vee R(x,y)))\,\&\,\forall x\,\forall y\,(\neg P(x)\vee\neg B(y)\vee\neg R(x,y))\,\&\,\exists x\,(S(x)\,\&\,B(x)),\end{aligned}$$

а затем переименуем переменные, чтобы объединить кванторы:

$$\begin{aligned}&\exists x\,\forall y\,(P(x)\,\&\,(\neg S(y)\vee R(x,y)))\,\&\,\forall x\,\forall y\,(\neg P(x)\vee\neg B(y)\vee\neg R(x,y))\,\&\,\exists x\,(S(x)\,\&\,B(x))\ =\\ &=\exists x\,\forall y\,(P(x)\,\&\,(\neg S(y)\vee R(x,y)))\,\&\,\forall u\,\forall v\,(\neg P(u)\vee\neg B(v)\vee\neg R(u,v))\,\&\,\exists z\,(S(z)\,\&\,B(z))\ =\\ &=\exists x\,\forall y\,\forall u\,\forall v\,\exists z\,\big(P(x)\,\&\,(\neg S(y)\vee R(x,y))\,\&\,(\neg P(u)\vee\neg B(v)\vee\neg R(u,v))\,\&\,S(z)\,\&\,B(z)\big).\end{aligned}$$

Далее приведем формулу к сколемовской форме. Для этого введем сколемовские функции: $x=c$, $z=f(y,u,v)$. Получим формулу

$$\forall y\,\forall u\,\forall v\,\big(P(c)\&(\neg S(y)\vee R(c,y))\&(\neg P(u)\vee\neg B(v)\vee\neg R(u,v))\&S(f(y,u,v))\&B(f(y,u,v))\big).$$

Чтобы применить метод резолюций, выпишем дизъюнкты, входящие в формулу под кванторами, записанную в конъюнктивной нормальной форме:

$$P(c),\ \ \neg S(y)\vee R(c,y),\ \ \neg P(u)\vee\neg B(v)\vee\neg R(u,v),\ \ S(f(y,u,v)),\ \ B(f(y,u,v)).$$

Запишем резольвенту первого и третьего дизъюнктов, сделав предварительно замену в третьем дизъюнкте $u:=c$:

$$\frac{P(c),\ \ \neg P(c)\vee\neg B(v)\vee\neg R(c,v)}{\neg B(v)\vee\neg R(c,v)}.$$

Далее найдем резольвенту второго и четвертого дизъюнктов, сделав замену $y:=f(y,u,v)$:

$$\frac{\neg S(f(y,u,v))\vee R(c,f(y,u,v)),\ \ S(f(y,u,v))}{R(c,f(y,u,v))}.$$

Наконец, найдем резольвенту полученных результатов $\neg B(v) \vee \neg R(c, v)$ и $R(c, f(y, u, v))$, сделав замену $v := f(y, u, v)$:

$$\frac{\neg B(f(y,u,v)) \vee \neg R(c, f(y,u,v)), \quad R(c, f(y,u,v))}{\neg B(f(y,u,v))}.$$

Полученная формула вместе с пятым дизъюнктом приводит к пустой резольвенте. Тем самым доказана невыполнимость исходной формулы, которая выражает нарушение логического следствия. Значит, логическое следствие выполняется.

Подробнее с этой темой можно ознакомиться в [10, 17, 22].

2.8 Практикум

2.8.1 Типовые примеры

Пример 2.1. Дана алгебраическая система $\mathcal{A} = \langle \mathbb{N}; + \rangle$. Найдем ее подсистему, порожденную множеством $X = \{3, 72\}$.

Напомним, что множество $A(X)$ должно удовлетворять следующим свойствам:

1) $X \subset A(X)$;
2) $A(X)$ — замкнуто, то есть результат применения операции $+$ к любым элементам множества $A(X)$ остается во множестве $A(X)$;
3) все элементы $A(X)$ можно выразить через $+$ и элементы X.

Итак, начнем складывать числа 3 и 72 до тех пор, пока не получим минимальное множество, содержащее числа 3, 72 и всевозможные результаты их сложений (всевозможные значения термов, построенных на этих двух константах).

Заметим, что, во-первых, все такие суммы кратны 3, потому что 72 делится на 3. Во-вторых, всякое натуральное число, кратное 3, можно выразить через $+$ и число 3. Значит, $A(X) = 3\mathbb{N}$.

Упражнение. Найдите подсистему той же алгебраической системы, порожденную множеством $Y = \{3, -72\}$.

Замкнутый человек никогда не выходит из себя.

Пример 2.2. Установим правильность рассуждения: “Каждый студент честен. Джон нечестен. Значит, он не студент”.

Формализуем высказывания. Пусть A — множество людей. Введем предикаты: $P(x) =$ “x — студент”, $Q(x) =$ “x честен”. Предметная константа: $a \in A$ — Джон.

Каждый студент честен: $\forall x\,(P(x) \to Q(x))$. Джон нечестен: $\neg Q(a)$. Джон — не студент: $\neg P(a)$.

Установим логическое следствие:

$$\forall x\,(P(x) \to Q(x)),\ \neg Q(a) \ \models\ \neg P(a).$$

Согласно критерию логического следствия надо доказать невыполнимость следующей формулы:

$$\big(\forall x\,(P(x) \to Q(x))\big)\ \&\ \neg Q(a)\ \&\ P(a).$$

Множество дизъюнктов:

$$\neg P(x) \vee Q(x),\ \neg Q(a),\ P(a).$$

Сделаем замену $x := a$ в первой формуле, после чего применим правило резолюции к первой и второй формулам. Получим

$$\frac{\neg P(a) \vee Q(a),\ \neg Q(a)}{\neg P(a)}.$$

Остается применить правило резолюции к полученной формуле $\neg P(a)$ и третьей формуле $P(a)$. Получаем пустую резольвенту. Значит, формула невыполнима и логическое следствие имеет место.

Правда всегда находится в пути (Томаш Бурек).

Пример 2.3. Формализуем составление расписания занятий в университете.

Зададим множества: `Дисциплины`, `Группы`, `Преподаватели`, `ДниНедели`, `НомераПар` и отношения `ГруппаДисц`, `ПреподПредмет` — это входные данные. Выходными данными будет отношение `Расписание` на множествах `Дисциплины`, `Группы`, `Преподаватели`, `ДниНедели`, `НомераПар`, удовлетворяющее заданным ограничениям.

Сформулируем ограничения:

- ни у одной группы не может быть дисциплины, не указанной в отношении `ГруппаДисц`;
- ни один преподаватель не может вести дисциплину, не указанную в отношении `ПреподПредмет`;
- не может быть два занятия у одной и той же группы в одно и то же время;
- не может быть два занятия у одного и того же преподавателя в одно и то же время;
- у каждой группы должно быть ровно одно занятие в неделю по каждой дисциплине.

Упражнение. Выразите эти условия на языке логики предикатов.

2.8.2 Задачи и упражнения

1. Используя предикат равенства $x = y$ и функции произведения xy, выразите следующие предикаты на множестве натуральных чисел:
   - x делится на y;
   - z является общим делителем x и y;
   - z является наибольшим общим делителем x и y;
   - z есть общее кратное x и y;
   - z есть наименьшее общее кратное x и y;
   - x и y взаимно просты.
2. Переведите на русский язык (смысл обозначений тот же):
   - $\exists z\,(2z = x)$;
   - $\neg\exists z\,(3z = x)$;
   - $\exists y\,(yy = x)$;
   - $\forall x\,\forall y\,(\exists u\,(yu = x)\ \&\ \exists v\,(xv = y) \to (x = y))$;
   - $\forall x\,\forall y\,\forall z\,\forall t\,((yz = t)\ \&\ \exists u\,(tu = x) \to \exists v\,(yv = x)\ \&\ \exists w\,(zw = x))$.
3. Переведите на символический язык:
   - Не все птицы могут летать.
   - Если всякий разумный философ — циник и только женщины являются разумными философами, то тогда, если существуют разумные философы, некоторые из женщин — циники.
   - Либо каждый любит кого-нибудь и ни один не любит всех, либо некто любит всех и кто-то не любит никого.
4. Покажите, что следующее рассуждение нелогично:

Перья есть только у птиц. Ни одно млекопитающее не является птицей. Значит, у некоторых млекопитающих есть перья.

5. Докажите справедливость утверждения при помощи метода резолюций.
   - Каждый член парламента является либо членом палаты общин, либо членом палаты лордов.
   - Все члены палаты общин находятся в полном рассудке.
   - Ни один член парламента, носящий титул пэра, не станет участвовать в скачках на мулах.
   - Все члены палаты лордов носят титул пэра.

   Следовательно, ни один член парламента не станет участвовать в скачках на мулах, если он в полном рассудке.

Индивидуальное домашнее задание № 2[2]

Выполните 4 задания, соответствующие вашему варианту.

Задание 1. Навесьте на предикат $P(x)$ квантор $\forall$ и найдите отрицание полученного выражения. Навесьте на предикат $P(x)$ квантор $\exists$ и найдите отрицание полученного выражения. Сформулируйте словами полученные высказывания:

1) $P(x) =$ "$x \leqslant 3$";
2) $P(x) =$ "$x^2 = 16$";
3) $P(x) =$ "$\cos x = 0$";
4) $P(x) =$ "x — делитель 16";
5) $P(x) =$ "x — кратное 5";
6) $P(x) =$ "$\ln x < 0$";
7) $P(x) =$ "$\cos x > 1$";
8) $P(x) =$ "$\operatorname{ctg} x > 1$";
9) $P(x) =$ "x делится без остатка на 3";
10) $P(x) =$ "$\operatorname{ctg} x$ не существует".

Задание 2. Докажите, что формула не общезначима. Для этого приведите пример алгебраической системы, в которой формула обращается в 0:

1) $\exists x\, P(x) \,\&\, \exists x\, Q(x) \to \exists x(P(x) \,\&\, Q(x))$;
2) $\forall x\, \exists y\, R(x, y) \to \exists x\, \forall y\, R(x, y)$;
3) $\forall x\, \exists y\, (R(x, y) \leftrightarrow R(y,x))$;
4) $\exists x\, \forall y\, (R(x, y) \leftrightarrow R(y, x))$;
5) $(\forall x\, P(x) \vee \forall x\, Q(x)) \leftrightarrow (\forall x(P(x) \vee Q(x)))$;
6) $(\forall x\, P(x) \to \forall x\, Q(x)) \leftrightarrow (\forall x\, (P(x) \to Q(x)))$;
7) $(\forall x\, P(x) \to \exists x\, Q(x)) \leftrightarrow (\exists x\, (P(x) \to Q(x)))$;
8) $\neg\forall x\, \exists y\, (Q(x, y) \,\&\, \overline{Q(x, x)})$;
9) $\neg\exists x\, \forall y\, (Q(x,y) \to \forall z R(x, y, z))$;
10) $\neg\forall x\, \forall y\, \forall z\, (Q(x,y) \vee Q(y, z))$.

Задание 3. Приведите формулу к предваренной, а затем к сколемовской форме:

1) $\exists x\, A(x) \vee \forall x\, B(x) \leftrightarrow \forall x\, (A(x) \vee B(x))$;
2) $\exists x\, P(x) \vee (\forall x\, (P(x) \to Q(x)) \to \forall x\, Q(x))$;
3) $\forall x\, A(x) \vee \exists x\, B(x) \to \exists x\, B(x) \vee \exists x\, C(x)$;
4) $\forall x\, (P(x) \vee Q(x)) \to \forall x\, P(x) \vee \forall x\, Q(x)$;
5) $\forall x\, A(x) \vee \exists x\, B(x) \to \forall x\, A(x) \,\&\, \forall x\, C(x)$;
6) $\exists x\, (P(x) \vee Q(x)) \leftrightarrow \exists x\, P(x) \vee \exists x\, Q(x)$;
7) $(\forall x\, A(x) \to \forall x\, B(x)) \to \forall x\, (A(x) \to C(x))$;
8) $\forall x\, (P(x) \,\&\, Q(x)) \leftrightarrow \forall x\, P(x) \,\&\, \forall x\, Q(x)$;

[2] Попова Л.А. Математическая логика: методические рекомендации по выполнению расчетной работы. — Рубцовск, 2021.

9) $\exists x\, P(x) \vee \exists x\, Q(x) \vee \forall x\, R(x) \to \forall x\, P(x)\ \&\ \forall x\, Q(x)$;
10) $\exists x\, (P(x) \to Q(x)) \to (\exists x\, P(x) \to \exists x\, Q(x))$.

Задание 4. Решите задачу.

1. На множестве $M = \{-5,\ -4,\ -3, 2,\ -1\}$ заданы предикаты $A(x)$ = "x — отрицательное число", $B(x)$ = "x — четное число". Найдите множества истинности предикатов $A(x)$, $B(x)$ и $A(x) \to \overline{B(x)}$. Получите высказывания из данных предикатов, добавив к ним квантор общности и квантор существования, и определите их истинностные значения.
2. На множестве $M = \{0, 1, 2, 3, 4, 6, 8\}$ заданы предикаты $P_1(x,y)$ = "x делится на y", $P_2(x,y)$ = "$x^2 - y^2 > 4$". Найдите множества истинности предикатов $P_1(x,y)$, $P_2(x,y)$ и $P_1(x,y)\ \&\ P_2(x,y)$. Получите высказывания из данных предикатов, добавив к ним кванторы общности (по всем переменным) и кванторы существования, и определите их истинностные значения.
3. На множестве $M = \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6\}$ заданы предикаты $P_1(x,y)$ = "$x > y + 3$", $P_2(x,y)$ = "$x^2 + y^2 > 35$". Найдите множества истинности предикатов $P_1(x,y)$, $P_2(x,y)$ и $P_1(x,y) \vee P_2(x,y)$. Получите высказывания из данных предикатов, добавив к ним кванторы общности (по всем переменным) и кванторы существования, и определите их истинностные значения.
4. На множестве $M = \{2,\ 3,\ 4,\ 6,\ 8\}$ заданы предикаты $P_1(x,y)$ = "x делится на y", $P_2(x,y)$ = "$x + y$ делится на 6". Найдите множества истинности предикатов $P_1(x,y)$, $P_2(x,y)$ и $P_1(x,y)\ \&\ P_2(x,y)$. Получите высказывания из данных предикатов, добавив к ним кванторы общности (по всем переменным) и кванторы существования, и определите их истинностные значения.
5. На множестве $M = \{-2,\ -3,\ 4,\ 6,\ 8\}$ заданы предикаты $P_1(x,y)$ = "$x + y \leqslant 0$", $P_2(x,y)$ = "$x + y > 10$". Найдите множества истинности предикатов $P_1(x,y)$, $P_2(x,y)$ и $P_1(x,y)\ \&\ P_2(x,y)$. Получите высказывания из данных предикатов, добавив к ним кванторы общности (по всем переменным) и кванторы существования, и определите их истинностные значения.
6. На множестве $M = \{-6,\ -4,\ -2,\ 2,\ 4,\ 6,\ 8\}$ заданы предикаты $A(x)$ = "x — число, кратное 4", $B(x)$ = "$x - 4 < 0$". Найдите множества истинности предикатов $A(x)$, $B(x)$ и $A(x) \vee B(x)$. Получите высказывания из данных предикатов, добавив к ним квантор общности и квантор существования, и определите их истинностные значения.
7. На множестве $M = \{0,\ 1,\ 2,\ 8\}$ заданы предикаты $P_1(x,y)$ = "$4x = y$", $P_2(x,y)$ = "$x^2 + y^2 < 5$". Найдите множества истинности предикатов $P_1(x,y)$, $P_2(x,y)$ и $P_1(x,y) \leftrightarrow P_2(x,y)$. Получите высказывания из данных предикатов, добавив к ним кванторы общности (по всем переменным) и кванторы существования, и определите их истинностные значения.
8. На множестве $= \{1,\ 3,\ 5,\ 7,\ 9\}$ заданы предикаты $A(x)$ = "$x > 5$", $B(x)$ = "x — положительное число". Найдите множества истинности предикатов $A(x)$, $B(x)$ и $\overline{A(x)} \to B(x)$. Получите высказывания из данных предикатов, добавив к ним квантор общности и квантор существования, и определите их истинностные значения.
9. На множестве $M = \{1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8\}$ заданы предикаты $A(x)$ = "x — положительное число", $B(x)$ = "x — число, кратное 4". Найдите множества истинности предикатов $A(x)$, $B(x)$ и $B(x) \to A(x)$. Получите высказывания из данных предикатов, добавив к ним квантор общности и квантор существования, и определите их истинностные значения.
10. На множестве $M = \{5, 6, 12, 13, 14, 16, 17\}$ заданы предикаты $A(x)$ = "x — составное число", $B(x)$ = "x делится на 5". Найдите множества истинности предикатов $A(x)$, $B(x)$ и $A(x) \to \overline{B(x)}$. Получите высказывания из данных предикатов, добавив к

ним квантор общности и квантор существования, и определите их истинностные значения.

2.8.3 Лабораторные работы

Лабораторная работа № 5 “Вычисление формул с кванторами”

Файл `stud.csv` содержит результаты опроса студентов, проходившего в 2022 году в ВВГУ.

Требуется написать программу, которая ответит на вопросы:

– Правда ли, что все студенты старше 1-го курса старше 18 лет?

– Правда ли, что найдется студент, которому 21 год и который имеет стаж работы от 2 до 3 лет?

В качестве примера у вас есть заготовка, которая отвечает на вопросы:

– Правда ли, что все студенты старше 2-го курса старше 18 лет?

– Правда ли, что найдутся студенты бакалавриата, которым менее 17 лет?

Первый вопрос можно формализовать так:

$$\forall x\,((x \text{ старше 2 курса}) \to (x \text{ старше 18 лет})).$$

Второй вопрос можно формализовать так:

$$\exists x\,((x \text{ – студент бакалавриата}) \,\&\, (x \text{ моложе 17 лет})).$$

Ответ на вопрос “Правда ли, что для всех выполняется свойство” предполагает либо положительный ответ, либо отрицательный с выводом всех контрпримеров.

Ответ на вопрос “Правда ли, что существует, для которого выполняется свойство” предполагает либо отрицательный ответ, либо положительный с выводом всех, кто подходит под это свойство.

Квантор общности программируется как длинная конъюнкция (умножение) логических условий. Квантор существования программируется как длинная дизъюнкция (сложение) логических условий.

Схема вычисления $\forall x\,P(x)$:

```
q = True
for x in A:
    if not P(x):
        q = False
# q = Forall x P(x)
```

Схема вычисления $\exists x\,P(x)$:

```
q = False
for x in A:
    if P(x):
        q = True
# q = Exists x P(x)
```

Лабораторная работа № 6 “Проблема выполнимости”

Проблема выполнимости состоит в том, чтобы выяснить, является ли булева формула тождественно ложной (невыполнимой) или выполнимой. Данная проблема вообще не имеет эффективного решения, но на практике решается специальным ПО — SAT-солверами.

Рассмотрим решение трех задач, сводящихся к проблеме выполнимости: ход в игре «крестики-нолики», расстановка ферзей на шахматной доске, решение судоку.

В языке программирования Python мы воспользуемся SAT-солвером из пакета `pycosat`[3].

Булева формула представляется в стандартном формате DIMACS CNF, в котором вводится КНФ. Обратите внимание на то, что формат, используемый в `pycosat`, отличается от того формата, в котором мы вводили КНФ в лабораторной работе на метод резолюций. Здесь в каждом множителе КНФ перечисляются номера переменных (со знаком плюс, если переменная без отрицания, и со знаком минус, если переменная с отрицанием).

Первое задание — придумайте любую ситуацию в игре “крестики-нолики”, когда крестики и нолики сделали по 2 хода. Теперь ходят крестики. Смогут ли они выиграть за один ход? Введите булевы переменные и запишите формулу алгебры логики для ответа на вопрос. Затем преобразуйте формулу к КНФ и введите в SAT-солвер.

Второе задание — расставьте 8 ферзей на шахматной доске так, чтобы они не били друг друга[4]. Чтобы записать булеву формулу, потребуется перечислить все запрещенные ситуации. В результате получится КНФ.

Дополнительное задание — решение судоку[5].

Лабораторная работа № 7 “Программирование с контрактами”

Метод Флойда состоит в выделении опорных точек алгоритма, в которых состояние вычислений выражается четко на языке логики предикатов.

Ознакомьтесь с турниром в тестирующей системе CATS[6]. Обратите внимание на формулировку задач в этом турнире: каждая задача содержит ограничение на входные данные, также четкое описание формата входа и выхода и соответствия между входом и выходом — это и есть контракт, или спецификация задачи.

Теперь представьте, что вы решаете крупную задачу. Вы хотите разбить ее на небольшие кусочки и поручить их решение разным людям. Чтобы вас поняли однозначно, вы напишете контракт для каждой подзадачи — точно так же, как в CATS.

Ваша задача — распределить нагрузку в команде, чтобы решить как можно больше из предложенных задач, пользуясь логическим комментированием ваших алгоритмов по методу Флойда.

Лабораторная работа № 8 “Формализация предметной области”

Лабораторная работа выполняется в командах, в команде не более 3 человек. Выберите предметную область[7], формальную модель которой вы запишете на языке логики предикатов.

Придумайте интерактивную часть для своей программы, оформите спецификацию ввода-вывода, распределите реализацию частей информационной системы в своей команде.

---

[3]URL: https://github.com/conda/pycosat

[4]Ознакомьтесь с решением по ссылке https://huyen-nguyen.github.io/n-queens/

[5]Gasquet O., Schwarzentruber F., Strecker M. Satoulouse: the computational power of propositional logic shown to beginners // Tools for teaching logic: Third International Congress, TICTTL 2011: Proceedings. — P. 77–84.

[6]URL: https://lapkin25.github.io/test-abstracts/algo-intro1.html

[7]Игнатьева О.В. Базы данных: учебно-методическое пособие для лабораторных работ. — Ростов-на-Дону, 2017. — С. 27.

Глава 3

Формальный вывод

Что главнее,
Форма или содержание?
Оформленное содержание!

3.1 Представление знаний

Представление знаний в системах символического искусственного интеллекта осуществляется на основе правил, применяемых к фактам в процессе реализации той или иной стратегии управления для достижения определенной цели [5].

