Владимир Пароннджанов

Алгоритмы и жизнеритмы

на языке ДРАКОН

Разработка алгоритмов
Безошибочные алгоритмы

Паронджанов В. Д.
Алгоритмы и жизнеритмы на языке ДРАКОН. Разработка алгоритмов. Безошибочные алгоритмы. — М., 2019. — 374 с. — Иллюстраций: 195.
ISBN

Улучшенные блок-схемы (дракон-схемы) позволяют быстро и без усилий изучить алгоритмы и жизнеритмы. Рассмотрены линейные, разветвленные, циклические и параллельные алгоритмы с примерами в виде наглядных и легко запоминающихся чертежей. Эргономичные дракон-алгоритмы, понятные с первого взгляда, помогут быстро освоить секреты мастерства. Даны примеры бизнес-процессов, потоков работ, клинических алгоритмов. Курс алгоритмической логики изложен с помощью удобных и привлекательных чертежей. Представлены алгоритмы реального времени и новый перспективный метод программирования без ошибок. Двести элегантных рисунков и схем помогут читателям самостоятельно создавать алгоритмы и жизнеритмы.

Для начинающих программистов, непрограммистов, программистов-любителей, студентов, бизнесменов и топ-менеджеров.

Новая система тщательно разработанных графических схем (дракон-схем) поможет вам легко и быстро освоить удивительное и прекрасное искусство алгоритмизации. Графика пробуждает творческие способности и позволяет придумывать и проектировать алгоритмы, жизнеритмы, и бизнес-процессы в любой области человеческих знаний с высоким качеством и без ошибок

КРАТКОЕ СОДЕРЖАНИЕ

Оглавление

Часть 6. Алгоритмы и жизнеритмы

Часть 7. Безошибочные алгоритмы и подавление ошибок

Часть 8. Стандарт на алгоритмы устарел и нуждается в замене

Заключение. Алгоритмы — важная часть человеческой культуры

ОГЛАВЛЕНИЕ

Часть 8. Стандарт на алгоритмы устарел и нуждается в замене

Глава 35. Сравнение дракон-схем и блок-схем

Глава 36. Критический анализ блок-схем алгоритмов по ГОСТ 19.701-90

Глава 37. Каким должен быть стандарт на алгоритмы

Введение

АЛГОРИТМЫ — ЭТО ПРОСТО

ПОМОЩЬ УЖЕ В ПУТИ

Есть много хороших книг по алгоритмам, много замечательных учебников. К сожалению, они очень трудные и сложные — придется затратить много сил, труда и времени. А нельзя ли найти другой путь?

Пора забыть о трудностях! Помощь уже в пути. Перед вами легкая и приятная книга, которую можно изучить быстро и без хлопот — вас выручат наглядные чертежи, которые называются *дракон-схемы.*

Алгоритмы могут пригодиться всем или почти всем. Начиная от врача и агронома и кончая топ-менеджером. Мы живем в мире алгоритмов, хотя зачастую не догадываемся об этом. Современный мир — это мир алгоритмов. Они окружают нас повсюду — и дома, и на работе.

Познакомиться с алгоритмами нам поможет язык графических фигур ДРАКОН.

ДРАКОН РОДИЛСЯ В КОСМИЧЕСКОЙ КОЛЫБЕЛИ

Язык ДРАКОН (DRAKON) разработан совместными усилиями Федерального космического агентства России (Научно-производственный центр автоматики и приборостроения им. академика Н. А. Пилюгина, г. Москва) и Российской академии наук (Институт прикладной математики им. М. В. Келдыша, г. Москва).

ДРАКОН возник как обобщение опыта работ по созданию легендарного космического корабля «Буран». На базе ДРАКОНа построена автоматизированная технология проектирования алгоритмов и программ под названием «ГРАФИТ-ФЛОКС» [1] [2]. Она успешно используется во многих крупных ракетно-космических проектах: «Ангара», «Фрегат», «Морской старт» и др.

ПОЧЕМУ АЛГОРИТМЫ ТРУДНЫ ДЛЯ ПОНИМАНИЯ

Существующий способ записи алгоритмов (принятый во всем мире) выбран неудачно. Он устарел и превратился в тормоз развития науки. Он удовлетворяет требованиям математической строгости, но не учитывает требования науки о человеческих факторах — эргономики. Этот устаревший способ упускает из виду психофизиологические характеристики человека. И тем самым затрудняет и замедляет работу с алгоритмами.

В книге излагается новый взгляд на алгоритмы. Мы покажем, что алгоритмы должны быть не только правильными, но и *дружелюбными* (people-friendly). То есть легкими для понимания и удобными для работы.

Язык ДРАКОН создает повышенный интеллектуальный комфорт, увеличивает продуктивность труда. С его помощью вы научитесь легко и быстро, затратив минимум усилий, создавать большие и сложные алгоритмы.

ОРУЖИЕ ИНТЕЛЛЕКТА

Визуальный язык ДРАКОН обладает уникальными эргономическими характеристиками. В отличие от блок-схем, дракон-схемы сохраняют наглядность даже для сложных многостраничных алгоритмов.

ДРАКОН — оружие интеллекта. Он помогает ясно и четко мыслить. И значительно облегчает творческий процесс создания алгоритмов, делая его доступным для широкого круга работников.

При создании орбитального корабля Буран автор книги был начальником лаборатории комплексной разработки вычислительной системы Бурана

АЛГОРИТМИЧЕСКАЯ ЛОГИКА

Важную роль в алгоритмах играет логика. Многие люди с трудом понимают сложные логические выражения, содержащие логические операции И, ИЛИ, НЕ. Это порождает досадные ошибки. Погрешности в алгоритмах приводят к печальным последствиям. Как избавиться от подобных неприятностей?

В языке ДРАКОН логические операции изображаются по-новому — с помощью удобной и прозрачной инфографики. Графика дает возможность удалить логические связки И, ИЛИ, НЕ. И за счет этого предотвратить многие ошибки.

Можно ли защитить алгоритмы от логических дефектов? Курс алгоритмической логики языка ДРАКОН позволит читателям узнать много интересного.

АЛГОРИТМЫ И ЖИЗНЕРИТМЫ

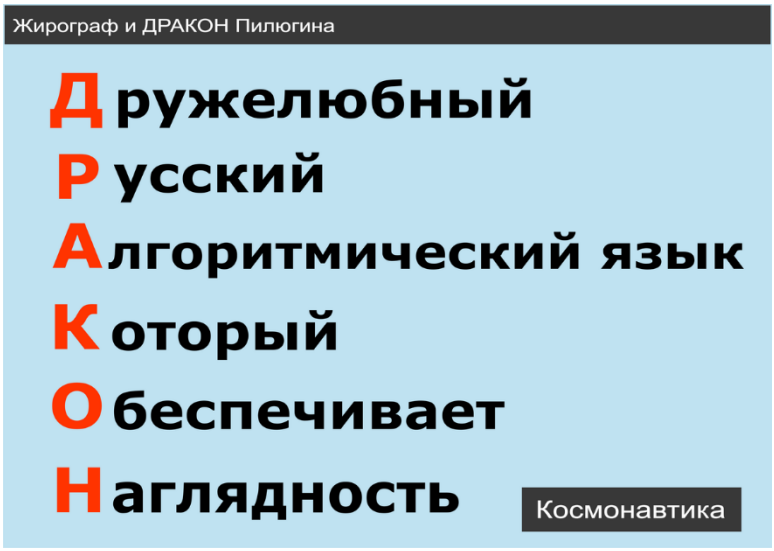
Алгоритмы бывают двух сортов: строгие алгоритмы и нестрогие (жизнеритмы). Первые выполняются автоматически и используются в компьютерных программах, вторые выполняются только людьми.