Систему правил можно рассматривать как формальную теорию, в которой исходная база фактов — это аксиомы формальной теории, правила — это дополнительные аксиомы, а цель — это теорема данной теории, доказательство которой необходимо построить. При этом в качестве правила вывода в этой теории выступает правило отделения.

Формальная теория — это структура, включающая в себя: язык теории, то есть множество всевозможных правильно построенных формул, а также множество таких формул, называемых аксиомами теории, и набор отношений между предложениями языка, называемых правилами вывода теории.

Теория должна находиться в некотором соответствии с формализуемой предметной областью. Во-первых, должна существовать такая интерпретация формул теории, при которой все выводимые формулы истинны в этой интерпретации. Во-вторых, если множество выводимых формул исчерпывает все истинные утверждения предметной области, то говорят, что теория полна. Теория, обладающая обоими свойствами, в полной мере соответствует предметной области. Таким образом, выполнением механических действий над формальными объектами обеспечивается получение новых фактов.

3.2 Исчисление высказываний

3.2.1 Язык ИВ

Построим вспомогательное исчисление, состоящее из аксиом и правил вывода, которое порождает язык исчисления высказываний (ИВ), состоящий из правильно построенных логических формул.

Вспомним, как вводилось понятие логической формулы. Под формулой мы понимали строку, состоящую из логических переменных, знаков логических операций и скобок, которую можно построить в соответствии со следующими правилами:

– базисные формулы (аксиомы) — логические переменные;

– правила вывода:

$$\frac{\varphi}{\neg\varphi}, \quad \frac{\varphi, \psi}{(\varphi \,\&\, \psi)}, \quad \frac{\varphi, \psi}{(\varphi \vee \psi)}, \quad \frac{\varphi, \psi}{(\varphi \to \psi)}.$$

Здесь φ обозначает любую строку, а дробь читается так: если строки над чертой относятся к языку, то строка под чертой непосредственно из неё выводима, а потому тоже будет относиться к языку формул.

Только что мы ввели *исчисление*, которое определяет формальный язык всех синтаксически корректных формул.

Например, строка $((A \vee \neg B) \to (A \,\&\, (C \vee B)))$ принадлежит языку формул, поскольку существует вывод из аксиом A, B, C:

$$A, B, C \vdash \neg B; \quad A, B, C \vdash (C \vee B).$$

И поскольку

$$A, B, C, (C \vee B) \vdash (A \,\&\, (C \vee B)); \quad A, B, C, \neg B \vdash (A \vee \neg B),$$

то в силу правила (Γ — множество строк, φ, ψ — строки)

$$\frac{\Gamma \vdash \varphi; \;\; \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi}$$

получаем

$$A, B, C \vdash (A \,\&\, (C \vee B)); \quad A, B, C \vdash (A \vee \neg B).$$

Отсюда следует аналогичным образом, что поскольку

$$(A \,\&\, (C \vee B)), \; (A \vee \neg B) \vdash ((A \vee \neg B) \to (A \,\&\, (C \vee B))),$$

то

$$A, B, C \vdash ((A \vee \neg B) \to (A \,\&\, (C \vee B))).$$

Выводом формулы φ в исчислении называется последовательность строк, каждая из которых является аксиомой или (непосредственно) выводима из предшествующих строк, а последняя строка — это формула φ.

В нашем примере вывод может выглядеть так:

1) A — аксиома;
2) B — аксиома;
3) C — аксиома;
4) $\neg B$ — из п. 2 по правилу введения $\neg$;
5) $(C \vee B)$ — из п. 2, 3 по правилу введения $\vee$;
6) $(A \,\&\, (C \vee B))$ — из п. 1, 5 по правилу введения $\&$;
7) $(A \vee \neg B)$ — из п. 1, 4 по правилу введения $\vee$;
8) $((A \vee \neg B) \to (A \,\&\, (C \vee B)))$ — из п. 6, 7 по правилу введения $\to$.

Выводимость в исчислении формулы φ обозначают $\vdash \varphi$.

Выводом в исчислении из множества гипотез Γ называется последовательность строк, каждая из которых является аксиомой или гипотезой (то есть содержится в Γ) или (непосредственно) выводима из предшествующих строк.

Выводимость в исчислении формулы φ из гипотез Γ обозначают $\Gamma \vdash \varphi$.

Например, если принять гипотезы $(A \to B)$ и $\neg C$, то формула $(((A \to B) \,\&\, \neg C) \vee D)$ будет выводима из гипотез:

1) $(A \to B)$ — гипотеза;
2) $\neg C$ — гипотеза;
3) $((A \to B) \,\&\, \neg C)$ — из п. 1, 2 по правилу введения $\&$;
4) D — аксиома;
5) $(((A \to B) \,\&\, \neg C) \vee D)$ — из п. 3, 4 по правилу введения $\vee$.

Поэтому можно записать $(A \to B), \neg C \vdash (((A \to B) \,\&\, \neg C) \vee D)$.

Всё множество строк, выводимых в описанном формальном исчислении, назовем *языком исчисления высказываний*. Итак, $\mathcal{L} = \{\varphi\colon \vdash \varphi\}$.

3.2.2 Аксиомы ИВ

Чтобы ввести *исчисление высказываний*, необходимо указать, из каких формул будут выводиться все прочие формулы. Эти формулы — *аксиомы* — должны интерпретироваться в алгебре высказываний как тождественно истинные. Каждая аксиома выражает свойства логических связок $\neg$, $\&$, $\vee$, $\to$.

В качестве аксиом ИВ примем множество всевозможных строк, получаемых из любой из следующих 11 схем при замене букв φ, ψ, χ на любые формулы ИВ:

1) $\varphi \to (\psi \to \varphi)$;
2) $(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))$;
3) $(\varphi \,\&\, \psi) \to \varphi$;
4) $(\varphi \,\&\, \psi) \to \psi$;
5) $\varphi \to (\psi \to (\varphi \,\&\, \psi))$;
6) $\varphi \to (\varphi \vee \psi)$;
7) $\psi \to (\varphi \vee \psi)$;
8) $(\varphi \to \chi) \to ((\psi \to \chi) \to ((\varphi \vee \psi) \to \chi))$;
9) $\neg\varphi \to (\varphi \to \psi)$;
10) $(\varphi \to \psi) \to ((\varphi \to \neg\psi) \to \neg\varphi)$;
11) $\varphi \vee \neg\varphi$.

Аксиома 1 выражает *правило введения посылки*. Например: “Раз все смертны, то если ты Сократ, то ты смертен”.

Если свойство A всегда истинно, то оно будет истинно и в частном случае, когда выполнено свойство B:

$$\boxed{A \to (B \to A)}$$

Аксиома 2 выражает *правило распределения посылки*. Например: “Допустим, нам известно, что у мужчин (A) переедание (B) приводит к ожирению (C). Отсюда можно сделать вывод о том, что если мужчины переедают ($A \to B$), то у них будет ожирение ($A \to C$)”.

Если при выполнении свойства A свойство B влечет за собой свойство C, то выполнение свойства B по крайней мере при выполнении A влечет выполнение свойства C по крайней мере при выполнении A:

$$\boxed{(A \to (B \to C)) \to ((A \to B) \to (A \to C))}$$

Аксиомы 3, 4 выражают *правило удаления конъюнкции*.

Если свойства A и B выполняются одновременно, то каждое из них выполняется по отдельности:

$$\boxed{(A \,\&\, B) \to A, \quad (A \,\&\, B) \to B}$$

Аксиома 5 выражает *правило введения конъюнкции*.

При выполнении свойств A и B выполняется их конъюнкция:

$$\boxed{A \to (B \to (A \,\&\, B))}$$

Аксиомы 6, 7 выражают *правило введения дизъюнкции*. Например: “Если ты мужчина, то ты человек (то есть мужчина или женщина)”.

**Если выполнено свойство A, то выполнено свойство «A или B».
Если выполнено свойство B, то выполнено свойство «A или B»:**

$$\boxed{A \to (A \vee B), \quad B \to (A \vee B)}$$

Аксиома 8 выражает *правило разбора случаев*, которое также называют *правилом удаления дизъюнкции*. Например: “Если ты ребенок (A), то ты учишься (C); если ты молод (B), то ты учишься (C); следовательно, если ты ребенок или молодой человек ($A \vee B$), то ты учишься (C)”.

Если свойство A влечет C и если свойство B влечет C, то выполнение хотя бы одного из свойств A или B также влечет C:

$$\boxed{(A \to C) \to ((B \to C) \to ((A \vee B) \to C))}$$

Аксиома 9 выражает *правило вывода из противоречия*. Например: “Если ты жив и мертв одновременно, то тебя не существует, поэтому про тебя можно вообразить всё, что угодно”.

Одновременное выполнение и нарушение какого-либо свойства заведомо противоречиво, что влечет истинность любого высказывания:

$$\boxed{\neg A \to (A \to B)}$$

Аксиома 10 выражает *правило рассуждения от противного*, или *правило введения отрицания*. Например: “Если ты айтишник, то ты знаешь Линукс, но ты не знаешь Линукс, значит, какой из тебя айтишник”. Или: “Если функция сработала верно, то программа завершилась корректно; даже если функция сработала верно, то программа завершилась с ошибкой — что-то с этой функцией не так”.

Если из посылки следует противоречие, то она ложна:

$$\boxed{(A \to B) \to ((A \to \neg B) \to \neg A)}$$

Аксиома 11 выражает *закон исключенного третьего*: третьего не дано.

Из любых двух противоречащих суждений одно окажется истинным:

$$\boxed{A \vee \neg A}$$

Перечисленных аксиом оказывается достаточно, чтобы построить формальную теорию, включающую все истинные формулы логики высказываний и только их.

3.2.3 Правила вывода ИВ

Единственным *правилом вывода* в исчислении высказываний является *правило отделения* (от лат. *modus ponens*). Если импликация $\varphi \to \psi$ выведена вместе со своей левой частью φ, то из этих двух формул выводима правая часть импликации ψ. Иначе говоря, от истинной импликации можно отделить истинную посылку и оставить только заключение, которое считается истинным:

$$\frac{\varphi \to \psi,\ \varphi}{\psi}.$$

Правило отделения гарантирует, что на любой оценке логических переменных, при которой истинны предпосылки, будет истинно и заключение. Это свойство обеспечивает корректность исчисления высказываний.

Теорема (корректность ИВ). Пусть Γ — множество формул, φ — формула. Если $\Gamma \vdash \varphi$, то $\Gamma \models \varphi$.

Доказательство можно найти в [2, с. 79].

Теорема (о дедукции). Если из множества формул $\Gamma \cup \{\varphi\}$ выводима формула ψ, то из множества формул Γ выводима формула $\varphi \to \psi$:

$$\Gamma, \varphi \vdash \psi \quad \Rightarrow \quad \Gamma \vdash \varphi \to \psi.$$

Доказательство. Нам дано, что $\Gamma, \varphi \vdash \psi$. Эта запись означает, что можно построить последовательность формул ИВ, каждая из которых либо аксиома ИВ, либо одна из гипотез (то есть принадлежит множеству $\Gamma \cup \{\varphi\}$), либо ее можно вывести из каких-нибудь двух предшествующих формул по правилу отделения. Обозначим эту последовательность формул через $c_1, c_2, \ldots, c_n$, в которой $c_n = \psi$, и докажем по индукции для всех $k = 1, 2, \ldots, n$ справедливость утверждения $\Gamma \vdash \varphi \to c_k$.

Рассмотрим случаи:

1. c_k — аксиома ИВ.

По аксиоме 1 формула $c_k \to (\varphi \to c_k)$ выводима как аксиома. Применяя правило отделения к формулам $c_k \to (\varphi \to c_k)$ и c_k, получаем искомую выводимость формулы $\varphi \to c_k$.

2. $c_k \in \Gamma$.

Разбирается аналогично случаю 1.

3. $c_k = \varphi$.

Выводимость формулы $\varphi \to \varphi$ доказывается как теорема (см. подразд. 3.2.5).

4. c_k получена по правилу отделения из формул c_i и $c_j = (c_i \to c_k)$.

По гипотезе индукции справедливы выводимости $\Gamma \vdash \varphi \to c_i$ и $\Gamma \vdash \varphi \to c_j$, то есть $\Gamma \vdash \varphi \to (c_i \to c_k)$.

Согласно аксиоме 2 следующая формула является аксиомой:

$$(\varphi \to (c_i \to c_k)) \to ((\varphi \to c_i) \to (\varphi \to c_k)).$$

Применяя дважды правило отделения, получаем искомую выводимость формулы $\varphi \to c_k$.

Упражнение. Выпишите вывод формулы $\varphi \to c_k$ для каждого из четырех случаев.

Из теоремы о дедукции вытекает дополнительное правило вывода, называемое *правилом введения импликации*:

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}.$$

Из обратной теоремы о дедукции следует *правило удаления импликации*:

$$\frac{\Gamma \vdash \varphi; \quad \Gamma \vdash \varphi \to \psi}{\Gamma \vdash \psi}.$$

Из аксиомы 5 получаем *правило введения конъюнкции*:

$$\frac{\Gamma \vdash \varphi; \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \,\&\, \psi}.$$

Из аксиом 3, 4 получаем *правила удаления конъюнкции*:

$$\frac{\Gamma \vdash \varphi \,\&\, \psi}{\Gamma \vdash \varphi}, \quad \frac{\Gamma \vdash \varphi \,\&\, \psi}{\Gamma \vdash \psi}.$$

Аксиомы 6, 7 позволяют сформулировать *правила введения дизъюнкции*:

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}, \quad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}.$$

Из аксиомы 8 и теоремы о дедукции следует *правило удаления дизъюнкции*, или *правило разбора случаев*:

$$\frac{\Gamma, \varphi \vdash \chi; \quad \Gamma, \psi \vdash \chi}{\Gamma, \ \varphi \vee \psi \vdash \chi}.$$

Частным случаем этого правила является *правило исчерпывающего разбора случаев*, вытекающее из аксиомы 11:

$$\frac{\Gamma, \varphi \vdash \psi; \quad \Gamma, \ \neg\varphi \vdash \psi}{\Gamma \vdash \psi}.$$

Правило введения отрицания, или *правило рассуждения от противного*, следует из аксиомы 10:

$$\frac{\Gamma, \varphi \vdash \psi; \quad \Gamma, \varphi \vdash \neg\psi}{\Gamma \vdash \neg\varphi}.$$

Правило удаления отрицания:

$$\frac{\Gamma \vdash \neg\neg\varphi}{\Gamma \vdash \varphi}.$$

Перепишем правила вывода в более простом виде.
Правило введения конъюнкции:

$$\frac{\varphi, \psi}{\varphi \,\&\, \psi}.$$

Правила удаления конъюнкции:

$$\frac{\varphi \,\&\, \psi}{\varphi}, \quad \frac{\varphi \,\&\, \psi}{\psi}.$$

Правила введения дизъюнкции:

$$\frac{\varphi}{\varphi \vee \psi}, \quad \frac{\psi}{\varphi \vee \psi}.$$

Правило разбора случаев:

$$\frac{\varphi \to \chi, \ \ \psi \to \chi}{\varphi \vee \psi \to \chi}.$$

Правило исчерпывающего разбора случаев:

$$\frac{\varphi \to \psi,\ \ \neg\varphi \to \psi}{\psi}.$$

Правило рассуждения от противного:

$$\frac{\varphi \to \psi,\ \ \varphi \to \neg\psi}{\neg\varphi}.$$

Правило удаления отрицания:

$$\frac{\neg\neg\varphi}{\varphi}.$$

Докажем еще несколько полезных правил вывода.

Правило отрицания (от лат. *modus tollens*) — это вариант правила рассуждения от противного:

$$\frac{\varphi \to \psi,\ \neg\psi}{\neg\varphi}.$$

Например: вдумчивое чтение развивает способности; способности не развиваются; следовательно, чтение не вдумчивое.

Построим вывод формулы $\neg\varphi$ из гипотез $\varphi \to \psi$ и $\neg\psi$:

1) $\varphi \to \psi$ — гипотеза;
2) $\neg\psi$ — гипотеза;
3) $\neg\psi \to (\varphi \to \neg\psi)$ — аксиома 1 ($\varphi := \neg\psi$, $\psi := \varphi$);
4) $\varphi \to \neg\psi$ — по правилу отделения из п. 2, 3;
5) $(\varphi \to \psi) \to ((\varphi \to \neg\psi) \to \neg\varphi)$ — аксиома 10 ($\varphi := \varphi$, $\psi := \psi$);
6) $(\varphi \to \neg\psi) \to \neg\varphi$ — по правилу отделения из п. 1, 5;
7) $\neg\varphi$ — по правилу отделения из п. 4, 6.

Следующее правило — *правило цепного заключения*:

$$\frac{\varphi \to \psi,\ \psi \to \chi}{\varphi \to \chi}.$$

Чтобы его обосновать, докажем выводимость $\varphi \to \psi,\ \psi \to \chi \vdash \varphi \to \chi$. По теореме о дедукции указанная выводимость равносильна выводимости $\varphi \to \psi,\ \psi \to \chi,\ \varphi \vdash \chi$, то есть если мы построим такой вывод, тем самым докажем, что можно построить вывод в исходной постановке.

Вывод:

1) $\varphi \to \psi$ — гипотеза;
2) $\psi \to \chi$ — гипотеза;
3) φ — гипотеза;
4) ψ — по правилу отделения из п. 1, 3;
5) χ — по правилу отделения из п. 2, 4.

Рассмотрим *правило соединения посылок*:

$$\frac{\varphi \to (\psi \to \chi)}{(\varphi \,\&\, \psi) \to \chi}.$$

Пользуясь теоремой о дедукции, сводим доказательство исходной выводимости к построению вывода, обосновывающего следующую выводимость: $\varphi \to (\psi \to \chi),\ \varphi \,\&\, \psi \vdash \chi$.

Вывод:

1) $\varphi \to (\psi \to \chi)$ — гипотеза;

2) $\varphi \,\&\, \psi$ — гипотеза;
3) φ — по правилу удаления конъюнкции из п. 2;
4) ψ — по правилу удаления конъюнкции из п. 2;
5) $\psi \to \chi$ — по правилу отделения из п. 1, 3;
6) χ — по правилу отделения из п. 4, 5.

Докажите *правило разъединения посылок* в качестве упражнения:

$$\frac{(\varphi \,\&\, \psi) \to \chi}{\varphi \to (\psi \to \chi)}.$$

Докажем правило *перестановки посылок*:

$$\frac{\varphi \to (\psi \to \chi)}{\psi \to (\varphi \to \chi)}.$$

Требуется доказать выводимость $\varphi \to (\psi \to \chi) \vdash \psi \to (\varphi \to \chi)$, которую с помощью теоремы о дедукции сведем к выводимости $\varphi \to (\psi \to \chi),\ \psi \vdash \varphi \to \chi$. Применяя еще раз теорему о дедукции, приходим к задаче доказать выводимость $\varphi \to (\psi \to \chi),\ \psi,\ \varphi \vdash \chi$.

Упражнение. Запишите вывод формулы χ из гипотез $\varphi \to (\psi \to \chi),\ \psi,\ \varphi$.

Правило сокращения является частным случаем правила резолюции:

$$\frac{\varphi \vee \psi,\ \neg\varphi}{\psi}.$$

Чтобы его обосновать, докажем выводимость $\varphi \vee \psi,\ \neg\varphi \vdash \psi$. Применим правило разбора случаев, для этого удалим дизъюнкцию $\varphi \vee \psi$, разбив ее на два случая:
1) $\varphi,\ \neg\varphi \vdash \psi$;
2) $\psi,\ \neg\varphi \vdash \psi$.

Из обеих выводимостей следует искомая выводимость правила сокращения. Первая из них вытекает из правила вывода из противоречия; вторая сразу очевидна.

3.2.4 Эквивалентность формул ИВ

В алгебре высказываний мы ввели понятие равносильности логических формул, которое означало их одновременную истинность либо ложность на любой оценке. Иначе можно было сформулировать равносильность формул φ и ψ так: из формулы φ следует формула ψ и из формулы ψ следует формула φ, то есть $\varphi \models \psi$ и $\psi \models \varphi$.

В исчислении высказываний вводят аналогичное понятие эквивалентности двух формул. Формулы φ и ψ называются *эквивалентными* (пишут $\varphi \equiv \psi$), если ψ выводима из φ и φ выводима из ψ, то есть $\varphi \vdash \psi$ и $\psi \vdash \varphi$.

Вообще мы видим, что знаки $\models$ и $\vdash$ означают не совсем одно и то же. Знак $\models$ означает логическое следствие и относится к логическим значениям — результатам вычисления формул. Знак $\vdash$ относится к самим формулам как синтаксическим конструкциям и обозначает возможность построить цепочку, приводящую от гипотез к заключительной формуле при помощи допустимых правил вывода.

Какова связь между этими двумя понятиями? Во-первых, по теореме о корректности ИВ (см. с. 53) всякая выводимая формула является истинной, так что всякая формула, полученная путем допустимых правил вывода из гипотез, будет истинна на любой оценке, на которой истинна каждая из гипотез. Во-вторых, правда ли, что всякая тождественно истинная формула выводима, то есть доказуема? В исчислении высказываний на этот вопрос дается положительный ответ, который составляет теорему о полноте ИВ.

Теорема (полнота ИВ). Пусть Γ — множество формул, φ — формула. Если $\Gamma \models \varphi$, то $\Gamma \vdash \varphi$.

Доказательство можно найти в [3, с. 62].

Из теоремы о корректности ИВ вытекает непротиворечивость ИВ: из аксиом невозможно вывести противоречие, то есть какую-либо формулу вместе с ее отрицанием. Выводимость противоречия означало бы истинность формулы $\varphi \,\&\, \neg\varphi$, где φ — некоторая формула. Однако мы знаем закон непротиворечия, согласно которому эта формула тождественно ложна. Следовательно, противоречие невыводимо в ИВ.

Оказывается, что не во всяком формальном исчислении можно доказать любое утверждение, истинное в этой теории, ограниченным набором средств. Было бы неплохо иметь под рукой такую универсальную программу, которая со временем без какого-либо труда с нашей стороны выдавала бы новые знания об окружающем мире в виде формул вместе с их интерпретацией. Однако чисто математические результаты (например, теорема Гёделя о неполноте формальной теории с арифметикой) намекают на то, что такого «вечного двигателя» в логике и вычислениях не существует.