Произошел несчастный случай. Остановилась сердце, наступила клиническая смерть. Дорога каждая секунда. Если рядом окажется врач, человек будет спасен. Почему? Потому что врач знает алгоритм реанимации. Этот спасительный алгоритм и есть *жизнеритм.*

Жизнеритмы бывают не только в медицине. Они часто встречаются и в других областях. Жизнеритм — любая последовательность человеческих действий, например технология выращивания огурцов в теплице. Все бизнес-процессы — тоже жизнеритмы.

КАК УСТРАНИТЬ ОШИБКИ В АЛГОРИТМАХ

Важная цель языка ДРАКОН — сократить число ошибок в алгоритмах и программах. Для этого текстовые операторы управления вычислительным процессом заменяются на управляющую графику (чертежи). Ошибки в графических циклах и разветвлениях исчезают. Мелкие огрехи могут сохраниться только на линейных участках программы, где их легко найти и устранить.



Телестудия Роскосмоса. Кадр из фильма «Жирограф и ДРАКОН Пилюгина»

Кроме того, для предотвращения ошибок используется специальный математический аппарат — визуальное логическое исчисление (исчисление икон).

Управляющая графика ДРАКОНа является мощным инструментом, причем ее мощь легка в освоении и легко применима на практике.

ДРАКОН обладает надежной защитой от алгоритмических ошибок. Вследствие этого вероятность появления ошибок в дракон-программах значительно снижается.

АЛГОРИТМЫ — ЧАСТЬ ПРОФЕССИОНАЛЬНОЙ КУЛЬТУРЫ

Очень важно научиться самостоятельно создавать эргономичные, то есть легкие для понимания алгоритмы. И показать, что это простое и даже приятное дело. На многочисленных примерах читатель убедится, что дружелюбные алгоритмы имеют огромные преимущества.

Умение писать и понимать алгоритмы — настоятельная необходимость. Такое умение должно стать частью профессиональной культуры специалистов почти всех специальностей. Эта мысль проходит красной нитью через всю книгу.

КЛИНИЧЕСКИЕ АЛГОРИТМЫ

В медицине существуют подробные правила диагностики и лечения больных. На жаргоне медиков такие правила называются *клиническими алгоритмами*. Хотя фактически они являются не алгоритмами, а всего лишь *жизнеритмами.*

Проблема в том, что в подобных «алгоритмах» нет необходимой точности, нет однозначности. Неопределенность в медицинской литературе может привести к непониманию и является глубинным источником многих врачебных ошибок.

Необходимо кардинально повысить качество клинических алгоритмов, уменьшить неопределенность, снизить вероятность врачебных ошибок и обеспечить безопасность пациентов.

Создать клинические алгоритмы *высокой точности* — значит приблизить их свойства к идеалу – то есть к свойствам алгоритмов. Это позволит предотвратить врачебные ошибки и спасти многие жизни.

Дело усугубляется тем, что у практикующих врачей не развито алгоритмическое мышление. Напрашивается неутешительный вывод. Налицо серьезные системные дефекты в ныне существующей практике описания клинических алгоритмов. Язык ДРАКОН позволяет успешно решить данную проблему; он помогает врачам исключать ошибки при выполнении клинических алгоритмов и обеспечить безопасность пациентов.

СТАНДАРТ НА АЛГОРИТМЫ УСТАРЕЛ И НУЖДАЕТСЯ В СОВЕРШЕНСТВОВАНИИ

Проблема стандартизации алгоритмов имеет большое значение. Стандарт на блок-схемы алгоритмов ГОСТ 19.701-90 отстал от жизни и не удовлетворяет современным требованиям. Данный стандарт оказывает негативное воздействие на систему среднего и высшего образования России.

С появлением дракон-схем (drakon-charts) блок-схемы алгоритмов по ГОСТ 19.701-90 полностью потеряли свое значение

На основании тщательного анализа сделан вывод:

- блок-схемы алгоритмов (flowcharts), описанные в ГОСТ 19.701-90 и международном стандарте ISO 5807:85, обладают серьезными дефектами; пользоваться ими недопустимо;
- вместо блок-схем для записи алгоритмов следует использовать дракон-схемы (drakon-charts);
- необходимо разработать и выпустить отдельный стандарт на алгоритмы на основе языка ДРАКОН.

ПРОГРАММА «ДРАКОН-КОНСТРУКТОР»

Если рисовать дракон-схему с помощью обычного графического редактора, мы получим картинку, но не программу. Кроме того, будет много ошибок, а рисование займет много времени. Чтобы разработать программу, исключить ошибки и ускорить процесс, необходимо использовать программу «ДРАКОН-конструктор». Программа предназначена для быстрого создания и редактирования дракон-алгоритмов с использованием математических и эргономических инструментов, специально разработанных для безошибочной работы.

Язык ДРАКОН содержит большое число правил. Однако их не надо учить и запоминать. Почему? Потому что правила знает, хранит в своей памяти и скрупулезно выполняет программа ДРАКОН-конструктор.

ДРАКОН-конструктор – ваш надежный помощник. Он умело подсказывает, как нужно составлять алгоритмы. контролирует каждый ваш шаг, не дает оступиться и сбиться с пути.

ГДЕ СКАЧАТЬ ДРАКОН-КОНСТРУКТОР

Ответ дан в главе 34.

СТРУКТУРА КНИГИ

Книга состоит из восьми частей.

Часть I (главы 1—6) носит ознакомительный характер. Изучаем забавный, но требующий смекалки алгоритм «Как научить попугая говорить». И заодно осваиваем основные понятия и приемы. Изучаем графические фигуры языка ДРАКОН и применяем их на практике.

Часть II (главы 7—11) описывает визуальные циклические алгоритмы.

В части III (главы 12—18) изложен курс алгоритмической логики.

Часть IV (главы 19, 20) посвящена программам реального времени и параллельным процессам.

Часть V (главы 21—26) посвящена увлекательным примерам. Представлено большое число алгоритмов на языке ДРАКОН, взятых из практической жизни. Примеры демонстрируют универсальность языка, показывают широкий спектр его возможностей для различных отраслей и предметных областей. Сюда относятся медицина, промышленность, сельское хозяйство, торговля и пр.

В части VI (главы 27—29) проводим анализ понятия «алгоритм» и показываем, что данный термин используется некорректно — для обозначения двух существенно разных понятий. Для нестрогих алгоритмов вводится термин «жизнеритм». Излагается теория жизнеритмов, позволяющая улучшить их качество. Язык ДРАКОН позволяет превратить некачественные жизнеритмы в жизнеритмы высокой точности.

Предлагается принять язык ДРАКОН в качестве медицинского стандарта для предотвращения врачебных ошибок и наиболее удобного изображения клинических алгоритмов, а также включить указанный язык в систему медицинского образования. Цель состоит в том, чтобы содействовать преобразованию клинической медицины в алгоритмическую клиническую медицину.

В части VII (главы 30—34) учимся программировать без ошибок. Для производства безошибочных алгоритмов предлагается эргономичная нотация, обеспечивающая высокую понимаемость алгоритмов и удовлетворяющая требованиям когнитивной эргономики.