Из определений равносильности и эквивалентности логических формул и теорем о корректности и полноте ИВ следует, что эти два понятия, введенные по-разному, на самом деле тождественны друг другу. Свойство выводимости, понимаемое синтаксически, влечет за собой выполнение семантического свойства логического следствия и наоборот.

$$\boxed{\varphi = \psi \;\Leftrightarrow\; \varphi \equiv \psi}$$

Зачем нужна эквивалентность формул ИВ? У эквивалентных формул есть ряд полезных свойств. Во-первых, вопрос о выводимости интересующей формулы сводится к вопросу о выводимости эквивалентной ей формулы. Во-вторых, есть теорема о замене подформулы на эквивалентную ей формулу: в результате замены получается формула, эквивалентная исходной. Данное свойство позволяет проводить в рамках исчисления высказываний такие же равносильные преобразования, как и в алгебре высказываний, в силу теоремы о полноте ИВ все основные равносильности (логические законы) остаются верными и в исчислении высказываний.

Упражнение. Проверьте, что свойство эквивалентности формул ИВ обладает свойствами рефлексивности, симметричности и транзитивности.

Лемма. Пусть φ, ψ — формулы ИВ, причем $\varphi \equiv \psi$. Тогда φ и ψ одновременно выводимы либо одновременно не выводимы в ИВ, то есть $\vdash \varphi \;\Leftrightarrow \vdash \psi$.

Доказательство. Нам дано, что $\varphi \equiv \psi$. Предположим, что $\vdash \varphi$, и докажем, что $\vdash \psi$. Действительно, в силу эквивалентности $\varphi \equiv \psi$ справедлива выводимость $\varphi \vdash \psi$. Отсюда по теореме о дедукции есть выводимость $\vdash \varphi \to \psi$. Теперь выводимость ψ вытекает из выводимости φ и выводимости $\varphi \to \psi$ по правилу отделения.

Лемма. Пусть $\varphi_1 \equiv \psi_1$ и $\varphi_2 \equiv \psi_2$. Тогда:

1) $\varphi_1 \,\&\, \varphi_2 \equiv \psi_1 \,\&\, \psi_2$;

2) $\varphi_1 \vee \varphi_2 \equiv \psi_1 \vee \psi_2$;

3) $\varphi_1 \to \varphi_2 \equiv \psi_1 \to \psi_2$;

4) $\neg\varphi_1 \equiv \neg\psi_1$.

Доказательство. 1. Требуется доказать, что $\varphi_1 \,\&\, \varphi_2 \vdash \psi_1 \,\&\, \psi_2$. По правилу удаления конъюнкции из гипотезы $\varphi_1 \,\&\, \varphi_2$ выводим формулы φ_1 и φ_2. Из формулы φ_1 выводим формулу ψ_1 (так как по условию леммы $\varphi_1 \equiv \psi_1$, а значит, $\varphi_1 \vdash \psi_1$). Аналогично из формулы φ_2 выводим формулу ψ_2. Наконец, из формул ψ_1, ψ_2 выводим искомую формулу $\psi_1 \,\&\, \psi_2$ по правилу введения конъюнкции.

2. Докажем, что $\varphi_1 \vee \varphi_2 \vdash \psi_1 \vee \psi_2$. По правилу введения дизъюнкции из выводимостей $\varphi_1 \vdash \psi_1$ и $\varphi_2 \vdash \psi_2$ следуют выводимости $\varphi_1 \vdash \psi_1 \vee \psi_2$ и $\varphi_2 \vdash \psi_1 \vee \psi_2$. Отсюда по правилу разбора случаев следует искомая выводимость $\varphi_1 \vee \varphi_2 \vdash \psi_1 \vee \psi_2$.

3. Требуется доказать, что $\varphi_1 \to \varphi_2 \vdash \psi_1 \to \psi_2$. По теореме о дедукции это сводится к доказательству выводимости $\varphi_1 \to \varphi_2,\ \psi_1 \vdash \psi_2$. В силу эквивалентности $\varphi_1 \equiv \psi_1$ имеем $\psi_1 \vdash \varphi_1$. Поэтому из гипотез $\varphi_1 \to \varphi_2$, ψ_1 можно вывести φ_1. Тогда из выведенных формул $\varphi_1 \to \varphi_2$ и φ_1 получаем по правилу отделения формулу φ_2, что и требовалось.

4. Докажем, что $\neg\varphi_1 \vdash \neg\psi_1$. Нам дано $\varphi_1 \equiv \psi_1$. Отсюда вытекает утверждение $\psi_1 \vdash \varphi_1$. По теореме о дедукции $\vdash \psi_1 \to \varphi_1$. Поскольку $\neg\varphi_1$ — гипотеза, то из формул $\psi_1 \to \varphi_1$ и $\neg\varphi_1$ выводим формулу $\neg\psi_1$ по правилу отрицания.

Теорема (о замене). Пусть φ — формула, содержащая подформулу φ'. Допустим, что формула ψ' такова, что $\varphi' \equiv \psi'$. В результате замены подформулы φ' в формуле φ на формулу ψ' получится новая формула ψ. Тогда $\varphi \equiv \psi$.

Доказательство. Индукция по структуре формулы φ. База индукции: берем формулу $\varphi_0 = \varphi'$, полагаем $\psi_0 = \psi'$. По условию теоремы $\varphi_0 \equiv \psi_0$. Гипотеза индукции: $\varphi_k \equiv \psi_k$. Шаг индукции: построим формулу φ_{k+1}, добавив к формуле φ_k одну логическую операцию. Точно так же построим формулу ψ_{k+1}, добавив к формуле ψ_k такую же логическую операцию. Остается доказать, что $\varphi_{k+1} \equiv \psi_{k+1}$.

Рассмотрим случай, когда $\varphi_{k+1} = \neg\varphi_k$ и $\psi_{k+1} = \neg\psi_k$. Так как по гипотезе индукции $\varphi_k \equiv \psi_k$, то по лемме $\varphi_{k+1} \equiv \psi_{k+1}$.

Разберем случай, когда $\varphi_{k+1} = \varphi_k \,\&\, \chi$ и $\psi_{k+1} = \psi_k \,\&\, \chi$. Здесь χ — некоторая формула. Снова по лемме получаем $\varphi_{k+1} \equiv \psi_{k+1}$.

Остальные случаи разбираются аналогично. Таким образом, наращивая шаг за шагом подформулу φ' до формулы φ и одновременно производя те же действия над формулой ψ', мы получим формулу ψ, эквивалентную формуле φ.

Упражнение. Распишите доказательство остальных случаев.

3.2.5 Примеры выводов в ИВ

Пример. Рассмотрим высказывания: “Если я вовремя пришел на пару, то я рано встал”, “Если я не успел позавтракать, то я встал поздно” и “Я встал поздно”. Формализуем эти высказывания, введя элементарные высказывания $A =$ “пришел на пару вовремя”, $B =$ “позавтракал”, $C =$ “рано встал”. Исходная информация формулируется в форме гипотез: $A \to C$, $\neg B \to \neg C$, $\neg C$. Попробуем провести логический вывод. Из первой и третьей гипотезы следует формула $\neg A$ по правилу отрицания. Таким образом, мы получили логическое следствие: “опоздал на пару”.

Пример. Допустим, ты школьник и прогуливаешь уроки. Тогда справедливо, что если ты отличник, то ты прогуливаешь уроки.

Формализуем наши рассуждения. Определим логические переменные $A =$ “школьник”, $B =$ “прогуливает уроки”, $C =$ “отличник”. Исходные данные запишем в виде формулы $A\&$ B. Из них должна следовать истинность формулы $C \to B$. Приходим к задаче установить выводимость

$$A \,\&\, B \vdash C \to B.$$

Распишем вывод:

1) $A \,\&\, B$ — гипотеза;
2) $A \,\&\, B \to B$ — аксиома 4 ($\varphi := A$, $\psi := B$);
3) B — по правилу отделения из п. 1, 2;
4) $B \to (C \to B)$ — аксиома 1 ($\varphi := B$, $\psi := C$);
5) $C \to B$ — по правилу отделения из п. 3, 4.

Пример. В сумку положили тетрадь и ручку. Тогда точно можно утверждать, что в сумке лежит ручка или карандаш.

Введем переменные A = "в сумке тетрадь", B = "в сумке ручка", C = "в сумке карандаш". Нам даны гипотезы $A \,\&\, B$. Выведем из них формулу $B \vee C$:

1) $A \,\&\, B$ — гипотеза;
2) B — по правилу удаления конъюнкции из п. 1;
3) $B \vee C$ — по правилу введения дизъюнкции из п. 2.

Пример. Студент посещает бассейн. Известно, что посещение бассейна поддерживает форму. Значит, если нагрузка по учебе возрастет, то студент будет в форме.

Выделим элементарные высказывания: A = "посещает бассейн", B = "держит форму", C = "нагрузка возросла". Требуется обосновать выводимость:

$$A,\ A \to B \vdash C \to B.$$

Построим вывод:

1) A — гипотеза;
2) $A \to B$ — гипотеза;
3) B — по правилу отделения из п. 1, 2;
4) $B \to (C \to B)$ — аксиома 1 ($\varphi := B$, $\psi := C$);
5) $C \to B$ — по правилу отделения из п. 3, 4.

Пример. Поели, но не попили. Значит, утверждение о том, что еда должна сопровождаться питьем, ложно.

Элементарные высказывания: A = "поели", B = "попили". Известно следующее: A, $\neg B$. Требуется вывести ложность импликации $A \to B$, то есть доказать выводимость

$$A,\ \neg B \vdash \neg(A \to B).$$

Согласно правилу рассуждения от противного, искомая выводимость будет следовать из выводимости $A,\ \neg B,\ A \to B \vdash \bot$, где $\bot$ обозначает противоречие, то есть истинность как некоторой формулы, так и ее отрицания[1].

Построим вывод:

1) A — гипотеза;
2) $\neg B$ — гипотеза;
3) $A \to B$ — гипотеза;
4) B — по правилу отделения из п. 1, 3;
5) $\bot$ — из п. 2, 4.

Пример. Если оба выключателя выключены, то неправда, что они оба включены.

Докажем выводимость: $\neg G,\ \neg H \vdash \neg(G \,\&\, H)$. Согласно правилу рассуждения от противного

$$\frac{\neg G,\ \neg H,\ G \,\&\, H \vdash \bot}{\neg G,\ \neg H \vdash \neg(G \,\&\, H)}.$$

Иными словами, истинность заключения противоречит гипотезам. Отсюда следует, что при истинных гипотезах заключение ложно[2].

Чтобы вывести противоречие из гипотез $\neg G,\ \neg H,\ G \,\&\, H$, применим правило удаления конъюнкции, чтобы вывести H. Итак, мы пришли к противоречию: вывели H и $\neg H$.

Пример. Докажем теорему: $\vdash A \to A$.

1. Заметим, что в аксиоме 2 логическим следствием является импликация. Подумаем, что подставить на место логических переменных в этой аксиоме, чтобы в итоге

[1] Вариант именования знаков: $\vdash$ — швабра, $\models$ — пылесос, $\bot$ — грабли.

[2] Не путать с теоремой о дедукции, когда посылку импликации переносят влево без отрицания!

получить $A \to A$:

$$(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi)).$$

Применим такую подстановку: $\varphi := A$, $\psi := (A \to A)$, $\chi := A$.

Получим

$$(A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A)).$$

2. В аксиоме 1: $\varphi \to (\psi \to \varphi)$ положим $\varphi := A$, $\psi := A \to A$, получим формулу

$$A \to ((A \to A) \to A).$$

3. Поскольку вторая формула в точности равна левой части импликации в формуле 1, отделяем правую часть импликации: $(A \to (A \to A)) \to (A \to A)$.
4. Левая часть импликации в формуле 3 вытекает из аксиомы 1: $A \to (A \to A)$.
5. Отделяем левую часть импликации в формуле 3, которую мы вывели в п. 4, приходим к искомой формуле $A \to A$.

3.2.6 Вывод основных равносильностей

Докажем основные логические законы, которые были сформулированы в алгебре высказываний в качестве равносильностей формул алгебры логики.

Формула замены импликации: $A \to B \equiv \neg A \vee B$.

Докажем выводимость в одну сторону: $A \to B \vdash \neg A \vee B$.

Согласно правилу исчерпывающего разбора случаев наша выводимость будет следовать из доказанных двух других выводимостей, а именно:

$$\frac{A \to B,\ A \vdash \neg A \vee B; \quad A \to B,\ \neg A \vdash \neg A \vee B}{A \to B \vdash \neg A \vee B}.$$

Выводимость $A \to B,\ A \vdash \neg A \vee B$ доказывается так. Из двух гипотез выводим B по правилу отделения. Затем из B выводим $A \vee B$ по правилу введения дизъюнкции.

Выводимость $\neg A \vdash \neg A \vee B$ вытекает из правила введения дизъюнкции.

Докажем выводимость в обратную сторону: $\neg A \vee B \vdash A \to B$.

По теореме о дедукции искомая выводимость будет следовать из доказанной другой выводимости:

$$\frac{\neg A \vee B,\ A \vdash B}{\neg A \vee B \vdash A \to B}.$$

В свою очередь, выводимость $\neg A \vee B,\ A \vdash B$ является разновидностью правила сокращения. Но в таком виде, как это правило сформулировано на с. 56, здесь мы использовать не можем: вторая гипотеза в нем должна быть отрицанием первого аргумента дизъюнкции, а у нас отрицание снято. Поскольку одна из гипотез представляет собой дизъюнкцию, то разберем эти два случая:

$$\frac{\neg A,\ A \vdash B; \quad B,\ A \vdash B}{\neg A \vee B,\ A \vdash B}.$$

Выводимость $\neg A,\ A \vdash B$ следует из правила вывода из противоречия. Выводимость $B,\ A \vdash B$ сразу очевидна.

Закон контрапозиции: $A \to B \equiv \neg B \to \neg A$.

Докажем выводимость в одну сторону: $A \to B \vdash \neg B \to \neg A$. По теореме о дедукции сведем эту выводимость к следующему виду: $A \to B,\ \neg B \vdash \neg A$. Применив правило отрицания (см. с. 55), получаем вывод.

Докажем выводимость в обратную сторону: $\neg B \to \neg A \vdash A \to B$. По теореме о дедукции сведем эту выводимость к следующему виду: $\neg B \to \neg A,\ A \vdash B$. Мы могли бы применить правило отрицания, если бы вместо A у нас была гипотеза $\neg\neg A$. Поэтому построим вывод заново.

Согласно правилу исчерпывающего разбора случаев наша выводимость будет следовать из доказанных двух выводимостей, в которых к нашим гипотезам добавили сначала B, а потом $\neg B$:

$$\frac{\neg B \to \neg A,\ A,\ B \vdash B; \quad \neg B \to \neg A,\ A,\ \neg B \vdash B}{\neg B \to \neg A,\ A \vdash B}.$$

Выводимость $\neg B \to \neg A,\ A,\ B \vdash B$ сразу очевидна. Построим вывод для выводимости $\neg B \to \neg A,\ A,\ \neg B \vdash B$:

1) $\neg B \to \neg A$ — гипотеза;
2) A — гипотеза;
3) $\neg B$ — гипотеза;
4) $\neg A$ — по правилу отделения из п. 1, 3;
5) $\bot$ — из п. 2, 4;
6) B — по правилу вывода из противоречия.

Коммутативность конъюнкции: $A \,\&\, B \equiv B \,\&\, A$.

Докажем выводимость: $A \,\&\, B \vdash B \,\&\, A$. Для этого выведем из гипотезы A и B по правилу удаления конъюнкции, а затем применим правило введения конъюнкции, чтобы вывести $B \,\&\, A$ из этих формул.

Ассоциативность конъюнкции: $(A \,\&\, B) \,\&\, C \equiv A \,\&\, (B \,\&\, C)$.

По правилу удаления конъюнкции из нашей гипотезы выводимы формулы $A \,\&\, B$ и C, из первой из которых выводимы A и B. Из B и C выводим $B \,\&\, C$ по правилу введения конъюнкции и затем получаем искомую формулу по правилу введения конъюнкции.

Коммутативность дизъюнкции: $A \vee B \equiv B \vee A$.

Докажем выводимость: $A \vee B \vdash B \vee A$. Применим правило разбора случаев — разобьем нашу выводимость на две и докажем их: $A \vdash B \vee A$ и $B \vdash B \vee A$. Эти утверждения сразу следуют из правила введения дизъюнкции.

Ассоциативность дизъюнкции: $(A \vee B) \vee C \equiv A \vee (B \vee C)$.

Чтобы доказать выводимость $(A \vee B) \vee C \vdash A \vee (B \vee C)$, построим дерево вывода:

$$\frac{\dfrac{A \vdash A \vee (B \vee C); \quad \frac{B \vdash B \vee C; \quad B \vee C \vdash A \vee (B \vee C)}{B \vdash A \vee (B \vee C)}}{A \vee B \vdash A \vee (B \vee C)}; \qquad \frac{C \vdash B \vee C; \quad B \vee C \vdash A \vee (B \vee C)}{C \vdash A \vee (B \vee C)}}{(A \vee B) \vee C \vdash A \vee (B \vee C)}.$$

Дистрибутивность конъюнкции относительно дизъюнкции: $A \,\&\, (B \vee C) \equiv (A \,\&\, B) \vee (A \,\&\, C)$.

Чтобы доказать выводимость в одну сторону: $A \,\&\, (B \vee C) \vdash (A \,\&\, B) \vee (A \,\&\, C)$, удалим конъюнкцию и разберем случаи, а затем воспользуемся правилом введения конъюнкции. Чтобы доказать выводимость в обратную сторону: $(A \,\&\, B) \vee (A \,\&\, C) \vdash A \,\&\, (B \vee C)$, применим следующие правила: разбора случаев, удаления конъюнкции, введения дизъюнкции и введения конъюнкции.

Упражнение. Распишите выкладки подробно.

Дистрибутивность дизъюнкции относительно конъюнкции: $A \vee (B \,\&\, C) \equiv (A \vee B) \,\&\, (A \vee C)$.

Упражнение. Докажите выводимость: $A \vee (B \,\&\, C) \vdash (A \vee B) \,\&\, (A \vee C)$.

Докажем выводимость в обратную сторону: $(A \vee B) \,\&\, (A \vee C) \vdash A \vee (B \,\&\, C)$. По правилу удаления конъюнкции выводим формулы $A \vee B$, $A \vee C$. Далее применяем правило исчерпывающего разбора случаев, согласно которому наша выводимость будет следовать

из двух выводимостей: $A \vee B,\ A \vee C,\ A \vdash A \vee (B \,\&\, C)$ и $A \vee B,\ A \vee C,\ \neg A \vdash A \vee (B \,\&\, C)$. Первая выводимость получается по правилу введения дизъюнкции. Вторая выводимость получается по правилам сокращения, введения конъюнкции и введения дизъюнкции.

Упражнение. Распишите выкладки подробно.

Закон двойного отрицания: $\neg\neg A \equiv A$.

Докажем выводимость в одну сторону: $\neg\neg A \vdash A$. Согласно правилу исчерпывающего разбора случаев достаточно доказать две выводимости:

$$\frac{\neg\neg A,\ A \vdash A; \quad \neg\neg A,\ \neg A \vdash A}{\neg\neg A \vdash A}.$$

Первая выводимость $\neg\neg A,\ A \vdash A$ сразу очевидна. Вторая выводимость $\neg\neg A,\ \neg A \vdash A$ следует из правила вывода из противоречия.

Докажем выводимость в обратную сторону: $A \vdash \neg\neg A$. Поскольку справа стоит формула с отрицанием, воспользуемся правилом рассуждения от противного: перенесем заключение к гипотезам без отрицания и из полученного множества гипотез выведем противоречие:

$$\frac{A,\ \neg A \vdash \bot}{A \vdash \neg\neg A}.$$

Выводимость $A,\ \neg A \vdash \bot$ очевидна.

Закон де Моргана: $\neg(A \vee B) \equiv \neg A \,\&\, \neg B$.

Докажем выводимость в одну сторону: $\neg(A \vee B) \vdash \neg A \,\&\, \neg B$. По правилу введения дизъюнкции $A \vdash A \vee B$. Отсюда по закону контрапозиции $\neg(A \vee B) \vdash \neg A$. Аналогичным образом выводим $\neg(A \vee B) \vdash \neg B$. По правилу введения конъюнкции $\neg(A \vee B) \vdash \neg A \& \neg B$.

Докажем выводимость в обратную сторону: $\neg A \& \neg B \vdash \neg(A \vee B)$. По правилу удаления конъюнкции из исходной гипотезы выводимы формулы $\neg A$, $\neg B$. Остается воспользоваться правилом рассуждения от противного:

$$\frac{\neg A,\ \neg B,\ A \vee B\ \vdash \bot}{\neg A,\ \neg B \vdash \neg(A \vee B)}.$$

Упражнение. Воспользуйтесь правилом сокращения для вывода противоречия из гипотез $\neg A,\ \neg B,\ A \vee B$.

Закон де Моргана: $\neg(A \,\&\, B) \equiv \neg A \vee \neg B$.

Воспользуемся доказанными выше эквивалентностями:

$$\neg A \vee \neg B \equiv \neg\neg(\neg A \vee \neg B) \equiv \neg(\neg\neg A \,\&\, \neg\neg B) \equiv \neg(A \,\&\, B).$$

3.3 Исчисление предикатов

Язык логики предикатов, как и язык логики высказываний, можно сделать формальным. Это нужно для того, чтобы придать математической теории максимальную строгость. Формальный язык необходим также для компьютерной обработки знаний.

Синтаксические соглашения логики предикатов будут следующими. Алфавит языка включает: предметные переменные, предикатные символы, функциональные символы, предметные константы, знаки логических операций и вспомогательные символы (скобки, запятые).

К системе аксиом исчисления высказываний добавим аксиому удаления квантора общности

$$\forall x\, \varphi(x) \to \varphi(t),$$

в которой $\varphi(x)$ — любая формула со свободным вхождением переменной x.

Добавим также аксиому введения квантора существования:

$$\varphi(t) \to \exists x\, \varphi(x).$$

Помимо правила отделения, добавим еще два правила вывода.

$\forall$-правило: из формулы $\psi \to \varphi(x)$ непосредственно выводится формула $\psi \to \forall x\, \varphi(x)$, если ψ не содержит свободных вхождений x (не зависит от x).

$\exists$-правило: из формулы $\varphi(x) \to \psi$ непосредственно выводится формула $\exists x\, \varphi(x) \to \psi$, если ψ не содержит свободных вхождений x (не зависит от x).

Оказывается, что введенное таким образом исчисление предикатов обладает свойством полноты. Это значит, что любую общезначимую формулу логики предикатов можно вывести в таком исчислении. Но, в отличие от исчисления высказываний, исчисление предикатов неразрешимо. Это значит, что не существует универсального алгоритма проверки общезначимости произвольной формулы логики предикатов.

3.4 Практикум

3.4.1 Типовые примеры

Пример 3.1. Проиллюстрируем и проверим первые две аксиомы исчисления высказываний:

1. $A \to (B \to A)$
   Если сегодня хорошее настроение, то, если сегодня пойдет дождь, настроение всё равно будет хорошее.
2. $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
   Если в дождь из наличия зонта следует хорошее настроение, то из наличия зонта в дождь следует хорошее настроение в дождь.

Чтобы убедиться в правильности первой аксиомы (как тавтологии алгебры высказываний), построим таблицу истинности:

A	B	$B \to A$	$A \to (B \to A)$
0	0	1	1
0	1	0	1
1	0	1	1
1	1	1	1

Чтобы проверить вторую аксиому, применим метод от противного.

Найдем такие значения A, B, C, для которых посылка импликации истинна, а заключение ложно:

$$A \to (B \to C) = 1, \quad (A \to B) \to (A \to C) = 0.$$

Второе уравнение сведем к двум уравнениям:

$$A \to B = 1, \quad A \to C = 0.$$

Из последнего уравнения находим $A = 1$, $C = 0$. Подставляя значение $A = 1$ в уравнение $A \to B = 1$, выводим $B = 1$. Однако найденные значения переменных не удовлетворяют самому первому уравнению $A \to (B \to C) = 1$, поскольку $1 \to (1 \to 0) = 1 \to 0 = 0$. Значит, искомых значений A, B, C не существует, поэтому логическое следствие невозможно опровергнуть.

Некоторые математические результаты при видимой их сложности сводятся к формуле $A \to A$, например: если тут всё нормально, то и тут всё нормально. Как мы выяснили на с. 59, эту формулу можно вывести из первых двух аксиом ИВ.

Пример 3.2. Наличие work-life balance можно охарактеризовать наличием как работы, так и личной жизни. Если человек — молодой исследователь, то в случае, когда он не выигрывает грантов, у него не остается денег на личную жизнь, а в случае, когда он выигрывает гранты, у него нет времени на личную жизнь. Правда ли, что для молодого исследователя недостижим work-life balance?

Обозначим свойство "быть молодым исследователем" как высказывание A, наличие работы — B, наличие личной жизни — C. Наличие у человека work-life balance выразим формулой $B \,\&\, C$. Таким образом, нам нужно проверить истинность (выводимость) формулы $A \to \neg(B \,\&\, C)$.

Конечно, сама по себе эта формула тавтологией не является, однако нам дана дополнительная информация, заведомая истинность которой может гарантировать, что наша формула при этих условиях окажется истинной. Например, мы можем утверждать истинность формулы $A \to B$, поскольку ситуацию «свободного художника» мы заведомо исключаем. Введем еще переменные G = "есть гранты", D = "есть деньги на личную жизнь", E = "есть время на личную жизнь". Тогда мы можем принять как гипотезы формулы: $G \to \neg D$, $\neg G \to \neg E$, а поскольку личная жизнь требует как денег, так и времени, то примем гипотезу $C \to (D \,\&\, E)$.

Мы пришли к задаче доказательства либо опровержения следующей выводимости:

$$A \to B,\ G \to \neg D,\ \neg G \to \neg E,\ C \to (D \,\&\, E) \vdash A \to \neg(B \,\&\, C).$$

Поскольку мы хотим вывести импликацию, то уместно свести задачу к более простой, применив теорему о дедукции. Согласно этой теореме наша выводимость равносильна следующей:

$$A \to B,\ G \to \neg D,\ \neg G \to \neg E,\ C \to (D \,\&\, E),\ A \vdash \neg(B \,\&\, C).$$

Справа от знака выводимости стоит отрицание формулы, что дает нам повод применить правило введения отрицания, согласно которому наша выводимость будет следовать из следующей:

$$A \to B,\ G \to \neg D,\ \neg G \to \neg E,\ C \to (D \,\&\, E),\ A,\ B \,\&\, C \vdash \bot,$$

где символ $\bot$ обозначает противоречие — это любая формула, выведенная вместе со своим отрицанием.

Распишем вывод противоречия из 6 гипотез:

1) $A \to B$ — гипотеза;
2) $G \to \neg D$ — гипотеза;
3) $\neg G \to \neg E$ — гипотеза;
4) $C \to (D \,\&\, E)$ — гипотеза;
5) A — гипотеза;
6) $B \,\&\, C$ — гипотеза;
7) B — по правилу отделения (modus ponens) из п. 1, 5
8) C — по правилу удаления конъюнкции из п. 6;
9) $D \,\&\, E$ — по правилу отделения из п. 4, 8;
10) D — по правилу удаления конъюнкции из п. 9;
11) E — по правилу удаления конъюнкции из п. 9;

12) $\neg G$ — по правилу отрицания (modus tollens) из п. 2, 10;
13) G — по правилу отрицания из п. 3, 11;
14) $\bot$ — из п. 12, 13.

Правда ли, что если бы всякое выяснение отношений сводилось к вычислениям, то человечество стало бы счастливее?

Пример 3.3. Чтобы у человека была личная жизнь, ему нужно для начала учиться или работать. Если человек учится, у него есть деньги на личную жизнь, только если он получает стипендию или подрабатывает. Если у человека нет денег на личную жизнь, то у него нет личной жизни. Что можно сказать о человеке, у которого есть личная жизнь, но при этом он не работает?

Формализуем высказывания. Введем логические переменные: A = "учится", B = "работает", C = "есть личная жизнь", D = "есть деньги на личную жизнь", S = "получает стипендию". Первое предложение означает, что из того, что у человека есть личная жизнь, следует, что он учится или работает, то есть $C \to (A \vee B)$. Во втором предложении присутствуют слова "только если", которые могут сбить с толку. Здесь подразумевается "если" в обратную сторону: все ситуации, в которых у человека есть деньги на личную жизнь, предполагают, что он получает стипендию или подрабатывает. Получили формулу $A \to (D \to (S \vee B))$. Третье предложение выражается формулой $\neg D \to \neg C$.

Нам нужно применить исчисление высказываний для вывода логических следствий из следующих гипотез:

$$C \to (A \vee B),\ A \to (D \to (S \vee B)),\ \neg D \to \neg C,\ C,\ \neg B \vdash\ ?$$

Применим к третьей гипотезе закон контрапозиции, согласно которому справедлива эквивалентность формул: $\neg D \to \neg C \equiv C \to D$, в том числе есть выводимость $\neg D \to \neg C \vdash C \to D$. Значит, формулу $C \to D$ мы можем присоединить к нашим гипотезам.

Затем из $C \to D$ и C выводим D по правилу отделения. Из C и первой гипотезы также выводим $A \vee B$.

Если бы точно нам было известно A, то мы могли бы отделить A во второй гипотезе и вывести $D \to (S \vee B)$ и далее $S \vee B$. Но поскольку истинность A нам неизвестна, то рассмотрим еще один случай, когда истинно $\neg A$. Но тогда из $\neg A$ и $A \vee B$ выводим B, значит, по правилу введения дизъюнкции получаем $S \vee B$.

Итак, в случае A из гипотез вывели $S \vee B$, в случае $\neg A$ также вывели $S \vee B$. По правилу исчерпывающего разбора случаев из гипотез выводима формула $S \vee B$.

Из $S \vee B$ и $\neg B$ по правилу резолюции выводим S. Таким образом, из полученной информации следует, что у не работающего человека с личной жизнью есть стипендия.

Порицать молодого человека за то, что он влюблен, все равно, что упрекать кого-нибудь в том, что он болен (Ш. Дюкло).

Пример 3.4. Докажем *правило слияния*:

$$\frac{A \vee B,\ A \to B}{B}.$$

Воспользуемся правилом разбора случаев. Разберем отдельно случай, когда истинно A, и отдельно случай, когда истинно B. Тогда

$$\frac{A,\ A \to B \vdash B; \quad B,\ A \to B \vdash B}{A \vee B,\ A \to B \vdash B}.$$

Эту запись мы получили из правила разбора случаев (см. с. 54), где положили $\Gamma := \{A \to B\}$, $\varphi := A$, $\psi := B$, $\chi := B$.

Нам остается обосновать две выводимости над чертой, тогда мы автоматически получим выводимость под чертой. Первая выводимость вытекает из правила modus ponens. Вторая выводимость сразу очевидна.

Смелость — начало дела, но случай — хозяин конца (Демокрит).

Пример 3.5. Если вы пожарили яичницу и сварили кофе, то, если вы завтракаете, вы выпьете кофе и съедите яичницу.

Выведем формулу, выражающую данное высказывание: $A \,\&\, B \to (C \to (B \,\&\, A))$. По теореме о дедукции

$$\frac{A \,\&\, B \vdash C \to (B \,\&\, A)}{\vdash A \,\&\, B \to (C \to (B \,\&\, A))}.$$

Эта запись означает, что если верна выводимость над чертой, то верна и выводимость под чертой.

Еще раз воспользуемся теоремой о дедукции:

$$\frac{A \,\&\, B,\ C \vdash B \,\&\, A}{A \,\&\, B \vdash C \to (B \,\&\, A)}.$$

Выводимость над чертой следует из коммутативности конъюнкции.

Жить с человеком, которого любишь, так же трудно, как любить человека, с которым живешь (Жан Ростан).

Пример 3.6. Если открыть окно, то подует свежий воздух, но станет холодно. Если закрыть окно, то станет душно. Если холодно или душно, то будет некомфортно. Если подует свежий воздух, то будет комфортно. Следовательно, окно закрыто.

Введем логические переменные: A = "окно открыто", B = "свежий воздух", C = "холодно", D = "душно", E = "комфортно". Формализуем рассуждения:

$$A \to B \,\&\, C,\ C \vee D \to \neg E,\ B \to E \vdash \neg A.$$

Воспользуемся правилом рассуждения от противного, согласно которому искомая выводимость вытекает из выводимости

$$A \to B \,\&\, C,\ C \vee D \to \neg E,\ B \to E,\ A \vdash \bot,$$

которую мы сейчас докажем:

1) $\dfrac{A \to B \,\&\, C,\ \ A}{B \,\&\, C}$ (MP) $\quad\Rightarrow\quad B \,\&\, C$;
2) $\dfrac{B \,\&\, C}{B}$ (&-уд) $\quad\Rightarrow\quad B$;
3) $\dfrac{B \,\&\, C}{C}$ (&-уд) $\quad\Rightarrow\quad C$;
4) $\dfrac{B,\ \ B \to E}{E}$ (MP) $\quad\Rightarrow\quad E$;
5) $\dfrac{C}{C \vee D}$ ($\vee$-вв) $\quad\Rightarrow\quad C \vee D$;
6) $\dfrac{C \vee D,\ \ C \vee D \to \neg E}{\neg E}$ (MP) $\quad\Rightarrow\quad \neg E$;
7) $\dfrac{E,\ \ \neg E}{\bot}$ $\quad\Rightarrow\quad \bot$.

Брак — это союз между мужчиной, который не может спать при закрытом окне, и женщиной, которая не может спать при открытом окне (Джордж Бернард Шоу).

Пример 3.7. Невозможно сравнить ВВГУ и ДВФУ, как нельзя сравнивать лучший вуз с крупнейшим.

Определим высказывание A = "ВВГУ — лучший вуз". Тогда его отрицанию придадим смысл $\neg A$ = "ВВГУ — крупнейший вуз". Аналогично B = "ДВФУ — лучший вуз" и $\neg B$ = "ДВФУ — крупнейший вуз". Примем гипотезы $A \to \neg B$, $\neg B \to A$. Невозможность сравнить два вуза выразим формулой $A \,\&\, \neg B \vee \neg A \,\&\, B$.

Докажем выводимость:

$$A \to \neg B,\ \neg B \to A \vdash A \,\&\, \neg B \vee \neg A \,\&\, B.$$

Согласно закону исключенного третьего (аксиома 11) справедлива формула $A \vee \neg A$. Разберем два случая. В случае, когда истинна формула A, выводим $\neg B$ (по modus ponens) и $A\&\neg B$ (правило введения конъюнкции). В случае, когда истинна формула $\neg A$, выводим $\neg\neg B$ (по modus tollens), что эквивалентно B, и по правилу введения конъюнкции выводим $\neg A\&B$. В каждом из этих случаев можно применить правило введения дизъюнкции, чтобы вывести искомую формулу $A \,\&\, \neg B \vee \neg A \,\&\, B$.

Есть, помимо классической, другие логики. В интуиционистской логике не принимают закон исключенного третьего, а отсюда пропадает и закон двойного отрицания. Такая логика дает возможность смоделировать ограниченность получаемой информации об окружающем мире.

3.4.2 Задачи и упражнения

1. Постройте вывод в исчислении высказываний:
   - $F \to (G \to H),\ F \,\&\, G \vdash H$;
   - $F \to G,\ G \to H,\ F \vee G \vdash H$;
   - $\neg G \to \neg F,\ \neg G \to F \vdash G$.
2. При помощи законов двойного отрицания и теоремы о дедукции докажите, что следующие формулы являются теоремами исчисления высказываний:
   - $\vdash (a \to b) \to (\overline{\overline{a}} \to b)$;
   - $\vdash b \to ((b \to a) \to \overline{\overline{a}}$;
   - $\vdash \overline{\overline{a}} \to ((a \to \overline{\overline{b}}) \to b)$;
   - $\vdash (a \to b) \to (\overline{\overline{a}} \to \overline{\overline{b}})$.
3. Докажите следующие теоремы исчисления высказываний [15, §13]:
   - $\vdash (F \to \neg F) \to \neg F$;
   - $\vdash (F \vee (G \vee H)) \to ((F \vee G) \vee H)$;
   - $\vdash \neg(F \to G) \to F$;
   - $\vdash (P \to (Q \to R)) \to (Q \to (P \to R))$;
   - $\vdash P \to (Q \to P)$.

   Используйте эти формулы в качестве тестов в лабораторной работе на исчисление высказываний.
4. Докажите теоремы:
   (a) $x \vee xy \to x$;
   (b) $xy \to x \vee y$;
   (c) $(xy)z \to x(yz)$;
   (d) $x \vee (y \vee z) \to (x \vee y) \vee z$;
   (e) $xy \vee xz \to x(y \vee z)$;
   (f) $(\overline{x} \to x) \to x$;
   (g) $((x \to y) \to x) \to x$;
   (h) $(x \to (y \to x)) \to (x \to y)$;
   (i) $(x \to y) \vee (y \to x)$;

(j) $(x \to \overline{x}) \to \overline{x}$.

5. Докажите выводимости:
   (a) $A \vdash A \vee B$;
   (b) $A, B \vdash B \,\&\, A$;
   (c) $A \to C,\ B \to C \vdash A \vee B \to C$;
   (d) $A \,\&\, B \vdash C \to A$;
   (e) $A \to B,\ \neg B \vdash \neg A$;
   (f) $\neg A \to \neg B,\ B \vdash A$;
   (g) $A,\ A \to C \vdash B \to C$;
   (h) $\neg A \vee B \vdash A \to B$;
   (i) $B, A \to (B \to C) \vdash A \to C$;
   (j) $A \,\&\, B,\ A \to (B \to C) \vdash C$.

Индивидуальное домашнее задание № 3 [44, 46, 47].

Выполните 4 задания, соответствующие вашему варианту.

Задание 1. Постройте вывод формулы из данных гипотез в исчислении высказываний:

1) $G \vdash H \to (F \to G)$;
2) $F \to G, F \to (G \to H), F \vdash H$;
3) $\overline{G} \to \overline{F}, F \vdash G$;
4) $F \vdash H \to (\overline{G} \to F)$;
5) $\overline{F}, \overline{G} \vdash \neg(\overline{F} \to G)$;
6) $\neg G, H \vdash \neg(G \,\&\, H)$;
7) $\neg G, H \vdash G \vee H$;
8) $\neg G, \neg H \vdash G \to H$;
9) $\neg G, H \vdash G \to H$;
10) $G, H \vdash G \to H$.

Задание 2. Постройте вывод формулы из данных гипотез в исчислении высказываний:

1) $A \to B,\ A \to C \ \vdash\ A \to B \,\&\, C$;
2) $A \to C,\ B \to C \ \vdash\ A \vee B \to C$;
3) $A \to B \ \vdash\ (C \to A) \to (C \to B)$;
4) $A \to B \ \vdash\ A \,\&\, C \to B \,\&\, C$;
5) $A \to B \ \vdash\ A \vee C \to B \vee C$;
6) $A \,\&\, B \ \vdash\ A \,\&\, (A \vee B)$;
7) $A \vee (A \,\&\, B) \ \vdash\ A$;
8) $A \,\&\, C \ \vdash\ A \,\&\, (\neg B \vee C)$;
9) $A \vee (\neg A \,\&\, B) \ \vdash\ A \vee B$;
10) $A \to B,\ A \,\&\, C \ \vdash\ (B \,\&\, C) \vee D$.

Задание 3. При помощи законов двойного отрицания и теоремы о дедукции докажите теорему исчисления высказываний:

1) $\vdash (a \to (b \to c)) \to ((a \to \overline{\overline{b}}) \to (a \to c))$;
2) $\vdash (a \to b) \to ((a \to (b \to \overline{\overline{c}})) \to (\overline{\overline{a}} \to c))$;
3) $\vdash (b \to a) \to ((b \to (a \to c)) \to (b \to \overline{\overline{c}}))$;
4) $\vdash (a \to (b \to c)) \to (\overline{\overline{b}} \to (a \to \overline{\overline{c}}))$;
5) $\vdash (a \to (\overline{\overline{b}} \to \overline{\overline{c}})) \to (\overline{\overline{a}} \to (b \to c))$;
6) $\vdash (a \to (b \to \overline{\overline{c}})) \to ((a \to b) \to (\overline{\overline{a}} \to c))$;
7) $\vdash (a \to \overline{\overline{b}}) \to ((b \to \overline{\overline{c}}) \to (\overline{\overline{a}} \to c))$;
8) $\vdash (\overline{\overline{a}} \to b) \to ((b \to \overline{\overline{c}}) \to (a \to c))$;
9) $\vdash (a \to b) \to ((b \to \overline{\overline{c}}) \to (\overline{\overline{a}} \to c))$;
10) $\vdash (\overline{\overline{a}} \to (\overline{\overline{b}} \to \overline{\overline{c}})) \to (a \to (b \to c))$.

Задание 4. При помощи закона контрапозиции и теоремы о дедукции докажите теорему исчисления высказываний.

1) $\vdash (a \to b) \to ((\overline{c} \to \overline{b}) \to (a \to \overline{\overline{c}}))$;
2) $\vdash (a \to b) \to ((a \to (\overline{c} \to \overline{b})) \to (a \to c))$;
3) $\vdash (a \to b) \to ((a \to (c \to \overline{b})) \to (a \to \overline{c}))$;
4) $\vdash (b \to a) \to ((\overline{b} \to c) \to (\overline{a} \to c))$;
5) $\vdash (\overline{b} \to \overline{a}) \to ((b \to c) \to ((\overline{\overline{a}} \to c))$;
6) $\vdash (b \to \overline{a}) \to ((\overline{b} \to c) \to (\overline{\overline{a}} \to c))$;
7) $\vdash (a \to (b \to c)) \to ((\overline{b} \to \overline{a}) \to (\overline{\overline{a}} \to c))$;
8) $\vdash (a \to (\overline{c} \to \overline{b})) \to ((\overline{b} \to \overline{a}) \to (\overline{\overline{a}} \to c))$;
9) $\vdash (\overline{b} \to \overline{a}) \to ((\overline{c} \to \overline{b}) \to (\overline{\overline{a}} \to c))$;
10) $\vdash (\overline{c} \to \overline{b}) \to ((\overline{b} \to \overline{a}) \to (\overline{\overline{a}} \to c))$.

3.4.3 Лабораторные работы

Лабораторная работа № 9 "Знакомство с исчислением высказываний"

Вам дана программа "логик-теоретик", которая принимает на вход логическую формулу и пытается найти ее вывод в исчислении высказываний с использованием правил вывода.

Программа применяет следующий алгоритм автоматического построения вывода:

- Чтобы получить отрицание, надо добавить к гипотезам выражение без отрицания и получить противоречие.
- Чтобы получить конъюнкцию, надо вывести оба множителя.
- Чтобы получить дизъюнкцию, надо добавить к гипотезам отрицание одного слагаемого и попытаться вывести другое слагаемое.
- Чтобы получить импликацию, надо добавить левую часть к гипотезам и попытаться вывести заключение.
- Чтобы получить противоречие, надо для каждой формулы из множества гипотез попытаться вывести её отрицание из остальных формул.
- Применяются также следующие индуктивные процедуры к множеству гипотез и выведенных формул:
  - modus ponens: если среди гипотез есть $A \to B$ и A, добавляем B;
  - modus tollens: если среди гипотез есть $A \to B$ и $\neg B$, добавляем $\neg A$;
  - сокращение слагаемого: если есть $\neg A$ и $A \vee B$ или $B \vee A$, добавляем B;
  - удаление конъюнкции;
  - введение дизъюнкции.

Приведите не менее 5 примеров формул, при выводе которых используются:

- правило введения импликации (теорема о дедукции);
- правило рассуждения от противного;
- правило введения конъюнкции;
- правило удаления конъюнкции;
- правило modus ponens.

Решите с помощью программы следующие задачи:

- правило соединения посылок: $A \to (B \to C) \vdash A \,\&\, B \to C$;
- правило разъединения посылок: $A \,\&\, B \to C \vdash A \to (B \to C)$;
- правило перестановки посылок: $A \to (B \to C) \vdash B \to (A \to C)$;
- $\vdash (X \to Z) \to ((Y \to Z) \to ((X \,\&\, Y) \to Z))$.

Глава 4

Модели вычислений

У попа была собака,
Он ее любил.
У попа была собака...