Одна из целей ДРАКОН-методологии — сократить число ошибок в алгоритмах и программах. Для этого используется метод «подавление ошибок», который включает четыре направления: эргономичную нотацию; алгоритмическую логику; исчисление икон; ДРАКОН-конструктор. Подробно рассматриваются указанные направления.

Важную роль в подавлении ошибок играет ДРАКОН-конструктор. Исчисление икон реализовано во внутренних алгоритмах ДРАКОН-конструктора. Конструктор осуществляет частичное автоматическое доказательство правильности графики дракон-схем.

Часть VIII (глава 35—37) содержит подробную и аргументированную критику стандарта на блок-схемы алгоритмов. Показано, что стандарт устарел и подлежит замене на новый стандарт, основанный на языке ДРАКОН.

ЧТО ГОВОРЯТ В СЕТИ ИНТЕРНЕТ

> «Если нужно рисовать алгоритм, теперь только и только на Драконе. Считаю, что он должен стать государственным стандартом для блок-схем вместо существующего. Удивительно, что авторы книг продолжают использовать прежние схемы, на которые после Дракона без ужаса смотреть невозможно» *(dvuugl)*.

«Это лучший язык для алгоритмов, ребята. Если вам нужно спроектировать алгоритм, вам нужен ДРАКОН. Любые системы проектируются на раз» *(Сергей Сторожев).*

«ДРАКОН позволяет объяснять сложные концепции другим людям» *(Alan Lucero)*.

«Язык Дракон — это способ визуального описания алгоритмов, исключающий ошибки» *(vtral)*.

Как связаться с автором

Владимир Данилович Пароннджанов
Mobile: +7-916-111-91-57
Viber: +7-916-111-91-57
E-mail: vdp2007@bk.ru
Skype: vdp2007@bk.ru
Website: http://drakon.su/
Webforum: http://forum.drakon.su/


dalmatkowy.xx.pl

Часть 1

МАТЕМАТИЧЕСКИЙ ЧЕРТЕЖ АЛГОРИТМА

Глава 1

КАК НАРИСОВАТЬ АЛГОРИТМ

КЛАССИЧЕСКИЙ ПУТЬ ИЗУЧЕНИЯ АЛГОРИТМОВ

Наша цель — помочь читателю познакомиться с алгоритмами и овладеть искусством алгоритмизации. Классический путь для этого — записывать алгоритмы в виде текста, псевдокода. Вся литература по алгоритмам и все современные учебники используют классический подход. Это надежный, многократно проверенный и хорошо зарекомендовавший себя способ.

Однако есть и недостаток — классический путь предъявляет к учащимся повышенные требования. Чтобы овладеть алгоритмами, придется упорно работать и затратить на изучение много труда и времени. Все, кому не хватит упорства или способностей, неизбежно останутся за бортом.

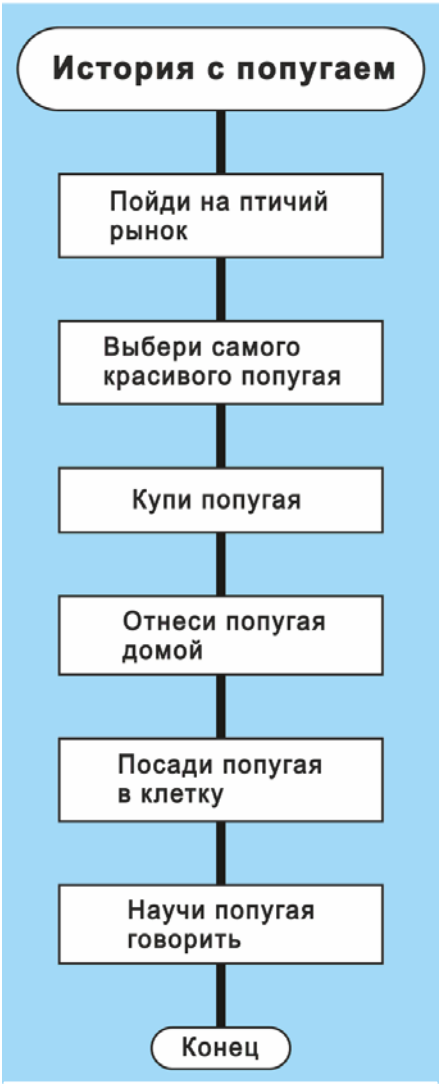


Рис. 1. Алгоритм «История с попугаем»

НЕКЛАССИЧЕСКИЙ ПОДХОД: ЧЕРТЕЖИ АЛГОРИТМОВ ВМЕСТО ТЕКСТА

Возможен ли другой, более гуманный путь к цели? Можно ли овладеть алгоритмами, не принося лишних жертв богу науки? Можно ли снизить слишком высокую нагрузку на учащихся?

Да, все это возможно. Мы будем использовать неклассический подход — записывать алгоритмы не в виде трудного текста, а в форме наглядных чертежей. Они похожи на блок-схемы алгоритмов, но имеют два принципиальных отличия.

Во-первых, они имеют математический характер. Во-вторых, математические чертежи алгоритмов (дракон-схемы) построены по законам когнитивной эргономики, что позволяет создавать эргономичные алгоритмы.

Математические чертежи алгоритмов и эргономичные алгоритмы выражают сущность неклассического подхода. Они позволяет значительно облегчить и ускорить изучение алгоритмов. В книге представлены 200 иллюстраций (дракон-схем), которые помогают быстро уяснить сложный материал.

ИКОНЫ

Упрощенно говоря, алгоритм — это последовательность действий. А что такое действие? Никлаус Вирт объясняет:

«Самым важным понятием является понятие *действия*. Здесь под действием понимается нечто, что имеет конечную продолжительность и приводит к желаемому и совершенно определенному результату» [3].

На рисунке 1 показан простейший алгоритм. Читать его нужно сверху вниз. Он содержит шесть действий:

- Пойди на птичий рынок.
- Выбери самого красивого попугая.
- Купи попугая.
- Отнеси попугая домой.
- Посади попугая в клетку.
- Научи попугая говорить.

Действия выполняются последовательно, по очереди. Сначала одно, потом другое, затем третье.

Время направлено вертикально вниз. Алгоритм нарисован по принципу «Чем ниже, тем позже». Например, действие «Отнеси попугая домой» выполняется позже, чем действие «Купи попугая».

Каждая фигура на рис. 1 называется *иконой,* от английского слова *icon.*

Прямоугольная икона именуется «Действие». В иконах Действие пишут команды. Первым словом в команде должен быть глагол в повелительном наклонении:

- Пойди
- Выбери
- Купи
- Отнеси
- Посади
- Научи

ЧТО ТАКОЕ ШАГ АЛГОРИТМА

Алгоритм делится на мелкие порции — шаги, которые следуют друг за другом (шаг за шагом).

На рисунке 2 показаны шаги алгоритма. Мы видим, что не все фигуры являются шагами. Фигур восемь, а шагов только шесть.

На каждом шаге выполняется одно действие. Шаги выполняются поочередно и решают поставленную задачу. Заголовок и Конец (закругленные фигуры) — это обрамление алгоритма; они не выполняют никаких действий.

Можно сказать по-другому: алгоритм есть графическая инструкция (памятка) для человека, который будет его выполнять.