4.1 Как вычисляет компьютер

Компьютерная архитектура предполагает наличие следующих элементов:
- центральный процессор: содержит устройство микропрограммного управления, регистровую память;
- оперативная память: связана с процессором через системную шину и предоставляет процессору пространство адресуемой памяти;
- устройства ввода/вывода.

Для управления вычислениями существует система машинных команд. Программа, которая считывается процессором из оперативной памяти, написана на машинном языке. Информация о последовательности микроопераций для реализации конкретной машинной команды содержится в устройстве микропрограммного управления.

Основные типы команд: команды перемещения данных между регистрами и оперативной памятью, арифметические и логические команды, команды условного и безусловного перехода[1].

Таким образом, работу компьютера можно представить как последовательную смену состояния памяти за счет перехода между командами программы. Математически любые вычисления можно выполнить на абстрактной вычислительной машине, считывающей программу из памяти и выполняющей только операции смены внутреннего состояния и состояния текущей ячейки с перемещением только между соседними ячейками (без произвольного доступа к памяти). Возможно смоделировать работу компьютера на такой машине[2].

Для облегчения программирования существуют языки, каждый из которых основан на определенной модели вычислений.

4.2 Языки и автоматы

Чтобы сделать информацию пригодной к машинной обработке, ее необходимо представить в строковом виде, то есть в виде *данных*. Все типы данных — числа, таблицы, строки, графы — можно представить в виде последовательности символов (например, только 0 и 1).

[1]Юров В.И. Assembler. Учебник для вузов. — Санкт-Петербург: Питер, 2008.

[2]Хопкрофт Д.Э., Мотвани Р., Ульман Д.Д. Введение в теорию автоматов, языков и вычислений. — Москва: Изд. дом "Вильямс", 2008.

Пусть задано непустое конечное множество A, называемое *алфавитом*. Из элементов алфавита, называемых *символами*, составим все возможные строки — множество таких строк обозначим A^*. В это множество входит и пустая строка Λ.

Поставим задачу вычисления предиката, заданного на множестве A^*. Напомним, что предикат — это свойство, которое для каждого элемента множества либо выполняется (предикат равен 1), либо не выполняется (предикат равен 0). Допустим, предикат задает, какие строки являются корректными. Поставлена вычислительная задача проверки строки на корректность. Говорят, что это задача *распознавания языка*.

Конечный автомат представляет собой вычислительную модель, способную распознавать регулярные языки, то есть языки, которые можно задать в виде регулярного выражения. На рисунке 4.1 представлена диаграмма состояний конечного автомата для распознавания десятичной записи числа без ведущих нулей. Кружочки — это состояния автомата, стрелочки — это переходы между состояниями при считывании определенного символа. Двойным кружочком обозначено заключительное состояние, символом \$ обозначен конец входной строки.

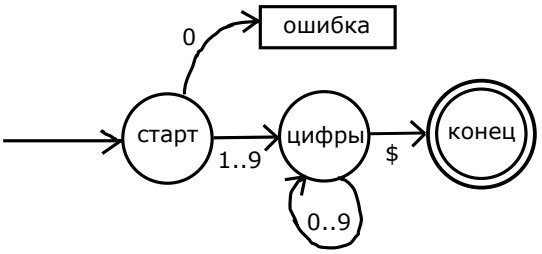


Рис. 4.1. Конечный автомат для распознавания десятичной записи числа

Упражнение. Переделайте автомат так, чтобы он распознавал в том числе и число 0.

Можно доказать, что не существует конечного автомата, распознающего корректность правильно построенных арифметических выражений. Это связано с тем, что выражения могут содержать рекурсивно вложенные конструкции любой степени вложенности, а так как число состояний автомата конечно, то их может не хватить для подсчета числа открывающих и закрывающих скобок.

Языки такого типа распознаются автоматами со стеком, так называемыми *магазинными автоматами*. По сравнению с компьютерами магазинные автоматы имеют доступ не ко всей памяти сразу, а только к информации, находящейся на вершине стека. Магазинный автомат совершает переход в зависимости от текущего состояния, входного символа и символа на вершине магазина.

Таким образом, для распознавания некоторых языков хватает упрощенных вычислителей. В следующем подразделе мы разберем наиболее универсальный вычислитель, возможности которого такие же, как у современных компьютеров, в том смысле, что любой алгоритм, который можно ввести в реальную вычислительную машину, можно запрограммировать и на абстрактной машине Тьюринга.

4.3 Машина Тьюринга

Представим робота для инструктирования человека на тренировках в спортзале. Робот дал физкультурнику задание: сделать 5 жимов штанги лежа по 20 кг и 10 жимов 30 кг на тренажере — 3 подхода. Чтобы робот мог посчитать, сколько раз человек сделал каждое упражнение, потребуется модель конечного автомата с лентой для записи результатов промежуточных вычислений.

Конечный автомат — это вычислительная модель, состоящая из множества состояний и множества переходов между состояниями в зависимости от информации, поданной на вход. Если автомат читает и пишет информацию с одной и той же ленты, получаем *машину Тьюринга* (рис. 4.2).

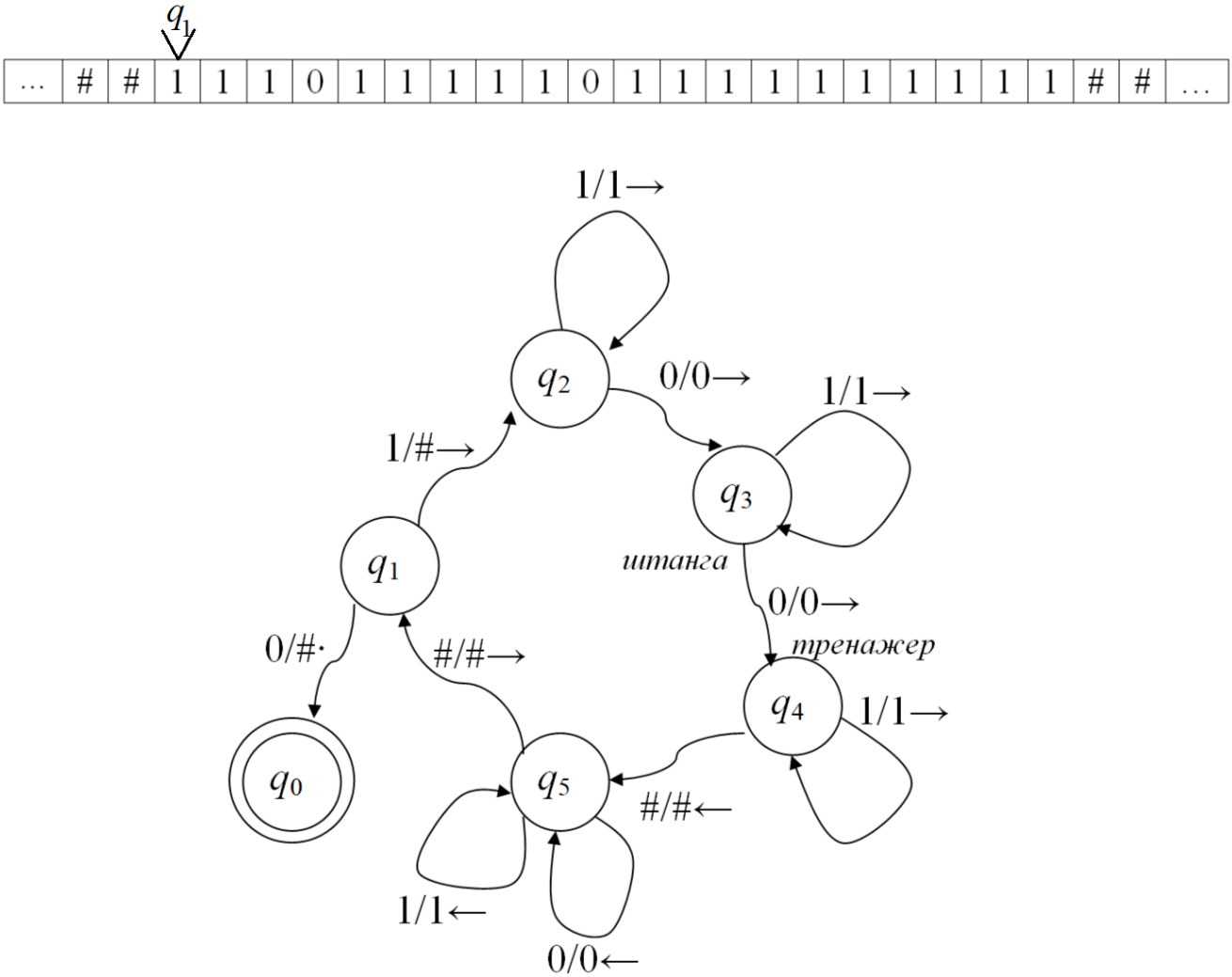


Рис. 4.2. Начальная конфигурация и диаграмма состояний машины Тьюринга

Таблица 4.1

Таблица переходов

	1	0	#
q_1	$q_2\# \rightarrow$	$q_0\#\cdot$	
q_2	$q_2 1 \rightarrow$	$q_3 0 \rightarrow$	
q_3	$q_3 1 \rightarrow$	$q_4 0 \rightarrow$	
q_4	$q_4 1 \rightarrow$		$q_5\# \leftarrow$
q_5	$q_5 1 \leftarrow$	$q_5 0 \leftarrow$	$q_1\# \rightarrow$

Перед запуском скрипта на ленте записаны три числа в унарной записи: 3, 5 и 10. Каждое число представлено последовательностью из такого же количества единиц, числа разделены нулем. Незаполненное пространство ленты обозначено символом #.

Робот должен повторить 3 раза одни и те же действия: сначала вывести на дисплей 5 раз «штанга» и затем 10 раз «тренажер». После каждой итерации робот будет ждать ввода — нажатия на кнопку.

Упражнение. Введите эту программу в эмулятор машины Тьюринга, запустите ее и проверьте лог эмулятора.

Начальное состояние — q_1, заключительное состояние — q_0. В состоянии q_1 машина стирает единицу и сдвигается вправо. Затем в состоянии q_2 она пропускает все единицы, вплоть до ближайшего нуля. Встретив ноль, машина переходит в состояние q_3, в котором она считает, сколько раз нужно поднять штангу, сдвигаясь на одну ячейку вправо. Встретив снова ноль, машина переходит в состояние q_4, в котором она считает, сколько раз нужно сделать упражнение на тренажере. Встретив пустой символ, машина переходит в состояние q_5, в котором она идет влево, пропуская все цифры, вплоть до ближайшего пустого символа. Встретив пустой символ, машина снова переходит в состояние q_1, и если текущий символ — 1 (подходы не закончились), то итерации продолжаются. Если текущий символ — 0, то он стирается и машина переходит в заключительное состояние q_0.

4.4 Рекурсивные функции

Цель настоящего подраздела — описать, что вычислимо, а что нет. Речь пойдет о массовых проблемах, то есть проблемах вычисления некоторой числовой функции на всей ее области определения. Говорят, что решение массовой проблемы вычислимо, если существует алгоритм, решающий эту проблему.

Будем рассматривать числовые функции, принимающие на вход целые неотрицательные числа и выдающие также целое неотрицательное число. Поскольку любую информацию можно представить в строковом виде, а все строки можно пронумеровать, то наиболее общая ситуация сводится к такой постановке.

Построим класс функций, называемых *примитивно рекурсивными*. В него входят все функции, которые можно построить с помощью оператора примитивной рекурсии, примененного к базисным функциям:

– тождественный ноль;
– прибавление единицы;
– выбор из кортежа значений.

Оператор примитивной рекурсии применяет алгоритмически заданную зависимость к результату вычисления той же функции, аргумент которой на единицу меньше.

Например, функцию прибавления числа 5 к числу x можно свести к пятикратному повторению функции прибавления единицы, так что $\mathrm{add5}(x) = \mathrm{next}(\mathrm{next}(\mathrm{next}(\mathrm{next}(\mathrm{next}(x)))))$.

Функцию прибавления числа y к числу x можно представить как итерационное повторение y раз прибавления 1 к числу x. Запишем это с помощью оператора примитивной рекурсии:

$$\mathrm{add}(x, y+1) = \mathrm{next}(\mathrm{add}(x, y)); \quad \mathrm{add}(x, 0) = x.$$

Функцию умножения числа a на число b можно представить как итерационное повторение сложения числа b с числом a:

$$\mathrm{mul}(a, b+1) = \mathrm{add}(\mathrm{mul}(a, b), a); \quad \mathrm{mul}(a, 0) = 0.$$

Оператор примитивной рекурсии состоит из: а) зависимости значения функции на следующем шаге от значения той же функции на предыдущем шаге; б) начального условия.

Переход от предыдущего шага к следующему осуществляется за счет прибавления 1 к переменной, по которой производится рекурсия.

Упражнение. Запишите схему примитивной рекурсии для функции усеченного вычитания sub, которая выдает целое неотрицательное число, равное разности двух чисел либо нулю, если первое число меньше второго.

Введем также класс *частично рекурсивных функций*. Слово "частично" подчеркивает, что функции из этого класса бывают частичными, то есть определенными не для всех входных значений. Частично рекурсивные функции получаются добавлением к классу примитивно рекурсивных функций всех функций, получаемых вдобавок к оператору примитивной рекурсии с помощью оператора минимизации — подсчета числа шагов.

Например,

$$\mathrm{div}(a,b) = \mu_x[\mathrm{sub}(a, \mathrm{mul}(b,x)) = 0].$$

Эта запись означает, что функция $\mathrm{div}(a,b)$ равна подсчету числа шагов, которые необходимо сделать до минимального x, для которого $\mathrm{sub}(a, \mathrm{mul}(b,x)) = 0$.

Упражнение. Внесите изменения в эту формулу, чтобы вычислить частное a/b, округленное вниз.

Оказывается, что описанных средств хватает, чтобы записать все функции, вычислимые по Тьюрингу.

Введение в теорию рекурсивных функций подробно изложено в [20].

4.5 Алгоритмическая логика

Центральный момент в любых вычислениях — спецификация этих вычислений. Спецификация программы должна точно описывать, что должно быть совершено в результате выполнения программы[3].

Обозначение

$$\{Q\}\ S\ \{R\},$$

где Q и R — предикаты; S — программа или ее часть, имеющая следующую интерпретацию: если выполнение S началось в состоянии, удовлетворяющем Q, то гарантируется, что оно завершится через конечное время в состоянии, удовлетворяющем R.

Предикат Q называется *предусловием*, предикат R — *постусловием*. При условии истинности $\models \{Q\}\ S\ \{R\}$ говорят, что программа *корректна*. Если при выполнении предусловия не гарантируется, что программа завершит выполнение за конечное число шагов, то такую программу можно назвать *частично корректной*.

Обозначим через $\mathrm{wp}(S,R)$ множество всех состояний, для которых выполнение команды S, начавшееся в таком состоянии, обязательно закончится через конечное время в состоянии, удовлетворяющем R. Это множество называется *слабейшим предусловием* для S по отношению к R, поскольку оно определяет множество всех состояний, таких, что выполнение, начавшееся в любом из них, закончится при истинном R.

Таким образом, обозначение $\{Q\}\ S\ \{R\}$ — другая форма предиката $Q \to \mathrm{wp}(S,R)$. Обратим внимание на то, что такая запись — это формула логики предикатов. Далее опишем дедуктивную систему для вывода в алгоритмической логике.

– Аксиома присваивания:

$$\vdash \{p(x)[t/x]\}\ \mathtt{x = t}\ \{p(x)\}.$$

Здесь $p(x)[t/x]$ обозначает формулу, в которой вместо x подставили t. Эта аксиома утверждает, что если изначально выполнялось условие $p(x)$, в котором x заменили

[3]Грис Д. Наука программирования. — Москва: Мир, 1984.

на t, то после выполнения присваивания `x = t` новое значение переменной `x` будет удовлетворять условию $p(x)$.

– Правило композиции:

$$\frac{\vdash \{p\}\ \texttt{S1}\ \{q\} \qquad \vdash \{q\}\ \texttt{S2}\ \{r\}}{\vdash \{p\}\ \texttt{S1 S2}\ \{r\}}.$$

Другими словами, если при выполнении начального условия p после выполнения `S1` будет выполнено условие q, а при выполнении начального условия q после выполнения `S2` будет выполнено заключительное условие r, то тогда при выполнении начального условия p после выполнения `S1` и `S2` будет выполняться заключительное условие r.

– Правило альтернативы:

$$\frac{\vdash \{p \,\&\, B\}\ \texttt{S1}\ \{q\} \qquad \vdash \{p \,\&\, \neg B\}\ \texttt{S2}\ \{q\}}{\vdash \{p\}\ \texttt{if (B) S1 else S2}\ \{q\}}.$$

Это правило означает, что если при выполнении условия p вместе с условием B после выполнения `S1` будет выполняться условие q, а при выполнении условия p вместе с условием $\neg B$ после выполнения `S2` также будет выполняться условие q, тогда если изначально выполнялось условие p, то после выполнения условного оператора `if (B) S1 else S2` в любом случае будет выполняться условие q.

– Правило цикла:

$$\frac{\vdash \{p \,\&\, B\}\ \texttt{S}\ \{p\}}{\vdash \{p\}\ \texttt{while (B) S}\ \{p \,\&\, \neg B\}}.$$

Формула p носит название инварианта цикла. Правило цикла утверждает следующее: если нам известно, что при выполнении предусловия B логическое условие p сохраняется после выполнения тела цикла `S`, то отсюда можно сделать заключение о частичной корректности цикла: после его завершения (если он завершится) будет выполнено условие p вместе с отрицанием условия цикла B.

– Правило следствия:

$$\frac{\vdash p_1 \to p \qquad \vdash \{p\}\ \texttt{S}\ \{q\} \qquad \vdash q \to q_1}{\vdash \{p_1\}\ \texttt{S}\ \{q_1\}}.$$

Правило следствия утверждает, что можно усилить предусловие и ослабить постусловие.

Формулы корректности можно использовать для построения программы по спецификации. Основная идея в том, чтобы сосредоточиться на инвариантах циклов. Удобно мыслить с конца, находя для цели программы слабейшее предусловие вместе с оператором, который обеспечит искомый результат [35].

4.6 Практикум

4.6.1 Типовые примеры

Пример 4.1. Составим программу для машины Тьюринга, вычисляющую функцию усеченного вычитания единицы $f(x) = x \dot{-} 1$.

На ленте машины Тьюринга записан вход — число x в унарной записи. Считаем, что число $x \geqslant 1$ представляется в виде последовательности из x единиц, а число 0 — в виде символа 0.

Приведем пример начальной конфигурации машины Тьюринга и составим программу так, чтобы она выполнялась корректно на этом примере. Пусть на ленте записано три единицы, причем каретка указывает на первый символ входа в состоянии q_1. Конфигурация: q_1111.

Требуется, чтобы машина стерла одну единицу и остановилась на первом символе выхода. Проще всего стереть первую единицу с помощью команды $1q_1 \to \#q_0R$. Следующая конфигурация: q_011.

Остается рассмотреть случай, когда на ленте записан символ 0. Запишем команду: $0q_1 \to 0q_0N$.

Получили программу:

$1q_1 \to \#q_0R$

$0q_1 \to 0q_0N$

Обратим внимание, что левые части команд машины Тьюринга не должны повторяться.

Направление иногда зависит и от "состояния": девочки — налево, мальчики — направо.

Пример 4.2. Докажем, что функция $\mathrm{sub1}(x) = x \dot{-} 1$ примитивно рекурсивна.

Составим схему примитивной рекурсии:

$$\mathrm{sub1}(0) = 0,$$

$$\mathrm{sub1}(x+1) = x.$$

Пример 4.3. Составим программу для машины Тьюринга, которая складывает два числа.

Начальная конфигурация: $q_1 \underbrace{11...1}_{n \text{ раз}} 0 \underbrace{11...1}_{m \text{ раз}}$. Конечная конфигурация: $q_0 \underbrace{11...1}_{(n+m) \text{ раз}}$.

Нужно стереть одну единицу, а затем заменить ноль на единицу.

Единицу сотрем с помощью команды $1q_1 \to \#q_2R$. Дойдем до нуля с помощью команды, которая будет выполняться "в цикле": $1q_2 \to 1q_2R$. Встретив ноль в состоянии q_2, заменим его на единицу и вернемся обратно: $0q_2 \to 1q_3L$; $1q_3 \to 1q_3L$; $\#q_3 \to \#q_0R$.

Пример 4.4. Докажем, что функция целой части квадратного корня из x частично рекурсивна.

Представим функцию $f(x) = [\sqrt{x}]$ с помощью оператора минимизации. Требуется определить функцию, равенство которой нулю будет означать конец "цикла while". Если мы посчитаем, сколько раз нужно прибавить к y единицу, чтобы y^2 оказался больше либо равен x, то мы сможем найти округление вверх корня из x:

$$\lceil\sqrt{x}\rceil = \mu_y(x \dot{-} y^2).$$

Как только выражение в скобках станет равным 0, цикл остановится и текущее значение y считается результатом вычисления выражения $\mu_y(x \dot{-} y^2)$.

Методом проб и ошибок можно получить следующую формулу:

$$\lceil\sqrt{x}\rceil = \mu_y((x+1) \dot{-} (y+1)^2).$$

Можно проверить на компьютере, перебрав x, что левая часть всегда равна правой.

Вместо того, чтобы гадать, вычислима функция или нет, надо взять ее и вычислить.

Пример 4.5. `while (x > 0) x = x − 1;`

Инвариантом цикла служит условие $x \geqslant 0$, так как при выполнении условия цикла оно сохраняется после выполнения тела цикла. После окончания выполнения цикла имеем $(x \leqslant 0)\ \&\ (x \geqslant 0)$. Отсюда $x = 0$. Таким образом, по правилу цикла

$$\frac{\vdash \{x > 0\}\, \mathtt{x = x - 1}\, \{x \geqslant 0\}}{\vdash \{x \geqslant 0\}\, \mathtt{while\ (x > 0)\ x = x - 1}\, \{x = 0\}}.$$

Пример 4.6. Проведем верификацию следующей программы, снабдив ее логическими комментариями:

```
x = 0; y = b;
while (y != 0) {
   // x = (b - y) * a
   x = x + a;
   y = y - 1;
   // x' = (b - (y - 1)) * a = (b - y') * a
}
// x = a * b
```

Чтобы доказать правильность работы цикла, проверяют: инициализацию (инвариант должен выполняться перед началом выполнения цикла), сохранение инварианта в процессе выполнения одного шага цикла, из которого вытекает, что инвариант будет выполняться после окончания выполнения цикла.