Итак, мы выяснили, что алгоритм состоит из отдельных шагов. Если нет шагов, то нет и алгоритма. Без шагов алгоритма не бывает.

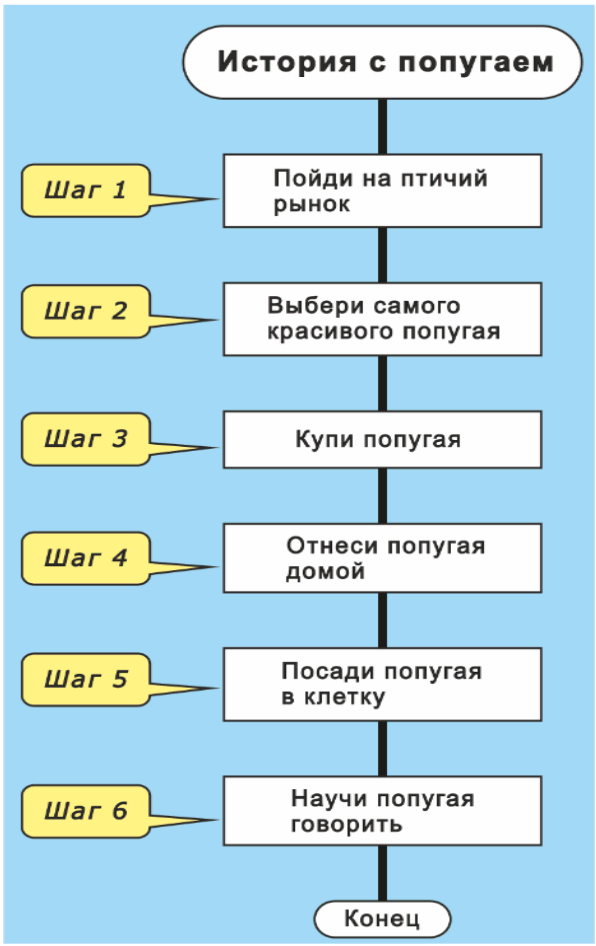


Рис. 2. Показаны шаги алгоритма «История с попугаем»

БЕГУНОК, ИЛИ РАБОЧАЯ ТОЧКА АЛГОРИТМА

Бегунок — это воображаемая точка, которая перемещается (бегает) по алгоритму. Зачем? Чтобы помочь студентам понять суть алгоритма.

Разумеется, она бегает не беспорядочно, а подчиняясь строгой дисциплине. Бегунок всегда держит путь из Начала в Конец, двигаясь по алгоритму от одной иконы к следующей. Точно так же, как поезд следует от одной станции к другой.

На рис. 2 бегунок начинает свой путь в самом верху. Приступая к работе, он (бегунок) объявляет название алгоритма «История с попугаем». Затем делает шаг вниз и читает приказ: «Пойди на птичий рынок». Затем спускается еще на шаг и читает следующий приказ «Выбери самого красивого попугая». Далее он, действуя тем же порядком, перемещается вертикально вниз по всем иконам. И заканчивает свой путь в иконе Конец. Конец — делу венец.

Если вам не нравится слово «бегунок», вместо него можно сказать: *рабочая точка алгоритма.*

ИКОНА «ВОПРОС»

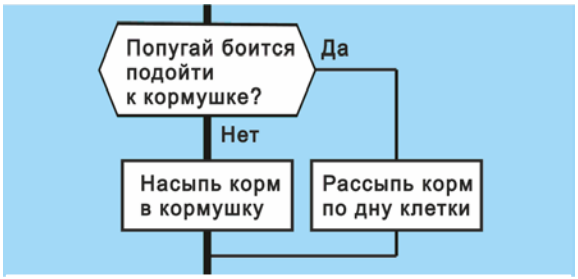


Рис. 3. Икона Вопрос

В верхней части рисунка 3 изображена икона Вопрос. Почему она так называется? Потому, что в ней пишут «да-нетный вопрос». То есть вопрос, на который можно ответить либо «Да», либо «Нет». Все другие ответы запрещены.

В нашей жизни да-нетные вопросы встречаются очень часто. Вот примеры: утюг сломался? Вася купил хлеб? Поезд пришел? Преступника арестовали? Делегация приехала? «Спартак» выиграл? Эта доска короче, чем та? На улице температура выше нуля?

На рис. 3 в иконе Вопрос находится вопросительное предложение: «Попугай боится подойти к кормушке?» При ответе «Да» выполняется действие «Рассыпь корм по дну клетки». При ответе «Нет» — «Насыпь корм в кормушку».

Чтобы понять смысл иконы Вопрос, нужно читать ее сверху вниз. Именно так поступает бегунок. Он входит в икону Вопрос сверху, а выходит вниз или вправо.

Алгоритмы нужны, чтобы решать сложные жизненные задачи. Такие задачи обычно содержат много развилок. Поэтому в алгоритмах нередко приходится использовать икону Вопрос. Ее применяют всякий раз, когда нужно выбрать одно направление из двух.

Иногда в алгоритме надо сделать не одну, а целый ряд развилок. Для этого используют несколько икон Вопрос.

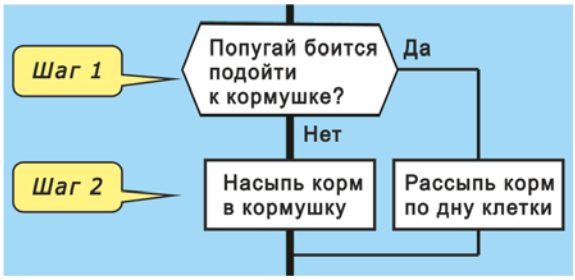


Рис. 4. Показаны шаги при разветвлении алгоритма. На 1-м шаге принимается решение.

ЧТО ТАКОЕ РЕШЕНИЕ

Следует различать два понятия: действие и решение.

Решение — это обоснованный ответ на да-нетный вопрос.

На рис. 2 все шаги описывают действия. На рис. 3 изображена более сложная картина — наряду с действиями появилось *решение*.

Мы вынуждены принимать решение всякий раз, когда на дороге встречается развилка. Куда идти? Направо или налево? «Налево пойдешь — коня потеряешь, направо пойдешь — голову сложишь».

Икона Вопрос выполняет две функции:

- организует развилку,
- обозначает решение.

Решение имеет сложную структуру. Как показано на рис. 4, в его состав входят четыре элемента:

- икона Вопрос;
- текст вопроса, записанный внутри иконы;
- выход из иконы, помеченный словом Да;
- выход, отмеченный словом Нет.

Решение состоит в том, что человек должен выбрать один из двух выходов иконы Вопрос. Проще говоря, он решает: Да или Нет? Для этого надо внимательно посмотреть на попугая.

Если попугай боится есть из кормушки — Да.

Если не боится — Нет (рис. 3).

Что такое шаг алгоритма	• Это действие или решение. • Один шаг — одно действие или одно решение.

ЧТО ТАКОЕ ШАМПУР

На рисунках 1–4 имеется жирная вертикальная линия. Присмотритесь к ней внимательно. Она прямая, как стрела — не имеет ни изгибов, ни изломов. Эта линия называется *шампур.* Почему шампур? Потому что она пронизывает иконы алгоритма точно так же, как настоящий стальной шампур пронизывает кусочки будущего шашлыка.

В языке ДРАКОН понятие *шампур* играет важную роль, помогая выявить структуру алгоритма и представить ее в наглядном виде.