В данном случае условие корректности выглядит так: $b \geqslant 0$.

Пример 4.7.
```
a = x + 1;
if (a - 1 == 0) {
   y = 1;
} else {
   y = a;
}
```

Чтобы разобраться, что делает эта программа, добавим в нее логические комментарии:

```
a = x + 1;
if (a - 1 == 0) {  // a == 1
   y = 1;
   // y = a = x + 1
} else {
   y = a;
   // y = a = x + 1
}
// y = x + 1
```

Многие знакомы с пошаговым выполнением программы, но не все из них выполняют пошаговый анализ алгоритма.

4.6.2 Лабораторные работы

Лабораторная работа № 10 “Программирование машины Тьюринга”

Воспользуйтесь любым эмулятором машины Тьюринга[4].

1. Реализуйте функцию усеченного вычитания 1 и функцию прибавления 2.
2. Вам дана программа умножения на 2. Исправьте программу так, чтобы она выводила два исходных числа в унарной записи, разделенные нулем. Например: $111 \to 1110111$.
3. Реализуйте программу деления нацело на 2.

Лабораторная работа № 11 “Композиция машин Тьюринга”

В [27, с. 119] вводятся базовые модули для машины Тьюринга, например: сдвиг вправо, сдвиг влево, транспозиция.

Вам даны программы, решающие эти подзадачи. Ваша задача — объединить их в соответствии с формулой [27, с. 121], чтобы реализовать циклическую перестановку трех аргументов.

Лабораторная работа № 12 “Частично рекурсивные функции”

Дана заготовка на Python, в которой реализован класс, позволяющий определить вычислительный блок по схеме примитивной рекурсии, и такой же класс для оператора минимизации.

Ваша задача — подавая на вход классам только базисные функции, реализовать вычисление следующих функций:

- $f(x, y) = x \dot{-} y$;
- $f(x) = x^2 + 2x + 1$;
- $f(x, y) = [x/y]$.

[4]Например: URL: https://github.com/lapkin25/turing-machine-simulator

Глава 5

Алгоритмы

Были данные,
Да стали другими.
Кто знает, какими?

5.1 Абстрактные типы данных

Информатика — наука об абстракциях, изучающая, как построить наилучшую модель предметной области, которая позволит автоматизировать решение поставленной задачи. Примером такого моделирования служит булева алгебра, которая помогает конструировать электронные схемы. Можно выделить "три кита" решения задач с помощью компьютера:

- Модели данных — это абстракции, выражаемые в виде дискретных математических структур.
- Структуры данных — это представление моделей данных с помощью языков программирования.
- Алгоритмы — способы обработки данных, представленных в виде моделей или структур данных, для решения определенных задач.

Данные, относящиеся к определенной модели данных, над которыми определен набор операций доступа и модификации, называют *абстрактным типом данных* (АТД). Один и тот же АТД может быть реализован с помощью различных структур данных, и от реализации будет зависеть эффективность операций над АТД — время выполнения и объем используемой памяти. Важно отметить, что реализация АТД всегда должна поддерживать структуру данных в корректном состоянии, то есть сохранять логический инвариант относительно представленных данных.

Таким образом, грамотное программирование предполагает явное указание всех логических инвариантов в качестве комментариев к классам и методам, а для каждой операции над АТД должна быть указана формальная спецификация.

Наиболее часто используются такие АТД, как: списки, таблицы, множества, деревья, графы [33].

5.2 Сложность вычислений

Временной сложностью алгоритма называют время его выполнения в зависимости от размера задачи.

Для измерения сложности алгоритма вводят асимптотические классы сложности[1]. Например, класс $O(n \log n)$ включает в себя такие зависимости от n, которые, начиная с некоторого $n = n_0$, растут не быстрее, чем $Cn \log n$, где C — какая-нибудь константа.

[1]Стивенс Р. Алгоритмы. Теория и практическое применение. — Москва: Изд-во «Э», 2016.

Эффективные алгоритмы имеют такую сложность, которую можно выразить полиномом от n. Малоэффективные (при больших n) алгоритмы выражаются функцией, превосходящей любой полином, например: $O(n!)$ или $O(2^n)$.

Например, задача проверки выполнимости булевой формулы является $\mathcal{NP}$-полной. Это означает, что она принадлежит классу $\mathcal{NP}$ (ее можно решить за полиномиальное время на недетерминированной машине Тьюринга) и, кроме того, любая задача из класса сложности $\mathcal{NP}$ может быть сведена к этой задаче. Еще один пример — задача коммивояжера с заданной границей для стоимости маршрута[2].

При подсчете времени выполнения реальной программы необходимо приблизительно оценить число элементарных операций, которые будут выполнены согласно этой программе. Число операций порядка $10^8 - 10^9$ соответствует примерно одной секунде машинного времени.

Например, если нужно упорядочить по возрастанию массив из $n = 10^6$ элементов, то алгоритм, имеющий сложность $O(n^2)$, потребует число операций, верхняя оценка которого увеличивается в 4 раза при увеличении n вдвое. Константа C при n^2 будет небольшой, если алгоритм состоит из двойного цикла с небольшим количеством операций в нем. Получаем $n^2 = 10^{12}$, а значит, число операций, которые потребуются для сортировки массива этим алгоритмом, потребует порядка 10^4 с. Для сортировки массива существуют алгоритмы, имеющие сложность $O(n \log n)$, которые могут отсортировать массив из 10^8 элементов за секунду.

5.3 Последовательности

5.3.1 Типы последовательностей

Одним из основных абстрактных типов данных является *список* — упорядоченный набор элементов. В зависимости от набора операций над списком, которые должны выполняться эффективно, применяются разные реализации списка.

Реализация списка с помощью *массива* обладает операциями доступа:

– $A[i]$ — чтение i-го по порядку элемента массива A;

– $\textsc{Length}(A)$ — число элементов в массиве A

и операциями модификации:

– $@A[i]$ — изменение i-го по порядку элемента массива A;

– $\textsc{Append}(@A, x)$ — вставка элемента x в конец массива A;

– $x = \textsc{Pop}(@A)$ — удаление элемента x с конца непустого массива A.

Реализация списка с помощью *связанного списка* обладает операциями:

– $*p$ — доступ к элементу списка по указателю p;

– $@ * p$ — модификация элемента списка по указателю p;

– $\textsc{Insert}(@L, p, x)$ — вставка нового элемента в список L сразу после элемента, на который указывает p;

– $\textsc{Delete}(@L, p, x)$ — удаление элемента списка L, на который указывает p;

– $p = \textsc{Head}(L)$ — указатель на начало списка L.

Все перечисленные операции имеют сложность $O(1)$, то есть имеют время выполнения, не зависящее от размера массива.

В связи с тем, что в массиве элементы хранятся в памяти друг за другом, существует эффективный способ обращения к любому элементу по его номеру — адреса элементов массива образуют арифметическую прогрессию.

[2]Рейнгольд Э., Нивергельд Ю., Део Н. Комбинаторные алгоритмы: теория и практика. — Москва: Мир, 1980.

В связанном списке ситуация другая — элементы списка располагаются в памяти где попало, но ссылаются друг на друга. За счет этого операции вставки и удаления элементов между элементами списка будут выполняться быстро, но доступа к произвольному элементу по его номеру в связанном списке нет.

Частными случаями списка являются: стек, очередь, дек. Вставка и удаление в стек производятся только с конца. Вставка в очередь производится в конец, а удаление — с начала. Вставка и удаление элементов из дека производятся с начала и конца.

Стек легко реализуется с помощью массива. Очередь и дек можно реализовать кольцевым массивом, а именно: заранее выделяется память для хранения элементов очереди и предполагается, что этой памяти будет достаточно. Сама очередь располагается в выделенном массиве, начиная с любого его элемента, а по достижении конца массива она продолжается с его начала. Получается, что массив как бы свернут в кольцо.

Упражнение. Реализуйте структуры данных «стек» и «очередь» с операциями вставки и удаления элемента.

Существует ли способ изменить размер массива? И каким образом оптимальнее это делать? Это возможно в *динамическом массиве*. Память для хранения динамического массива выделяется не всякий раз, когда его размер изменяется. Эффективнее увеличивать и уменьшать размер динамического массива только вдвое. Тогда вставка m элементов в массив потребует всего $\log m$ операций выделения памяти.

5.3.2 Сортировка

Сортировка вставками

Рассмотрим последовательность элементов некоторого типа данных, которую нужно расположить в таком порядке, чтобы в отсортированной последовательности каждый последующий элемент следовал за предыдущим (в смысле отношения линейного порядка, введенного на множестве значений нашего типа данных). Последовательность может быть представлена в виде массива или списка. Для выполнения процедуры сортировки можно: 1) сравнивать любые пары элементов последовательности; 2) перемещать элементы последовательности в другое место, например менять два элемента местами.

Принцип работы сортировки вставками следующий. Вся последовательность разбивается на две части: отсортированная часть и следующая за ней неотсортированная часть. На каждой итерации алгоритма берется первый элемент неотсортированной части и вставляется в отсортированную часть в нужное место, то есть чтобы отсортированная часть осталась отсортированной.

Инициализация: отсортированная часть пуста, неотсортированная часть – это вся исходная последовательность. Шаг алгоритма: пусть элементы $a[1..i-1]$ отсортированы, а остальные элементы $a[i..n]$ расположены в произвольном порядке. Найдем для элемента $a[i]$ место среди элементов $a[1..i]$, а именно: найдем наибольший индекс $j \in 1..i-1$, для которого $a[j] \leqslant a[i]$, а если такого j не нашлось, то полагаем $j = 0$. Тогда элемент $a[i]$ нужно вставить в позицию $j + 1$. Завершение: по окончании работы алгоритма отсортированная часть — это вся последовательность.

Упражнение. Реализуйте подпрограмму перемещения элемента массива в заданную позицию либо разберитесь, как это сделать с помощью функций стандартного контейнера выбранного языка программирования. Если у вас есть своя реализация связанного списка, реализуйте эту подпрограмму для списка.

Поиск места для вставки займет (в худшем случае) порядка i операций, перемещение i-го элемента массива в найденную позицию также займет порядка i операций, а в сумме получим $O(n^2)$ операций — это сложность алгоритма сортировки вставками.

Сортировка выбором

Упражнение. Проанализируйте следующий алгоритм и оцените время его выполнения:

```
SelectionSort (@A[1..n]) = {
   for i = 2..n
      // подмассив A[1..i-1] отсортирован
      //   и содержит i-1 наименьших элементов массива A[1..n]

      // в подмассиве A[i..n] выбираем наименьший элемент
      // и складываем результат в A[i]
      for j = i .. n {
         if A[i] > A[j] {
            swap(@A[i], @A[j])  // меняем местами два элемента массива
         }
      }

      // подмассив A[1..i] отсортирован и содержит i наименьших
      //   элементов массива A[1..n]
   }
}
```

Рекуррентное соотношение: на i-й итерации алгоритма сортировки выбором к массиву из $i-1$ наименьших элементов последовательности $\{a_k\}$ добавляется наименьший из элементов последовательности $a_i, a_{i+1}, \ldots, a_n$. В результате получается массив из i наименьших элементов последовательности $\{a_k\}$.

Алгоритм сортировки выбором обладает свойством неустойчивости: он не сохраняет порядок одинаковых элементов. Убедитесь в этом.

Пирамидальная сортировка

Пирамидальная сортировка основана на частичном упорядочении множества элементов последовательности при построении специальной структуры данных — *пирамиды*, или кучи.

Чтобы построить из заданного набора чисел пирамиду, нужно договориться о ее свойствах. Потребуем, чтобы в каждом узле пирамиды находился элемент, не превосходящий элементов, находящихся в потомках этого узла. Тогда в вершине пирамиды будет находиться минимальный элемент нашего множества. Тогда, чтобы линейно упорядочить элементы пирамиды, извлечем из нее минимальный элемент n раз. Таким образом, нам потребуются две операции: 1) построение пирамиды по заданному набору элементов; 2) удаление элемента с вершины пирамиды.

Пирамида — это двоичное дерево, для хранения которого можно использовать массив, в котором для каждого узла с номером $i = 1..n$ два его потомка имеют индексы $2i$ и $2i+1$.

Для построения пирамиды вызовем для каждого элемента половины массива $i = 1..[n/2]$ процедуру восстановления свойства пирамиды: если данный элемент превосходит какого-то из своих потомков, то его нужно обменять местами с наименьшим из них и рекурсивно вызвать процедуру восстановления свойства пирамиды опять для данного элемента уже в новом месте, которая может спустить его еще ниже, и так далее. Процедура восстановления свойства пирамиды предполагает, что для поддеревьев данного элемента свойство пирамиды выполнено. Поэтому при построении пирамиды вызывать ее нужно с конца половины массива.

Чтобы удалить минимальный элемент из пирамиды, поставим на место минимального элемента $a[1]$ элемент $a[n]$ и, если пирамида не пуста, вызовем процедуру восстановления свойства пирамиды для $i = 1$.

Время выполнения процедуры удаления минимального элемента пропорционально высоте пирамиды и составляет $O(\log n)$. Построение пирамиды требует столько операций, чему равна сумма высот вершин половины пирамиды. Число вершин высоты i равно 2^{h-i} ($h = \log n$ — высота пирамиды); для таких вершин процедура восстановления кучи займет порядка i операций. Поэтому общая стоимость построения кучи равна

$$\sum_{i=2}^{h} i2^{h-i} = n\sum_{i=2}^{h}\frac{i}{2^i} < n\sum_{i=2}^{\infty}\frac{i}{2^i} = Cn$$

(ряд сходится по признаку сравнения, так как $i < 2^{i/2}$ для $i \geqslant 4$, а сумму ряда можно найти почленным дифференцированием). Поэтому сложность пирамидальной сортировки равна $O(n + n\log n) = O(n\log n)$.

Сортировка слиянием

Поясним обозначение O, введенное выше. Под *временной сложностью алгоритма* понимают функцию $T(n)$ от размерности входных данных n, равную наибольшему времени работы алгоритма на входных данных размерности n. Размерность входных данных — это мера их количества; она может быть задана несколькими числами.

Эффективность алгоритма зависит от того, как быстро растет $T(n)$ с ростом n. Основным показателем временной сложности алгоритма служит асимптотическая скорость увеличения времени работы алгоритма при $n \to \infty$. Например, если временная сложность — это полином $T(n) = an^p + bn^{p-1} + cn + d$, то при больших n слагаемое со старшей степенью n^p будет значительно больше, чем все другие слагаемые, и поэтому порядок скорости роста $T(n)$ характеризует эффективность алгоритма при больших n.

Говорят, что $T(n)$ — бесконечно большая функция менее высокого порядка, чем $f(n)$, если $T(n) \leqslant Cf(n)$, начиная с некоторого $n = n_0$. В этом случае пишут $T(n) = O(f(n))$, подразумевая под $O(f(n))$ класс функций, а равенство в этой формуле обозначает принадлежность функции этому классу.

Предположим, что временная сложность алгоритма выражается рекуррентным соотношением

$$T(n) = aT(n/b) + f(n).$$

Это имеет место, когда, например, задача размера n разбивается на a подзадач размера n/b и время обработки результатов их решения для получения решения задачи размера n составляет $f(n)$. Тогда если $f(n) = O(n^d)$, то

$$T(n) = \begin{cases} O(n^d), & \text{если } a < b^d, \\ O(n^d\log n), & \text{если } a = b^d, \\ O(n^{\log_b a}), & \text{если } a > b^d. \end{cases}$$

Доказательство этого утверждения, на которое ссылаются как на мастер-теорему, проводится методом математической индукции.

Рекурсивный алгоритм сортировки слиянием состоит в следующем: последовательность разбивается на две равные части, затем каждая часть сортируется и к результатам сортировки применяется процедура слияния двух отсортированных последовательностей в одну:

```
MergeSort (@A[1..n]) = {
   if n <= 1 {
      return
   }
   MergeSort(@A[1..[n/2]])
   MergeSort(@A[[n/2] + 1..n])
   @A = Merge(A[1..[n/2]], A[[n/2] + 1..n])
}
```

Заметим, что подпрограмма сортировки дважды вызывает саму себя с глубиной рекурсии порядка $\log_2 n$. Суммарное количество сравнений на каждой глубине рекурсии равно $O(n)$. Таким образом, сортировка слиянием потребует порядка $O(n \log n)$ операций.

Упражнение. Запишите рекуррентное соотношение для времени выполнения сортировки слиянием и примените для его оценки мастер-теорему.

Алгоритм сортировки слиянием хорошо подходит для сортировки связанных списков, так как он, в отличие от алгоритмов пирамидальной и быстрой сортировки, не основывается на произвольном доступе к элементам. При сортировке массивов требуется вспомогательный буфер для выполнения слияния двух отсортированных массивов. Отсортированные связанные списки можно легко слить вместе, не требуя дополнительной памяти, а просто упорядочивая указатели.

Сортировка слиянием является классическим примером алгоритмов типа "разделяй и властвуй". Этот метод разбивает одну большую задачу на две подзадачи, которые легче решить. Далее пользуемся двумя частичными решениями для создания решения всей задачи, что было сделано операцией слияния.

Быстрая сортировка

Идея быстрой сортировки состоит в таком разбиении массива на два подмассива, при котором все элементы первого подмассива не превосходят любой элемент второго подмассива. Очевидно, что в таком случае можно вызвать рекурсивно процедуру сортировки для обоих подмассивов, и тогда весь массив окажется отсортированным.

Алгоритм быстрой сортировки массива:

Разделение. Массив $A[p..r]$ разбивается на два (возможно, пустых) подмассива $A[p..q-1]$ и $A[q+1..r]$, таких, что каждый элемент $A[p..q-1]$ меньше или равен $A[q]$, который, в свою очередь, не превышает любой элемент подмассива $A[q+1..r]$. Индекс q вычисляется в ходе процедуры разбиения.

Обработка. Подмассивы $A[p..q-1]$ и $A[q+1..r]$ сортируются с помощью рекурсивного вызова процедуры быстрой сортировки.

Соединение. Поскольку подмассивы сортируются на месте, для их объединения не требуются никакие действия: весь массив $A[p..r]$ оказывается отсортированным.

Быстрая сортировка реализуется следующей процедурой:

```
QuickSort (@A[p..r]) = {
   if p < r {
      q = Partition(@A[p..r])
      QuickSort(@A[p..q - 1])
      QuickSort(@A[q + 1..r])
   }
}
```

Проанализируем процедуру разбиения:

```
q = Partition (@A[p..r]) = {
   q = p
   for j = p..r - 1 {
      // инвариант цикла:
      // p <= q <= j,
      // A[i] <= A[r], p <= i <= q-1, A[i] > A[r], q <= i <= j-1
      if A[j] <= A[r] {
         swap(@A[j], @A[q])
         q = q + 1
      }
      // p <= q <= j+1,
      // A[i] <= A[r], p <= i <= q-1, A[i] > A[r], q <= i <= j
   }
   swap(@A[r], @A[q])
}
```

Инвариант цикла: $p \leqslant q \leqslant j$ и подмассив $A[p..j-1]$ разбивается на 2 подмассива: $A[p..q-1] \leqslant A[r]$, $A[q..j-1] > A[r]$.

Инициализация. При $j = p$ подмассив $A[p..j-1]$ пуст, поэтому инвариант выполняется.

Сохранение. Чтобы обеспечить инвариант на j-й итерации цикла, сравним $A[r]$ с $A[j]$. Если $A[r] < A[j]$, то подмассив $A[p..q-1] \leqslant A[r]$ (остается на месте), и $A[q..j] > A[r]$. А если $A[j] \leqslant A[r]$, то первый элемент второго подмассива перемещаем в конец, $A[q] \leftrightarrow A[j]$, а $A[j]$ помещаем в конец первого подмассива.

В первом случае $A[r] < A[j]$ граница подмассивов и первый подмассив остаются прежними, и новое логическое условие $A[p..q-1] \leqslant A[r]$ выполняется, а также $A[q..j] > A[r]$. Во втором случае $A[j] \leqslant A[r]$ имеем $A'[p..q'-1] = A'[p..q] \leqslant A'[r]$ (так как $A'[q] = A[j] \leqslant A[r] = A'[r]$); $A'[q'..j] > A'[r]$, где $A'[q'..j] = A[q+1..j-1] \times A[q] > A[r]$.

Завершение. При $j = r$ после окончания выполнения цикла инвариант сохраняется, так что $p \leqslant q \leqslant r$, $A[p..q-1] \leqslant A[r]$, $A[q..r-1] > A[r]$.

Поменяем местами $A[r] \leftrightarrow A[q]$, получим искомое разбиение массива $A[p..r]$ на два подмассива $A[p..q-1]$, $A[q]$, $A[q+1..r]$ (возможно, пустых), таких, что $A[p..q-1] \leqslant A[q]$, $A[q+1..r] > A[q]$. Здесь $p \leqslant q \leqslant r$.

В данном случае в качестве опорного элемента в процедуре Partition мы выбрали самый правый элемент. Можно подобрать набор тестов, на котором быстрая сортировка будет давать асимптотику $O(n^2)$. Для исключения плохих случаев применяют рандомизированные алгоритмы, а именно: поменяем самый правый элемент со случайным элементом либо выберем 3 случайных элемента и возьмем в качестве опорного элемента медиану выбранной тройки.

Время выполнения быстрой сортировки в худшем случае равно $O(n^2)$, но за счет рандомизации удается отойти от худшей оценки, а среднее время выполнения быстрой сортировки растет как $n \log n$. Это следует из рекуррентного соотношения для среднего времени выполнения

$$T_{\text{ср}}(n) \leqslant \frac{2}{n-1} \sum_{k=1}^{n-1} T_{\text{ср}}(k) + Cn,$$

которое имеет решение $T_{\text{ср}}(n) = O(n \log n)$.

5.3.3 Упорядоченный список

Не всякая последовательность хранится именно как последовательность. Можно удивиться, если узнать, что в классе `SortedList` из пакета `sortedcontainers` в Python для

реализации упорядоченного списка с операциями вставки, удаления и поиска элементов используется дерево двоичного поиска.