В чем ценность шампура? Прямой, как штык, шампур вводит в алгоритм строгий порядок и устраняет путаницу.

СИЛУЭТ И ТРИ ШАМПУРА

Правило	Выполнить алгоритм – значит пройти путь от начала до конца алгоритма

Применим это правило к рисунку 5. На нем изображена алгоритмическая конструкция *силуэт.* Попробуем понять, как силуэт работает. Проследим движение бегунка от иконы Заголовок до иконы Конец.

Выехав из Заголовка, бегунок мчится отвесно вниз по крайнему левому шампуру. Он движется через станции (рис. 5):

- Покупка попугая.
- Пойди на птичий рынок.
- Выбери самого красивого попугая.
- Купи попугая.
- Отнеси попугая домой.
- Приручение попугая.

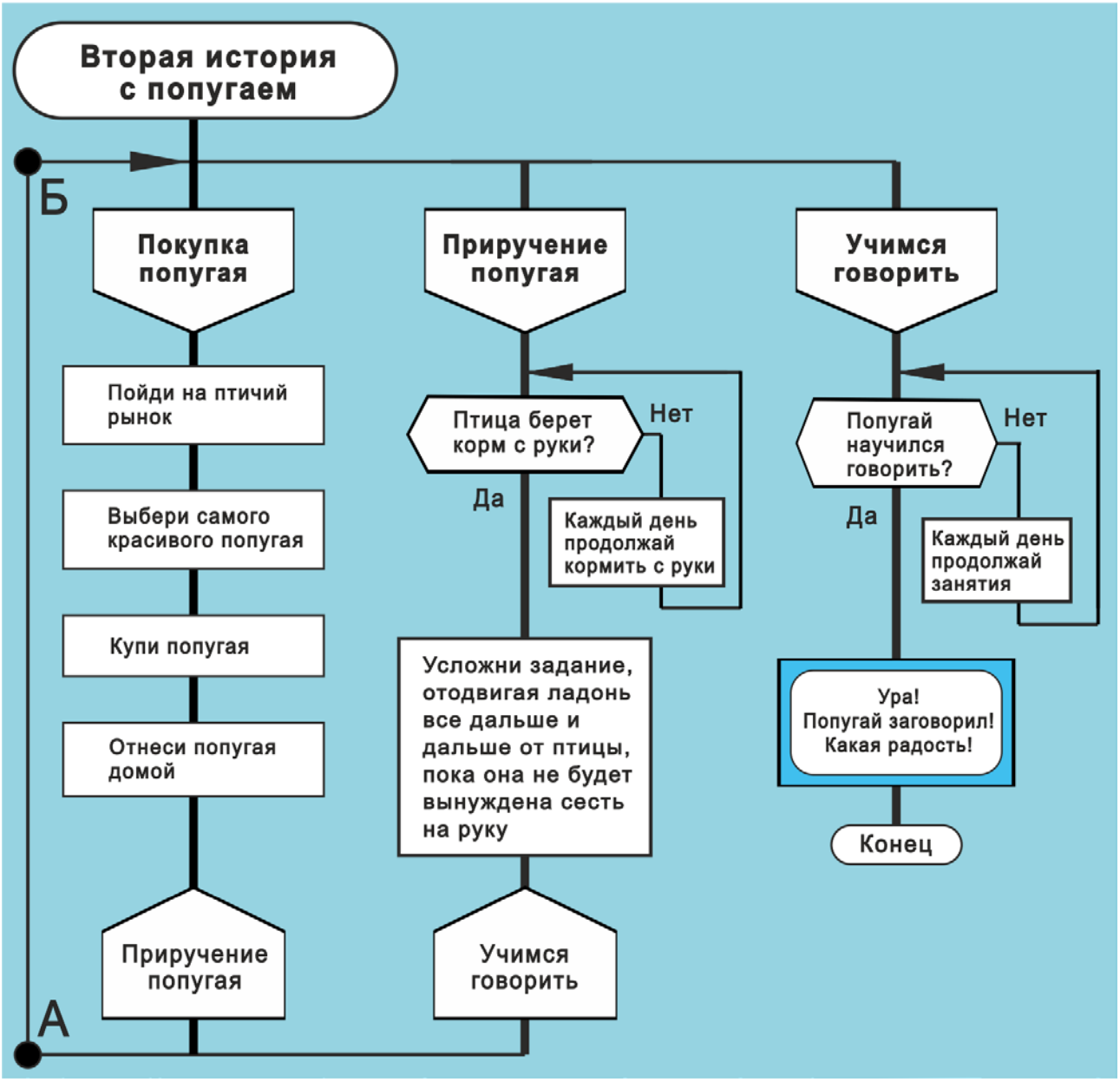


Рис. 5. Алгоритм-силуэт «Вторая история с попугаем»

Икона «Адрес» – последняя станция первой ветки. Куда ехать дальше? Ответ содержится внутри самой иконы. Эта икона потому и зовется «Адрес», что сообщает адрес (название) следующей станции. В данном случае она говорит: следующая станция называется «Приручение попугая».

На рисунке 5 видно, что данная станция находится в начале второго шампура. Но как бегунок доберется туда? По какой линии?

Ответ прост. Выехав из иконы «Адрес», бегунок сворачивает налево и попадает в точку А (рис. 5). Потом движется вверх к точке Б. Затем едет направо по стрелке и въезжает в верхнюю икону «Приручение попугая».

Дальше все ясно. Бегунок скользит вниз по второму шампуру. Добравшись до иконы Адрес, узнает адрес следующей станции («Учимся говорить»). Затем огибает схему по линии АБ, попадает в начало третьего шампура и спускается по нему до конца. На этом выполнение алгоритма заканчивается.