Дерево двоичного поиска — это двоичное дерево, узлы которого помечены ключами и значениями. Характеристическое свойство дерева двоичного поиска: все элементы, хранящиеся в узлах левого поддерева узла x, меньше элемента, содержащегося в узле x, а все элементы, хранящиеся в узлах правого поддерева узла x, больше элемента, содержащегося в узле x[3].

Чтобы сохранить последовательность в виде дерева, мы просто можем хранить множество элементов в этом дереве. Чтобы найти элемент по индексу, надо хранить в каждом узле дерева размер его поддерева. Алгоритм обхода дерева рекурсивный: те же самые действия, запущенные для одного узла дерева, вызываются этой же функцией для других узлов.

5.4 Практикум

5.4.1 Длинные числа

Обсудим технологию разработки вычислительных библиотек на примере модуля для работы с длинными числами. Прежде всего, в любой серьезной библиотеке должна присутствовать четкая спецификация: что должно подаваться на вход и что следует ожидать на выходе. Чтобы сформулировать спецификацию функций для работы с длинными числами, начнем с четкого задания структуры данных "длинное число".

Тип данных определяет математическую конструкцию, реализованную в компьютере с помощью структуры данных, вместе с набором операций, выполняемых над этими данными. Структура данных — это конкретная реализация типа данных, которая определяет способ хранения и представления информации о математической конструкции и способ реализации операций над этим объектом. В нашем случае тип данных "длинное число" — это не что иное как представление целого числа в системе счисления по основанию $m > 1$, то есть последовательность цифр.

Представлением числа $a \in \mathbb{N}_0$ в системе счисления по основанию m называется запись числа в виде последовательности цифр $a_0, a_1, a_2, \ldots, a_{k-1}$ из множества $0..m-1$, удовлетворяющей равенству $a = a_0 + a_1 m + a_2 m^2 + \ldots + a_{k-1} m^{k-1}$, причем при $k > 1$ имеем $a_{k-1} > 0$. Можно доказать, что для любого числа $a \in \mathbb{N}_0$ такое представление существует и единственно.

Для компьютерной реализации типа данных "длинное число" потребуется конкретизировать способ хранения последовательности цифр, а затем реализовать на языке программирования алгоритмы операций над длинными числами. Хранить последовательность цифр будем в виде массива целых чисел digits[ ] (цифры числа от младшей к старшим), целочисленнной переменной num_digits (количество цифр) и основания системы счисления radix. Важно, чтобы все операции над длинными числами сохраняли логический инвариант, определяющий корректное состояние структуры данных.

Корректным состоянием структуры данных "длинное число по модулю radix" назовем одновременное выполнение логических условий:

- $\text{radix} \geqslant 2$;
- $1 \leqslant \text{num\_digits} \leqslant \text{MAX\_NUM\_DIGITS}$;
- $0 \leqslant \text{digits[i]} < \text{radix}$ для $i = 0 \,..\, \text{num\_digits} - 1$;
- если $\text{num\_digits} > 1$, то $\text{digits}[\text{num\_digits} - 1] > 0$.

[3] Ахо А.В., Хопрофт Д., Ульман Д.Д. Структуры данных и алгоритмы. — Москва: Изд. дом "Вильямс", 2001.

Упражнение. Нужно ли добавить сюда условие равенства нулю всех элементов массива цифр с индексами больше num_digits − 1 или лучше считать эти значения неопределенными?

Первое условие — это проверка входных данных (заданного основания системы счисления) на корректность. При переполнении длинного числа (когда нарушено второе условие) реализующая соответствующую операцию функция должна возвращать сообщение об ошибке. Третье и четвертое условия должны выполняться автоматически, то есть не должны нарушаться в результате выполнения всех операций над длинными числами.

Алгоритм инициализации структуры данных "длинное число" состоит в переводе числа value в систему счисления по основанию radix.

Упражнение. Найдите и исправьте ошибку в следующем алгоритме, решающем указанную задачу для числа value $\geqslant 0$:

```
digits ← [ ]
x ← value  // x — начальный отрезок исходного числа
while x > 0 {
  digits.append(x mod radix)  // добавляем остаток от деления к массиву цифр
  x ← [x/radix]  // неполное частное: убираем одну цифру с конца
}
```

Сложение длинных чисел. Определим спецификацию функции сложения двух длинных чисел: на вход подается два длинных числа с одинаковыми значениями radix, находящихся в корректном состоянии. На выходе получаем третье длинное число в корректном состоянии, равное сумме двух данных чисел. Если произошло переполнение (число цифр в сумме больше MAX_NUM_DIGITS), то возвращается сообщение об ошибке.

Обсудим алгоритм сложения "в столбик". Поскольку алгоритм должен работать для любых корректных состояний a и b, то важно рассмотреть все возможные частные случаи. Основных случаев два:

1) количество цифр в результате равно максимальному количеству цифр в a и b;

2) количество цифр в результате на единицу больше, чем максимальное количество цифр в a и b,

а также третий случай:

3) один или оба слагаемых равны 0.

Получим формулы для цифр суммы чисел a и b, чтобы обосновать и лучше понять алгоритм сложения "в столбик".

Запишем представления чисел a и b в системе счисления по основанию m:

$$a = a_0 + a_1 m + \ldots + a_{k-1} m^{k-1},$$

$$b = b_0 + b_1 m + \ldots + a_{l-1} m^{l-1}.$$

Требуется найти представление числа $c = a + b$ в той же системе счисления, то есть такие числа $c_i \in 0..m-1$, $i = 0..s-1$, что

$$c = c_0 + c_1 m + \ldots + c_s m^{s-1},$$

причем если $s > 1$, то $c_{s-1} > 0$. Для этого сложим представления $a = \sum\limits_{i=0}^{k-1} a_i m^i$ и $b = \sum\limits_{i=0}^{l-1} b_i m^i$:

$$c = \sum_{i=1}^{p-1} (a_i + b_i) m^i,$$

где $p = \max\{k,l\}$ и считаем $a_i = 0$ при $i > k-1$ и $b_i = 0$ при $i > l-1$, и преобразуем сумму к виду $c = \sum\limits_{i=0}^{s-1} c_i m^i$, где $c_i \in 0..m-1$.

Разделим первое слагаемое на m:

$$a_0 + b_0 = qm + r.$$

Обозначим неполное частное $d_1 := q$, остаток $c_0 := r$. Другими словами, $d_1 = [(a_0 + b_0)/m]$, $c_0 = (a_0 + b_0) \bmod m$. Таким образом,

$$c = c_0 + (a_1 + b_1 + d_1)m + \sum_{i=2}^{p-1}(a_i + b_i)m^i =$$

$$= c_0 + m\left[(a_1 + b_1 + d_1) + \sum_{i=1}^{p-2}(a_{i+1} + b_{i+1})m^i\right].$$

Аналогичным образом можно выделить младшую цифру числа, стоящего в квадратных скобках: $c_1 = (c_1 + b_1 + d_1) \bmod m$, и найти неполное частное $d_2 = [(a_1 + b_1 + d_1)/m]$. Получим

$$c = c_0 + c_1 m + m^2\left[(a_2 + b_2 + d_2) + \sum_{i=1}^{p-3}(a_{i+2} + b_{i+2})m^i\right].$$

Этот процесс можно повторять до бесконечности, но рано или поздно цифры закончатся. Приходим к равенству (заметив, что сумма показателя степени при m и верхнего предела суммирования равна $p-1$)

$$c = c_0 + c_1 m + c_2 m^2 + \ldots + c_{p-2}m^{p-2} + m^{p-1}(a_{p-1} + b_{p-1} + d_{p-1}) =$$

$$= c_0 + c_1 m + \ldots + c_{p-1}m^{p-1} + d_p m^p,$$

где

$$c_i = (a_i + b_i + d_i) \bmod m, \quad i = 0..p-1,$$

$$d_0 = 0, \quad d_{i+1} = [(a_i + b_i + d_i)/m], \quad i = 0..p-1.$$

Заметим, что если $c_{p-1} = 0$, то $d_p = 1$, что дает оценку количества цифр в записи числа в m-ичной системе счисления.

Упражнения. 1. Реализуйте алгоритм сложения длинных чисел, используя полученные формулы.

2. Разберите случай переполнения массива цифр при записи результата сложения длинных чисел.

3. Нужно ли обнулять a_i и b_i при $i > k-1$ и $j > l-1$?

Вычитание длинных чисел. *Упражнение.* Реализуйте функцию, выполняющую сравнение двух длинных чисел.

Предположим, что $a \geqslant b$. Вычтем из представления $a = \sum\limits_{i=0}^{k-1} a_i m^i$ представление $b = \sum\limits_{i-0}^{l-1} b_i m^i$, получим $c = a - b = \sum\limits_{i=0}^{k-1}(a_i - b_i)m^i$. Теперь, если $a_i < b_i$, то заменим соответствующее слагаемое на $(m + a_i - b_i) - m$.

Обозначим $d_0 = 0$,

$c_i = a_i - b_i - d_i$, $d_{i+1} = 0$, если $a_i - b_i - d_i \geqslant 0$

и $c_i = m + (a_i - b_i - d_i)$, $d_{i+1} = 1$, если $a_i - b_i - d_i < 0$.

Тогда

$$(a_i - b_i - d_i) + (a_{i+1} - b_{i+1})m = c_i + (a_{i+1} - b_{i+1} - d_{i+1})m,$$

поэтому

$$c = a - b = a_0 - b_0 + \sum_{i=1}^{k-1}(a_i - b_i)m^i = c_0 + (a_1 - b_1 - d_1)m + \sum_{i=2}^{k-1}(a_i - b_i)m^i =$$

$$= c_0 + c_1 m + (a_2 + b_2 - d_2)m^2 + m^3 \sum_{i=0}^{k-4}(a_{i+3} - b_{i+3})m^i =$$

$$= c_0 + c_1 m + c_2 m^2 + \ldots + (a_{k-1} - b_{k-1} - d_{k-1})m^{k-1} = \sum_{i=0}^{k-1} c_i m^i.$$

Упражнение. Проверьте условие $0 \leqslant c_i < m$.

Итак, мы доказали, что, используя коэффициенты c_i и d_i, можно найти цифры разности $a - b$. Наши выкладки опирались на индуктивный переход, который с точки зрения построения алгоритмов означает рекурсию — вызов функцией самой себя с другим набором аргументов с хранением данных в стеке вызовов.

Упражнение. Сформулируйте рекурсивное определение наших вычислений по сведению преобразования последовательности цифр к преобразованию последовательности цифр меньшего размера. Функция принимает на вход два отрезка последовательностей $\{a_i\}$ и $\{b_i\}$, а также число d и возвращает отрезок последовательности $\{c_i\}$. Реализуйте это преобразование рекурсивно либо итерационно.

Умножение длинных чисел. Пусть $\overline{a} = (a_0, a_1, \ldots, a_{k-1})$, $\overline{b} = (b_0, b_1, \ldots, b_{l-1})$ — представления целых неотрицательных чисел a и b в системе счисления по основанию m. Требуется найти представление числа $c = ab$ в той же системе счисления.

По определению

$$b = b_0 + b_1 m + b_2 m^2 + \ldots + b_{l-1} m^{l-1}.$$

Тогда

$$c = ab = ab_0 + ab_1 m + ab_2 m^2 + \ldots + ab_{l-1} m^{l-1}.$$

Предположим, что для СД "длинное число по модулю m" определены операции:
- $c = \mathrm{add}(a,b)$ — сложение длинных чисел a,b;
- $c = \mathrm{mult\_short}(a,b)$ — умножение длинного числа a на обычное число b;
- $c = \mathrm{mult\_power}(a,j)$ — умножение длинного числа a на m^j (сдвиг массива цифр).

Алгоритм:

```
mult(a,b) → c = {
  l ← b.num_digits
  c ← 0
  for i ← 0 .. l − 1 {
    // c = ab_0 + ab_1 m + ... + ab_{i−1} m^{i−1}
    x ← mult_short(a, b[i])
    // x = ab_i
    x ← mult_power(x, i)
    // x = ab_i m^i
    c ← add(c, x)
    // c = ab_0 + ab_1 m + ... + ab_{i−1} m^{i−1} + ab_i m^i
  }
```

$// \; c = ab_0 + ab_1 m + \ldots + ab_l m^l = ab$

}

На выполнение алгоритма будет затрачено порядка kl единиц времени.

Упражнение. Реализуйте функцию для перевода представления длинного числа в системе счисления по основанию radix в представление того же числа в системе счисления по основанию new_radix.

Деление длинных чисел. Пусть $\overline{a} = (a_0, a_1, \ldots, a_{k-1})$ и $\overline{b} = (b_0, b_1, \ldots, b_{l-1})$ — два числа, записанные в m-ичной системе счисления. Требуется найти неполное частное и остаток от деления a на b. Как и при делении в столбик, алгоритм будет подбирать разряды частного от старших к младшим.

Обозначим $p = k - l$ и подберем наибольшее $c_p \in 0..m-1$, такое, что

$$a - bc_p m^p \geqslant 0.$$

Иными словами, число b сдвигается на p разрядов влево, умножается на коэффициент c_p и вычитается из числа a. Таким образом, $a - bc_p m^p \geqslant 0$, $a - b(c_p + 1)m^p < 0$, то есть

$$bc_p m^p \leqslant a < b(c_p + 1)m^p,$$

$$c_p m^p \leqslant \frac{a}{b} \leqslant (c_p + 1)m^p.$$

Заметим, что $a - bm^{p+1} = a - bm^{k-l+1} < 0$, так как $b \geqslant m^{l-1}$, $a < m^k$.

Затем повторим аналогичную процедуру для остатка $a - bc_p m^p$: найдем наибольшее $c_{p-1} \in 0..m-1$, для которого

$$a - b(c_p m^p + c_{p-1} m^{p-1}) \geqslant 0.$$

Таким образом,

$$a - b\left(c_p m^p + (c_{p-1} + 1)m^{p-1}\right) < 0,$$

$$0 \leqslant a - b(c_p m^p + c_{p-1} m^{p-1}) < bm^{p-1}.$$

Заметим, что $a - b(c_p m^p + m \cdot m^{p-1}) < 0$.

Аналогично, повторяя рассуждения, будем иметь

$$0 \leqslant a - b(c_p m^p + c_{p-1} m^{p-1} + \ldots + c_j m^j) < bm^j, \;\; j = 0..p.$$

При $j = 0$ получим

$$a = bq + r, \text{ где } q = \sum_{i=0}^{p} c_i m^i, \;\; 0 \leqslant r < b.$$

По теореме о делении с остатком получим, что неполное частное $[a/b] = \sum\limits_{i=0}^{p} c_i m^i$. При этом c_p может оказаться равным 0.

Опишем алгоритм схематично. В цикле по j от $p = k - l$ до $j = 0$ с шагом -1 будем вычислять слагаемые остатка от деления $a - b(c_p m^p + c_{p-1} m^{p-1} + \ldots + c_j m^j)$ и для каждого j подбирать наибольшее $c_j \in 0..m-1$, для которого данное выражение неотрицательно. Поскольку m может достигать больших значений, целесообразно применить для поиска искомого c_j двоичный поиск с временной сложностью $O(\log m)$. В качестве промежуточных данных вычисляем эту сумму (длинное число), к которой на каждой итерации цикла добавляется новое слагаемое, а также сумму в скобках. По завершении цикла всё это выражение будет остатком от деления, а величина в скобках — неполным частным.

Оценим время выполнения алгоритма. Величина $r - bc_j m^j$ вычисляется за время $O(k + l)$. Это время нужно умножить на число итераций цикла по j и на время выполнения двоичного поиска. Получим $O((k + l)(k - l + 1) \log m)$, или $O(k^2 \log m)$.

В заключение отметим, что для грамотной и безошибочной реализации вычислительной библиотеки необходимо продумывать каждую её строчку, четко формулируя спецификации всех функций, а также результаты промежуточных вычислений.

Лабораторная работа. Реализуйте библиотеку для работы с длинными числами. Снабдите текст своей программы описанием, облегчающим его чтение: укажите в комментариях спецификации всех функций (что на входе, что на выходе, какие условия должны быть выполнены для входа), опишите логический инвариант структуры данных, прокомментируйте результаты промежуточных вычислений. Протестируйте свою библиотеку на основе сравнения с результатами вычислений с обычными числами и посчитайте, например, факториалы натуральных чисел, числа Фибоначчи и т.п.

5.4.2 Обход графов

Кратчайшие пути в графе

Графом $G = \langle V, E \rangle$ называют конечное множество V вместе с введенным на нем отношением E. Отношение указывает, какие элементы множества (вершины графа) соединены ребром. Если отношение симметрично ($(u, v) \in E \Rightarrow (v, u) \in E$), то граф называется неориентированным. В общем случае граф ориентированный.

Чтобы представить эту математическую конструкцию в компьютерной памяти, достаточно закодировать множество E. Первый способ представления множества — это двоичный вектор, элементы которого в данном случае образуют двоичную матрицу, называемую матрицей смежности графа:

$$a_{ij} = \begin{cases} 1, & \text{если } (v_i, v_j) \in E, \\ 0, & \text{иначе.} \end{cases}$$

Здесь множество вершин пронумеровано: $V = \{v_1, v_2, \ldots, v_n\}$.

Другой способ представления множества — это список его элементов. В данном случае всё множество E разбивается на подмножества $E_i = \{(v_i, u) \in E\}$ ребер, идущих из каждой i-й вершины. Вместо подмножества E_i удобно рассмотреть множество вершин, смежных с i-й вершиной: $\mathrm{adj}[i] = \{u \in V : (v_i, u) \in E\}$. Таким образом, граф можно задать списками смежности: для каждой вершины v_i нужно перечислить элементы множества $\mathrm{adj}[i]$.

Допустим, что нам требуется найти кратчайший путь из вершины a в вершину b. Путь из a в b — это последовательность вершин графа, в которой из каждой вершины идет ребро в следующую по порядку вершину: $a = u_0, u_1, \ldots, u_{k-1}, u_k = b$, где $(u_i, u_{i+1}) \in E$ для $i = 0..k - 1$. Число k называется длиной пути.

Для перечисления вершин графа в порядке возрастания длины кратчайшего пути от источника s можно использовать обход графа в ширину. Вначале в очередь помещается вершина s и полагается $d[s] = 0$ и $d[v] = \infty$ для $v \neq s$. Затем на каждом шаге алгоритма из начала очереди извлекается вершина v_i и в конец очереди добавляются все вершины u из множества $\mathrm{adj}[i]$, для которых $d[u] = \infty$, полагая для всех этих вершин $d[u] \leftarrow d[v_i] + 1$. Процесс повторяется до тех пор, пока очередь не пуста.

Рассмотренный алгоритм основан на логическом инварианте: предполагается, что для всех вершин v, помещенных в очередь, вычислена длина кратчайшего пути, которая находится в переменной $d[v]$. Чтобы это обосновать, рассмотрим множества V_k вершин, для которых длина кратчайшего пути от вершины s равна k. Нетрудно заметить, что вершины

v_i в очереди расположены в порядке возрастания $d[v_i]$, так что сначала в очередь будет добавлено множество V_0, строго за ним — множество V_1, затем V_2 и так далее вплоть до максимального расстояния от источника. Когда мы извлекаем вершину из очереди, мы добавляем вслед за ней в очередь все вершины, для которых кратчайшее расстояние больше, чем расстояние до удаляемой вершины: в противном случае существует вершина с меньшим расстоянием, из которой в новую вершину идет ребро и которая должна была быть обработана раньше. Похожие рассуждения используются при анализе алгоритма Дейкстры, который вычисляет кратчайшие пути от источника до всех остальных вершин во взвешенном графе с неотрицательными весами ребер. Чтобы найти кратчайший путь, следует для каждой вершины v, добавляемой в очередь, запоминать вершину $p[v]$, из которой алгоритм пришел в вершину v. Время выполнения обхода в ширину составляет $O(n + m)$, где n — число вершин; m — число ребер графа.

Упражнение. Граф задан списками смежности. Перечислите связные компоненты графа: выведите в каждой строке список вершин, принадлежащих очередной связной компоненте. Эту же задачу можно решить и обходом в глубину.

Обход в глубину

Обход в глубину служит составной частью других алгоритмов на графах. Суть этого обхода состоит в построении остовного дерева с заданным корнем для связной компоненты графа, содержащей этот корень. Остовным деревом графа называют дерево, в котором множество вершин совпадает с множеством вершин графа, а множество ребер является подмножеством множества ребер графа.

Обход в глубину помещает новые вершины не в очередь, а в стек, например в стек вызовов рекурсивной функции. На каждом шаге алгоритма из стека извлекается вершина и в стек добавляются все вершины, смежные с ней, которые еще не были обработаны алгоритмом. Таким образом, получающееся при этом остовное дерево совпадает с деревом рекурсивных вызовов.

Обход в глубину позволяет выделить циклы в графе, проверить граф на ацикличность, осуществить топологическую сортировку ациклического графа, то есть упорядочить вершины так, чтобы ребра шли из предшествующих вершин к последующим, а также обход в глубину позволяет разложить ориентированный граф на сильно связные компоненты, в каждой из которых каждая вершина достижима из каждой.

Лабораторная работа. Реализуйте структуру данных “лабиринт со стенками”, которая содержит информацию о лабиринте, состоящем из ячеек, между которыми могут находиться горизонтальные и вертикальные стенки. По заданному начальному положению персонажа найдите путь до заданной конечной ячейки обходом в глубину и обходом в ширину. Реализуйте генерацию случайного лабиринта. Реализуйте отдельную структуру данных “граф” и сведите задачу поиска пути в лабиринте к задаче на графе. Затем добавьте в лабиринт порталы — пары ячеек, находящиеся далеко друг от друга, но между которыми можно провести ребро в графе. Если программа была спроектирована модульно, то решение такой задачи потребует внесения несущественных изменений в текст программы. Наконец, реализуйте поиск минимального пути во взвешенном графе, проходящего по всем вершинам, и примените эту реализацию к лабиринту, задав множество пунктов, которые надо посетить из заданной начальной ячейки, после чего вернуться в исходный пункт.

Обход дерева

Дерево — это множество узлов, соединенных ребрами в связный граф без циклов. Дерево с выделенным корнем обычно определяется как множество узлов, в котором для

каждого узла, кроме корня, указан предок. Таким образом, множество потомков узла определяется как множество всех узлов дерева, у которых предком является данный узел.