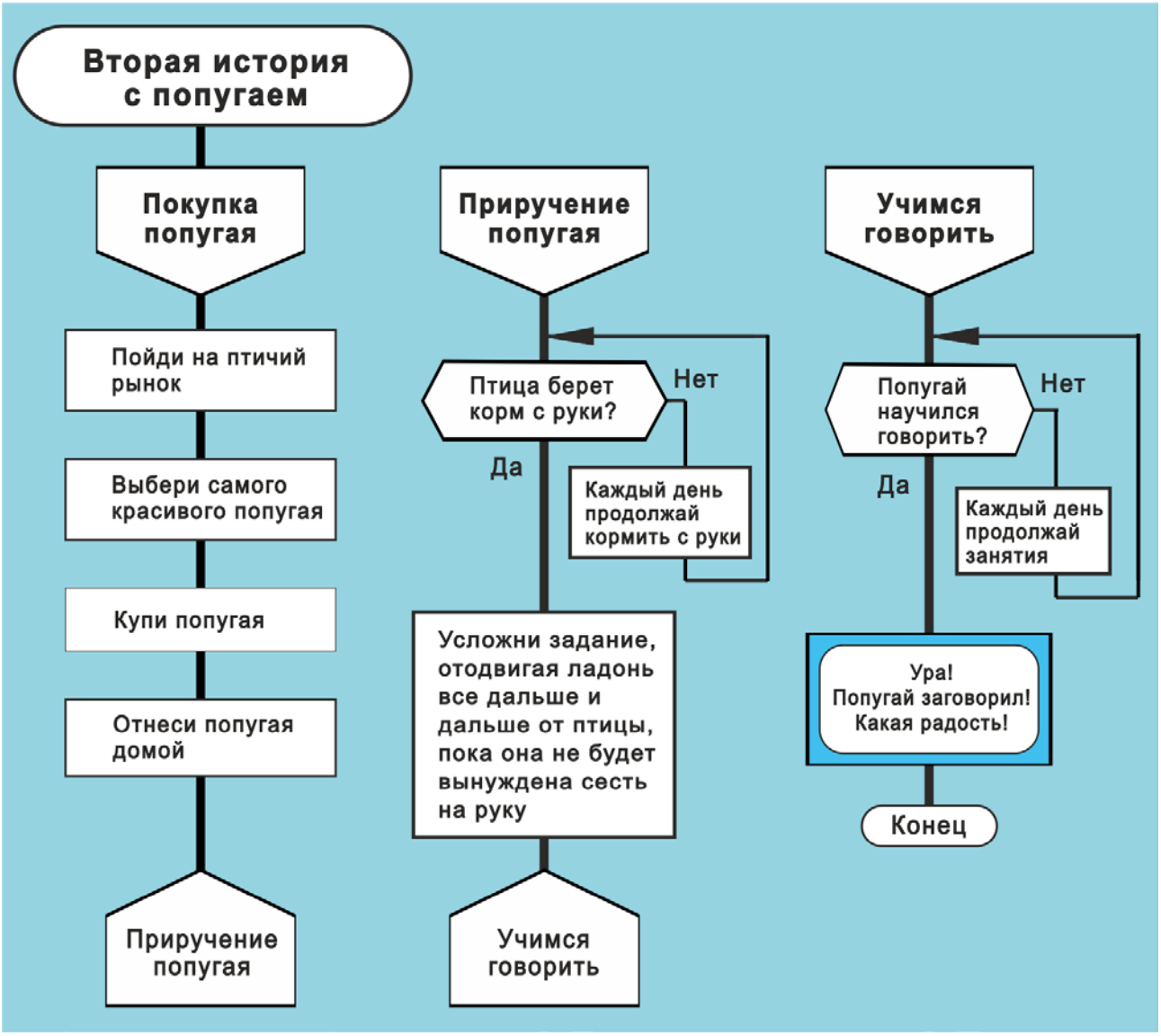


Рис. 6. Алгоритм с удаленным контуром

В ЧЕМ СЕКРЕТ ИКОНЫ «АДРЕС»

Сейчас мы поступим, как чеховский злоумышленник – будем разбирать рельсы. Рельсы — это линии контура, окружающие алгоритм на рис. 5.

Контур содержит:

- нижнюю горизонтальную линию (шину), проходящую через точку А;
- верхнюю горизонтальную линию (шину), проходящую через точку Б;
- линию АБ;
- стрелку справа от точки Б.

Давайте полностью удалим контур — получим рисунок 6. Как будет работать силуэт после таких исправлений?

К нашему удивлению, отсутствие рельсов никак не сказывается на работе алгоритма. В этом легко убедиться. Выехав из иконы Заголовок, бегунок движется вниз по крайней левому шампуру. Опустившись до конца, он попадает в икону Адрес.

Казалось бы, это тупик. Рельсы кончились, дальше ехать некуда. Но это не так. Ведь в иконе Адрес записан адрес следующей станции (Приручение попугая). Зная адрес, бегунок прыгнет туда, куда нужно – в начало второго шампура. И поедет вниз.

Добравшись до конца второго шампура, совершит второй прыжок. И попадет в третий шампур. И так далее

Таким образом, икона «Адрес Z» – это команда «Прыгни в начало «шампура Z». Или, выражаясь по-научному, «Передай управление в начало шампура Z». Данная команда передает приказ: «Брось данный шампур и начни выполнять шампур Z». (Выполнить шампур – значит исполнить нанизанные на него команды).

Переходя от рис. 5 к рис. 6, мы для упрощения стёрли линии контура. Теперь мы убедились, что для работы алгоритма они совершенно не нужны. Маршрут бегунка определяют не они, а указания, записанные в иконе Адрес.

ВАЖНАЯ РОЛЬ ЭРГОНОМИКИ

Тем не менее, указанные линии не следует удалять по эргономическим соображениям. Обрамляющие линии образуют контур алгоритма. Контур — это очень важно. Почему?

Потому что линии контура зрительно склеивают разрозненные куски алгоритма. Они превращают их в приятный для глаза целостный зрительно-смысловой образ. И наоборот, устранение скрепляющих линий приводит к тому, что схема зрительно рассыпается на части, что сбивает с толку читателя. Это наглядно видно на рисунке 6.

ЕЩЕ ОДНА ФУНКЦИЯ КОНТУРА

Маршрут — это путь, идущий от начала до конца алгоритма. В сложном алгоритме может быть много маршрутов. Возникает задача — проследить каждый маршрут и убедиться, что он не содержит ошибок. Для этого полезно провести непрерывную линию из начала в конец. Или, что одно и то же, провести линию пальцем (или указкой), не отрывая палец от бумаги или экрана.

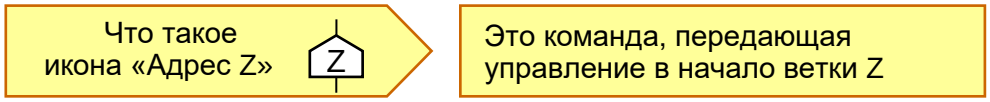
Если есть контур, провести такую линию можно. А если контур исчез, как на рисунке 6 — нельзя.

ШАМПУР И ВЕТКА

Шампур — это просто жирная вертикальная линия (вертикаль). Строго говоря, иконы, которые на нее нанизаны, не входят в понятие шампура. На рис. 5 имеются иконы, находящиеся справа от шампура. Поэтому надо различать два понятия: *шампур* и *ветка*.

Ветка есть зрительно-смысловая часть алгоритма. Она содержит не только шампур, но и связанные с ним иконы, а также соединительные линии. Силуэт на рисунке 5 разбит на три ветки, которые имеют содержательные названия:

- Покупка попугая.
- Приручение попугая.
- Учимся говорить.



КАК СЛЕДУЕТ РАСПОЛАГАТЬ ВЕТКИ НА ЧЕРТЕЖЕ

Ветки упорядочены двояко: логически и пространственно. Логическая последовательность исполнения веток определяется метками, записанными в иконах Адрес.

Однако логический порядок – это еще не все. Очень важно правильно расположить ветки в пространстве. Как это сделать?

Давайте мысленно перетасуем ветки на рис. 6, как колоду карт. И расположим их на чертеже в произвольном порядке. Легко сообразить, что подобное «перепутывание» веток никак не отразится на работе алгоритма. Ведь очередность веток зависит только от команд Адрес. И совсем не зависит от расположения веток на листе бумаги. Словом, сколько ветки ни тасуй, получим тот же самый алгоритм.

Запомните

- Здесь есть тонкость. Перестановка веток не отражается на правильности алгоритма. Однако алгоритм должен быть не только правильным, но и понятным, эргономичным.
- Хаотичное расположение веток затрудняет понимание. А это недопустимо.

Поэтому нужно обязательно упорядочить ветки на чертеже. Удобнее всего расположить их слева направо в той последовательности, в какой они включаются в работу. Для этого служит правило: «Чем правее, тем позже». Оно означает: чем правее находится ветка, тем позже она включается в работу.

Отсюда вытекает эргономическая

Рекомендация

Чтобы схема была наглядной и удобной для восприятия, ветки должны быть упорядочены слева направо согласно принципу: «Более правая ветка работает позже, чем любая ветка, расположенная левее нее» (кроме веточных циклов).

Алгоритм, нарисованный согласно правилу «Чем правее – тем позже», считается хорошим, эргономичным (рис. 5). Схемы, где это правило нарушается, объявляются плохими. Их использование запрещено.

ВЫВОДЫ

1. Алгоритм делится на шаги, которые выполняются друг за другом.
2. Один шаг — одно действие или одно решение.
3. Маршрут — путь, идущий от начала до конца алгоритма.
4. Бегунок — рабочая точка алгоритма.
5. В иконе Вопрос пишут «да-нетный вопрос».
6. Ветка есть зрительно-смысловая часть силуэта.
7. Каждая ветка имеет один шампур.
8. Икона «Адрес Z» — это команда, передающая управление в начало ветки Z.
9. Ветки в силуэте расположены по принципу «Чем правее, тем позже».

Глава 2

ИЗУЧАЕМ АЛГОРИТМ «КАК НАУЧИТЬ ПОПУГАЯ ГОВОРИТЬ»

ВВЕДЕНИЕ

Как выглядит большой и сложный алгоритм на языке ДРАКОН? В качестве модели такого алгоритма можно продемонстрировать новый алгоритм о попугаях. Это по-настоящему сложный алгоритм. Он состоит из четырех частей, занимает почти четыре книжных разворота (семь страниц) и показан на рисунке 14.

Сверхзадача главы — подробно изучить алгоритмическую конструкцию силуэт. Силуэт является мощным и удобным инструментом языка ДРАКОН, который обладает большими возможностями и не имеет аналогов в других языках.

ДЕТАЛИЗАЦИЯ АЛГОРИТМА

Мы бегло познакомились с конструкцией *силуэт* в предыдущей главе. На рисунке 5 показан набросок проекта по обучению попугаев. Это предварительный, явно недостаточный план. В нем всего три ветки и три шампура. Отсутствуют многие важные и необходимые детали.

План на рис. 5 представляет собой скелет, который должен обрасти «мясом». Наша задача — выявить детали и наклеить их на скелет. Процесс добавления подробностей и построение окончательного алгоритма можно назвать детализацией.

Детализация — это творческий процесс разработки окончательного алгоритма, представленного на рис. 14.

Сравним два силуэта на рис. 5 и 14. Первый силуэт краткий и неполный. Второй, наоборот, содержит все подробности и хорошо подогнанные детали.

Первый силуэт назовем *эскизом*, второй — *готовым проектом.* Превращение алгоритма из эскиза в готовый проект можно рассматривать как метод последовательных приближений к окончательному результату.

РАЗДЕЛЯЙ И ВЛАСТВУЙ: ВЕТКИ И ШАМПУРЫ

Алгоритм на рисунке 14 можно сравнить с повестью, которая разбита на главы. Внимательно прочитайте эту повесть от начала до конца.

Найдите на рис. 14 жирные вертикальные линии. Подсчитайте, сколько их. Правильно, девятнадцать. В нашей повести 19 *шампуров.*

Это значит, что алгоритм разделен (по принципу «Разделяй и властвуй») на 19 частей. Части называются *ветками.*

Подобно тому, как повесть делится на главы, дракон-алгоритм делится на ветки. Каждая ветка имеет только один шампур. Стало быть, *число веток всегда равно числу шампуров.*

Далее мы будем поочередно изучать отдельные ветки силуэта и сопоставлять их с полной картиной алгоритма на рис. 14.

ПЕРВАЯ ВЕТКА СИЛУЭТА

Рассмотрим самую первую ветку (рис. 7). В случае большого алгоритма желательно предпослать ему краткое предисловие о его целях и задачах. В нашем случае это будет краткое напутствие для любителей попугаев, которое называется «Пояснение к алгоритму».

Что делает первая ветка? С алгоритмической точки зрения, она ровным счетом ничего не делает. Но это не беда. Ценность этой ветки заключается в другом. В ней находится большая икона «Комментарий», содержащая важные сведения об особенностях поведения домашних попугаев. Это полезный инструктаж для тех, кто собирается приобрести и держать у себя диковинных птиц.

Комментарий разделен на две части. Верхняя часть «Что нужно знать о попугаях» содержит четыре наиболее важных правила и предостережения будущему владельцу птицы. Нижняя часть (под чертой) содержит советы общего характера.

Повторим: данная ветка алгоритмически пустая. Она не выполняет никаких действий. Нужна ли в алгоритме подобная пустышка? Для простых алгоритмов, конечно, не нужна.

А для сложных ситуация иная. Хороший комментарий пригодится тем, кто будет исправлять алгоритм после вас, на этапе сопровождения. Забота о будущих читателях сложного алгоритма может быть не только полезной, но и необходимой.

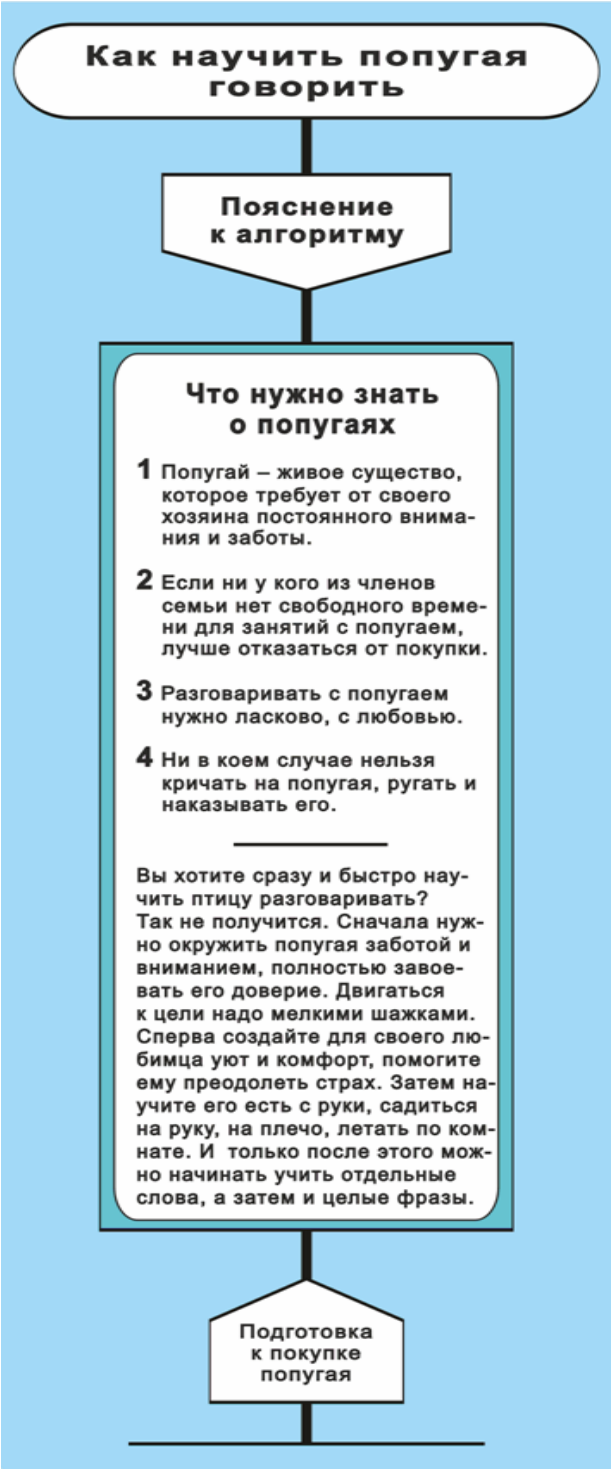


Рис. 7. Начальная ветка алгоритма может содержать икону Комментарий, в которой записано развернутое пояснение

Как работает ветка? После того, как вы прочитали комментарий, надо спуститься в икону Адрес и через нее перейти в ветку «Подготовка к покупке попугая».