Различают прямой, обратный и симметричный обход дерева. Прямой обход сначала посещает корень дерева, а затем по порядку все поддеревья с корнем в потомках корня дерева. Симметричный обход сначала посещает поддерево с корнем в первом потомке, затем корень и только после этого все поддеревья с корнем в остальных потомках корня дерева. Обратный обход обходит в обратном порядке поддеревья с корнем в потомках корня дерева, после чего посещает корень дерева.

Чтобы реализовать структуру данных "дерево", нужно хранить в компьютерной памяти данные, связанные с узлами, и для каждого узла запомнить индекс его предка, а также список индексов всех его потомков.

Упражнение. Постройте дерево правильной скобочной последовательности.

5.4.3 Разбор выражений

Формальные языки

Алфавит — это непустое конечное множество символов. Цепочка в алфавите V — это прозвольный кортеж из множества V^k для различных $k = 0, 1, 2, \ldots$ Множество всех слов в алфавите V обозначаем V^*, множество всех непустых слов записываем как V^+.

Язык в алфавите V — это произвольное подмножество множества V^*. На множестве языков вводят алгебраические операции объединения, конкатенации и итерации. Итерацией языка называют объединение всех его степеней.

Регулярный язык может быть представлен как результат применения вышеуказанных операций к языкам, состоящим из одной цепочки с не более чем одним символом. Таким образом, множество регулярных языков — это подсистема алгебраической системы языков, порожденная множеством, состоящим из пустого языка, а также языка, состоящего из пустой цепочки, и всех языков, состоящих из односимвольных цепочек. Представляя операции над языками с помощью символов $+$, $\cdot$, *, $^+$, формируют регулярные выражения.

Для обработки цепочек регулярного языка применяют конечные автоматы. Автомат — это машина с конечным числом состояний, включающая в себя: алфавит входных символов, множество состояний, множество принимающих состояний, функцию переходов и выделенное начальное состояние. Всякому автомату соответствует регулярный язык, который он распознает, и, наоборот, всякий регулярный язык допускается некоторым конечным автоматом.

Вообще для описания формальных языков применяют формальные грамматики. Это понятие включает в себя: множество нетерминальных символов, множество терминальных символов, начальный нетерминал и множество правил вывода вида $\alpha \to \beta$, где α, β — цепочки, состоящие из нетерминалов и терминалов. Если все правила вывода имеют вид $A \to \alpha$, где A — нетерминал, то такую грамматику называют контекстно-свободной, соответствующий язык распознается автоматом с магазинной памятью. Если сильнее сузить класс правил до $A \to aB$, где A, B — нетерминалы, a — терминал, то получим регулярные грамматики, которые в точности соответствуют классу регулярных языков.

Лексический разбор

Как правило, задаче синтаксического анализа сопутствует аналогичная задача для более простой, регулярной, грамматики — задача лексического анализа, состоящая в разбиении строки на лексемы — элементарные единицы, входящие в правые части правил подстановки: `число, +, *, -, /, (, )`.

Для решения задачи лексического анализа применяют конечный автомат — это множество состояний вместе с функцией переходов, которое реализуется следующим образом: на первом этапе автомат находится в начальном состоянии, а затем, считывая очередной символ, переходит в одно из своих состояний. Например, после считывания цифры автомат переходит в состояние "чтение числа", а из этого состояния он перейдет, например, в состояние "плюс". При переходе в другое состояние выполняется функция-обработчик, которая выдает очередную лексему.

Лабораторная работа. Дана строка, содержащая последовательность диапазонов страниц либо отдельных страниц, разделенную запятыми, например: `1, 4-6, 9 , 10-15`. Требуется преобразовать строку в массив пар (a,b), где `a-b` – диапазон страниц (если $a = b$, то указана одна страница). Затем для полученного списка пар страниц нужно вывести инструкцию по двухсторонней печати.

Разбор состоит из двух этапов: лексический разбор — строка преобразуется в последовательность лексем: чисел, дефисов, запятых; синтаксический разбор — последовательность лексем преобразуется в искомый массив. Синтаксический разбор: считаем, что обработчик на каждом шаге разбора находится в одном из нескольких состояний. Состояние определяет поведение обработчика при считывании очередного символа, а именно: в какое состояние будет произведен переход и какая функция для преобразования символов в другие данные будет вызвана.

Выделим следующие состояния обработчика (конечного детерминированного автомата):

– S – стартовое состояние либо после запятой;
– N – число после запятой либо в начале строки;
– N- – число с дефисом;
– N-N – число после дефиса.

Автомат работает так:
(S, число) → N;
(S, ,) → ошибка: лишняя запятая;
(S, -) → ошибка: лишний дефис;
(N, число) → ошибка: два числа разделены пробелами;
(N, -) → N- ;
(N, ,) → S + вызов процедуры добавления нового диапазона в массив;
(N-, число) → N-N ;
(N-, -) → ошибка: два дефиса подряд;
(N-, ,) → ошибка: запятая после дефиса;
(N-N, число) → ошибка: ожидалась запятая;
(N-N, ,) → S + вызов процедуры добавления нового диапазона в массив;
(N-N, -) → ошибка: число между двумя дефисами.

Следует также рассмотреть поведение автомата, когда ему на вход поступает лексема "конец строки".

Синтаксический разбор

Определим узел дерева выражений как элемент одного из следующих типов данных: сумма, разность, произведение, частное, возведение в степень, отрицание, число. В листьях дерева находятся числа, а нелистовые узлы содержат операции, которые применяются к потомкам соответствующего узла, то есть к соответствующим подвыражениям.

```
// арифметические операции
enum op_t {
    OP_PLUS, OP_MINUS, OP_MULT, OP_DIV, OP_NEG, OP_POW
};

// реализация типа данных "выражение"
struct expr_t {
    int num;
    struct expr_t *arg1, *arg2;
    enum op_t op;
};

// реализация объекта "выражение"
struct expr_t *expr;  // указатель на корень дерева
```

Упражнение. Реализуйте функцию создания корневого узла дерева выражения с заданной операцией и заданными операндами (типа "выражение"). Реализуйте также функцию удаления дерева выражения из динамической памяти. Для этого выполните рекурсивный вызов этой же функции для поддеревьев, после чего удалите корневой узел.

При прямом обходе дерева выражений получаем префиксную форму выражения, при симметричном обходе — инфиксную, а при обратном — постфиксную (польскую).

Упражнение. По построенному дереву выражения выведите все три формы его записи, не забыв про расстановку скобок согласно приоритету операций.

Упражнение. Напишите программу, принимающую на вход обратную польскую запись выражения (например: `2 3 + 4 1 - *`) и вычисляющую его значение. Указание: используйте стек для хранения промежуточных результатов.

Разберем задачу синтаксического разбора строкового представления арифметического выражения. Множество всех допустимых выражений описывается так называемой формальной грамматикой, которая состоит из правил подстановки:

```
<выражение0> ::= число | ( <выражение4> )
<выражение1> ::= <выражение0> ^ <выражение1> | <выражение0>
<выражение2> ::= - <выражение1> | <выражение1>
<выражение3> ::= <выражение2> * <выражение3> |
     <выражение2> / <выражение3> | <выражение2>
<выражение4> ::= <выражение3> + <выражение4> |
     <выражение3> - <выражение4> | <выражение3>
```

Начиная с символа `<выражение4>`, можно каждый раз заменить один из символов на его определение, например:

```
<выражение4>  ->  <выражение3> + <выражение4>  ->
->  <выражение2> * <выражение3> + <выражение4>  ->
->  <выражение1> * <выражение3> + <выражение4>  ->
->  <выражение0> * <выражение3> + <выражение4>  ->
->  число * <выражение3> + <выражение4>  ->  и т.д.
```

Синтаксический вывод можно представить в виде синтаксического дерева, в узлах которого находятся символы, а потомки узла — это символы, составляющие замену этого символа по правилу подстановки. Задача синтаксического анализа состоит в восстановлении по строке ее синтаксического вывода, представленного в виде синтаксического дерева. Дерево выражений, построенное по строковому выражению, и будет синтаксическим деревом.

Разобьем задачу синтаксического анализа на подзадачи построения синтаксического вывода определенного символа. В нашем случае у нас 5 подзадач, по количеству символов, стоящих в левых частях правил подстановки. Для решения каждой подзадачи определим отдельную рекурсивную функцию, которая применяет правила подстановки к соответствующему символу, либо определим одну рекурсивную функцию, принимающую на вход параметр i — индекс символа `<выражение_i>`. Дерево рекурсивных вызовов, таким образом, восстановит синтаксическую структуру выражения.

Приложение
Языки программирования

Пусть я знаю,
Как это называется.
Но знаю ли я об этом?

Python

В языке Python имеются следующие встроенные структуры данных:
– целые числа;
– вещественные числа;
– строки;
– кортежи;
– списки;
– словари;
– множества.

Из перечисленных типов данных списки, словари, множества являются *изменяемыми* (mutable), в то время как числа, кортежи, строки *неизменяемы* (immutable).

Переменная является ссылкой на объект. При присваивании переменной другого объекта значение ссылки изменяется. Это значит, что если одному списку присвоить другую переменную, ссылающуюся на список, то после выполнения присваивания обе переменные станут ссылаться на один и тот же список.

Чтобы определить список, следует заключить его элементы, перечисленные через запятую, в квадратные скобки: `[1, 2, 3, 5]`. Точно так же можно определить кортеж, заключив перечисление элементов в круглые скобки: `(1, 2, 3, 5)`. Если заключить перечисление элементов в фигурные скобки, получим множество: `{1, 2, 3, 5}`. Наконец, если связать с каждым перечисленным элементом какое-нибудь значение (например, имя), получим словарь: `{1: ''Alice'', 2: ''Bob'', 3: ''Vasya'', 5: ''Petya''}`.

Пакет `logics`[4] предоставляет функции для работы с логическими формулами.

Подробно о языке Python можно прочесть в литературе[5]. Рекомендуется также ознакомиться с курсами Института биоинформатики на платформе Stepik[6].

Автоматический прувер Lean

Lean[7] представляет собой язык программирования, позволяющий записывать формальные рассуждения о математических объектах. В частности, на языке Lean можно записать и проверить вывод формулы исчисления высказываний.

Логическое программирование на Prolog

Язык Пролог — это частный случай логического программирования, а именно, это реализация метода резолюций в логике первого порядка для хорновского случая со стратегией “поиск от целей” [5].

[4]URL: https://logics.readthedocs.io/

[5]Лутц М. Изучаем Python. — Санкт-Петербург: ООО “Диалектика”, 2019.
Любанович Б. Простой Python. Современный стиль программирования. — Санкт-Петербург: Питер, 2017.
Северанс Ч.Р. Python для всех. — Москва: ДМК Пресс, 2021.

[6]URL: https://stepik.org/org/bioinf

[7]URL: https://avigad.github.io/logic_and_proof/

Фраза Хорна — это дизъюнкт, содержащий не более одного положительного литерала. Иначе говоря, у всех переменных (возможно, кроме одной) имеется отрицание. Правило резолюции, примененное к двум хорновским дизъюнктам, приводит к их логическому следствию, которое также является хорновским дизъюнктом.

Программа на языке Prolog содержит элементы, напоминающие формализацию предметной области в логике предикатов. Это множества (домены), предикаты, а также факты и правила, сформулированные в виде клауз (дизъюнктов). Эти элементы составляют базу знаний. Чтобы ею воспользоваться, необходимо задать цель (запрос). Интерпретатор будет пытаться вывести цель из имеющихся формул, пользуясь методом резолюций.

Подробнее ознакомиться с языком Prolog можно в [10].

Заключение

Видишь в коде
Одни только буквы?
А как же модели?

Компьютерная логика нужна для того, чтобы грамотно подойти к построению вычислительных алгоритмов и формальных моделей.

Любая задача обработки информации может быть рассмотрена как на уровне кода, что приводит к излишней детальности, так и на языке логики. В последнем случае те же самые абстракции становятся короче и понятнее. Например, проблема построения навигационного приложения требует сначала аккуратного проектирования модели объекта, которая последовательно детализируется вплоть до формальной спецификации, а эта спецификация уже реализуется в программном коде. Таким образом, процессы проектирования и программирования идут совместно.

Мы выяснили, что формальная спецификация явно или неявно сопровождает всякий фрагмент программы. Программный код — это не только буквы, но и математические модели, с помощью которых происходят вычисления. Чтобы сформулировать спецификацию, нужны логические конструкции, которые также помогают строить формальные модели предметных областей. Работа с ними требует алгебраического навыка — для быстрого преобразования (в уме и на бумаге) арифметических и логических выражений, и навыка формального вывода — для эффективных логических умозаключений.

Логика предикатов — универсальный инструмент для описания состояния вычислений. Метод Флойда позволяет явно формулировать инварианты циклов и утверждения, которые должны выполняться в опорных точках программы. Программы читаются между строк, и результаты промежуточных вычислений формулируются на языке логики предикатов.

У студента первого курса спрашивают: “За какой промежуток времени можно выучить матлогику?”. Он подумал и ответил: “Примерно за три месяца”.

Студент выпускного курса в ответ на этот вопрос переспрашивает: “А методичка есть?”, — “Конечно”, — “Тогда дай полчаса, и пойдем сдавать”.

Библиографический список

Основная литература

1. Гуц, А.К. Математическая логика и теория алгоритмов: учеб. пособие. / А.К. Гуц. — Москва: ЛЕНАНД, 2023.

2. Крупский, В.Н. Математическая логика и теория алгоритмов: учеб. пособие для студ. учреждений высш. проф. образования / В.Н. Крупский, В.Е. Плиско. — Москва: Изд. центр «Академия», 2013.

3. Лавров, С.С. Программирование. Математические основы, средства, теория / С.С. Лавров. — Санкт-Петербург: БХВ-Петербург, 2002.

4. Непейвода, Н.Н. Прикладная логика: учеб. пособие / Н.Н. Непейвода. — Новосибирск: Изд-во Новосиб. ун-та, 2000.

5. Новиков, Ф.А. Символический искусственный интеллект: математические основы представления знаний: учеб. пособие для вузов / Ф.А. Новиков. — Москва: Юрайт, 2023.

Дополнительная литература

6. Белов, Ю.А. Лекции по математической логике и теории алгоритмов: учеб. пособие / Ю.А. Белов, В.А. Соколов. — Ярославль: ЯрГУ, 2013.

7. Вайнштейн, Ю.В. Математическая логика и теория алгоритмов: учеб. пособие / Ю.В. Вайнштейн, Т.Г. Пенькова, В.И. Вайнштейн. — Красноярск: Сиб. федер. ун-т, 2019.

8. Верещагин, Н.К. Лекции по математической логике и теории алгоритмов / Н.К. Верещагин, А. Шень. — Москва: МЦНМО, 2017.

9. Гашков, С.Б. Дискретная математика: учебник и практикум для вузов / С.Б. Гашков, А.Б. Фролов. — Москва: Юрайт, 2024.

10. Гринченков, Д.В. Математическая логика и теория алгоритмов для программистов: учеб. пособие / Д.В. Гринченков, С.И. Потоцкий. — Москва: КНОРУС, 2010.

11. Дехтярь, М.И. Лекции по дискретной математике: учебник / М.И. Дехтярь, С.М. Дудаков, Б.Н. Карлов. — Тверь: Твер. гос. ун-т, 2021.

12. Ерусалимский, Я.М. Дискретная математика. Теория и практикум: учебник / Я.М. Ерусалимский. — Санкт-Петербург: Лань, 2022.

13. Ефремов, Е.Л. Алгоритмы вычисления частичных функций: учебно-методическое пособие / Е.Л. Ефремов. — Владивосток: Изд-во Дальневост. федерал. ун-та, 2021.

14. Журавлев, Ю.И. Дискретный анализ. Формальные системы и алгоритмы: учеб. пособие для вузов / Ю.И. Журавлев, Ю.А. Флеров, М.Н. Вялый. — Москва: Юрайт, 2024.

15. Игошин, В.И. Математическая логика: учеб. пособие / В.И. Игошин. — Москва: ИНФРА-М, 2024.

16. Игошин, В.И. Теория алгоритмов: учеб. пособие / В.И. Игошин. — Москва: ИНФРА-М, 2019.

17. Карпов, Ю.Г. Теория автоматов / Ю.Г. Карпов. — Санкт-Петербург: Питер, 2003.

18. Лихтарников, Л.М. Математическая логика. Курс лекций. Задачник-практикум и решения: учеб. пособие / Л.М. Лихтарников, Т.Г. Сукачева. — Санкт-Петербург: Лань, 2009.

19. Мальцев, И.А. Дискретная математика: учеб. пособие / И.А. Мальцев. — Санкт-Петербург: Лань, 2011.

20. Матрос, Д.Ш. Теория алгоритмов: учебник / Д.Ш. Матрос, Г.Б. Поднебесова. — Москва: БИНОМ. Лаборатория знаний, 2008.

21. Матросов, В.Л. Математическая логика: учебник для бакалавриата / В.Л. Матросов, М.С. Мирзоев. — Москва: Прометей, 2020.

22. Пруцков, А.В. Математическая логика и теория алгоритмов: учебник / А.В. Пруцков, Л.Л. Волкова. — Москва: КУРС: ИНФРА-М, 2016.

23. Рублев, В.С. Основы теории алгоритмов: учеб. пособие для студ. вузов / В.С. Рублев. — Москва: Научный мир, 2008.

24. Рыбин, С.В. Дискретная математика и информатика: учебник для вузов / С.В. Рыбин. — Санкт-Петербург: Лань, 2022.

25. Скорубский, В.И. Математическая логика: учебник и практикум для вузов / В.И. Скорубский, В.И. Поляков, А.Г. Зыков. — Москва: Юрайт, 2022.

26. Судоплатов, С.В. Дискретная математика: учебник и практикум для вузов / С.В. Судоплатов, Е.В. Овчинникова. — Москва: Юрайт, 2024.

27. Судоплатов, С.В. Математическая логика и теория алгоритмов: учебник и практикум для вузов / С.В. Судоплатов, Е.В. Овчинникова. — Москва: Юрайт, 2024.

28. Супрун, В.П. Основы математической логики: учеб. пособие / В.П. Супрун. — Москва: ЛЕНАНД, 2017.

29. Хаггард, Г. Дискретная математика для программистов: учеб. пособие / Г. Хаггард, Дж. Шлипф, С. Уайтсайдс. — Москва: БИНОМ. Лаборатория знаний, 2010.

30. Хаггарти, Р. Дискретная математика для программистов / Р. Хаггарти. — Москва: Техносфера, 2012.

31. Шапорев, С.Д. Математическая логика: курс лекций и практических занятий / С.Д. Шапорев. — Санкт-Петербург: БХВ-Петербург, 2005.

32. Шиханович, Ю.А. Минимум по теории алгоритмов: для нематематиков: учеб. пособие / Ю.А. Шиханович. — Москва: ЛЕНАНД, 2021.

33. Aho, A.V. Foundations of computer science: C edition / A.V. Aho, J.D. Ullman. — W. H. Freeman, 1994.

34. Avigad, J. Mathematical logic and computation / J. Avigad. — Cambridge University Press, 2023.

35. Ben-Ari, M. Mathematical logic for computer science / M. Ben-Ari. — Springer, 2012.

36. Huth, M. Logic in computer science / M. Huth, M. Ryan. — Cambridge, 2004.

37. Leary, C.C., Kristiansen L. A friendly introduction to mathematical logic / C.C. Leary, L. Kristiansen. — 2015.

38. O'Donnell, J., Hall C., Page R. Discrete mathematics using a computer / J. O'Donnell, C. Hall, R. Page. — Springer, 2006.

39. Rosen, K.H. Discrete mathematics and its applications / K.H. Rosen. — McGraw-Hill, 2007.

Задачники

40. Борзунов, С.В. Задачи по дискретной математике с алгоритмами на Python / С.В. Борзунов, С.Д. Кургалин. — Санкт-Петербург: БХВ-Петербург, 2022.

41. Валединский, В.Д. Методы программирования в задачах и примерах на C/C++: учеб. пособие / В.Д. Валединский, А.А. Корнев. — Москва: Изд-во Московского ун-та, 2023.

42. Дехтярь, М.И. Задачник по дискретной математике: учеб. пособие / М.И. Дехтярь, С.М. Дудаков, Б.Н. Карлов — Тверь: Твер. гос. ун-т, 2021.

43. Иванов, М.К. Алгоритмический тренинг. Решения практических задач на Python и C++ / М.К. Иванов. — Санкт-Петербург: БХВ-Петербург, 2023.

44. Игошин, В.И. Сборник задач по математической логике и теории алгоритмов: учеб. пособие / В.И. Игошин. — Москва: КУРС: ИНФРА-М, 2019.

45. Козлова, М.Г. Задания и упражнения по курсу «Математическая логика и теория алгоритмов»: учебно-метод. пособие / М.Г. Козлова. — Симферополь, 2016.

46. Садыков, А.В. Основы математической логики: учеб. пособие / А.В. Садыков. — Санкт-Петербург: «Свое издательство», 2018.

47. Степанова, А.А. Математическая логика и теория алгоритмов: практикум / А.А. Степанова, Т.Ю. Плешкова, Е.Г. Гусев. — Владивосток: Изд-во ВГУЭС, 2010.

48. Степанова, А.А. Основы математической логики в примерах и задачах: учеб. пособие / А.А. Степанова. — Владивосток: Изд-во ДВФУ, 2020.

49. Степанова, А.А. Основы теории алгоритмов в примерах и задачах: учебно-метод. пособие / А.А. Степанова, С.Г. Чеканов. — Владивосток: Изд-во ДВФУ, 2020.

50. Шишмарёв, Ю.Е. Дискретная математика: сборник задач / Ю.Е. Шишмарёв, Е.Д. Емцева, К.С. Солодухин. — Владивосток: Изд-во ВГУЭС, 2000. — Ч. 1.

51. Gonczarowski, Y.A. Mathematical logic through Python / Y.A. Gonczarowski, N. Nisan. — Cambridge University Press, 2022.

Электронное учебное издание

Гренкин Глеб Владимирович

ОСНОВЫ КОМПЬЮТЕРНОЙ ЛОГИКИ

Учебное пособие