Икона Адрес, где бы она ни находилась, всегда выполняют одну и ту же функцию — функцию перехода (goto) к нужной ветке.

ВЕТКА «ПОДГОТОВКА К ПОКУПКЕ ПОПУГАЯ»

Алгоритм есть вереница действий, или последовательность команд. Именно такая последовательность содержится в ветке на рис. 8. Читаем:

- Купи клетку для попугая.
- Вставь в клетку кормушку.
- Выбери комнату, где будет жить попугай.
- Выбери место для клетки.
- Повесь клетку в нужное место.

Дальше следует да-нетный вопрос:

- Нравится?

Если «Нет», не нравится, следует команда:

- Перевесь клетку в другое место.

После того, как хозяин разобрался с клеткой и водворил ее на место, звучит последний приказ:

- Создай в комнате нужную температуру и влажность.

На этом основная работа ветки заканчивается. Бегунок попадает в икону Адрес и через нее перемещается в следующую, третью по счету ветку.

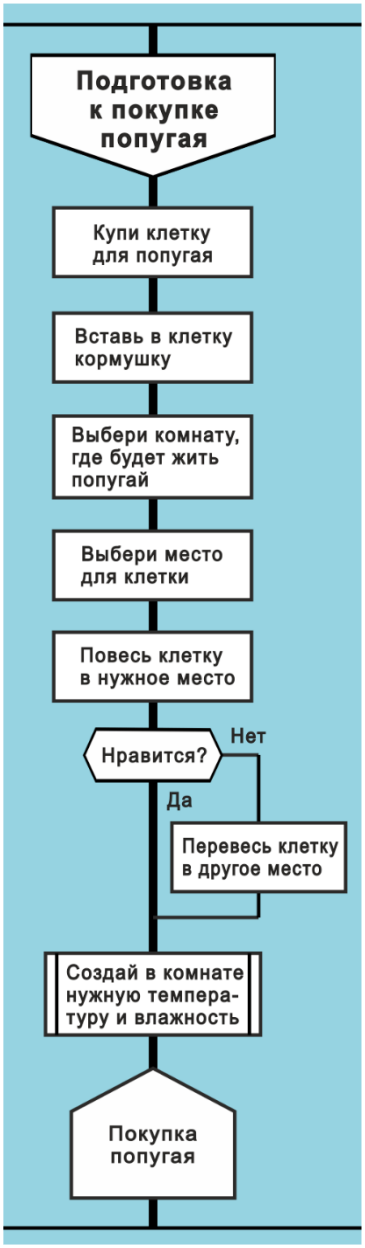


Рис. 8. Ветка «Подготовка к покупке попугая»

ИКОНА «ВСТАВКА»

В нижней части рисунка 8 появилась необычная икона с двойными линиями по бокам. Она называется *Вставка* и обозначает отсутствующий алгоритм. Надпись «Создай в комнате нужную температуру и влажность» говорит о том, что речь идет о другом алгоритме, который здесь не поместился и не показан.

А где же он показан? Он обязательно должен быть раскрыт где-нибудь в подходящем для этого месте. Причем читатель должен иметь возможность попасть в это место и ознакомиться с алгоритмом.

Таким образом, икона Вставка — это всего лишь памятный знак. Знак напоминает, что алгоритм «Создай в комнате нужную температуру…» подробно разрисован не здесь, а на другом чертеже. Слово «вставка» означает, что икону Вставка можно мысленно удалить, а вместо нее мысленно *вставить* нужный алгоритм.

ЧТО ТАКОЕ ПЕРЕКЛЮЧАТЕЛЬ

Мы должны приступить к изучению третьей ветки, которая называется «Покупка попугая». Трудность в том, что появилась незнакомая нам алгоритмическая конструкция — *переключатель* (switch).

Познакомимся с ней поближе. Для этого отвлечемся на время от нашей темы и обратимся к рисункам 9 и 10.

Схема на рис. 9 позволяет сделать в алгоритме развилку на три направления. Для этого используются две иконы Вопрос. Однако задачу можно решить и другим способом — с помощью переключателя (рис. 10).

Переключатель — разветвление алгоритма на несколько тропинок, которые потом сливаются в одну. Переключатель сложная структура. Он строится из простых кирпичиков. Такими кирпичиками служат иконы Выбор и Вариант. Зачем они нужны?

Икона Выбор содержит вопрос или приказ, имеющий несколько ответов. Каждый ответ пишут в отдельной рамочке — иконе Вариант.

Взглянем на рис. 10. На первый взгляд там нет вопроса. Однако на самом деле вопрос есть, правда неявный. Чтобы убедиться, слово «Светофор» прочитаем так: «Какой сигнал светофора сейчас горит?». Получим три ответа:

- зеленый,
- желтый,
- красный.

Попробуем описать, как работает переключатель. Описание обычно начинается со слова «если». Вот примеры.

- Если светофор зеленый — жми на газ.
- Если светофор желтый — притормози.
- Если светофор красный — стой (рис. 10).

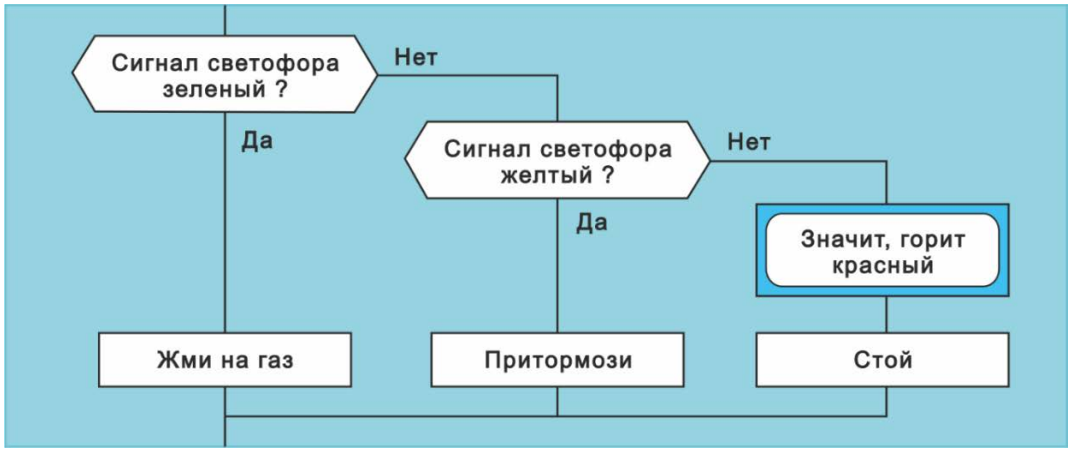


Рис. 9. Развилка на три направления, построенная с помощью иконы Вопрос

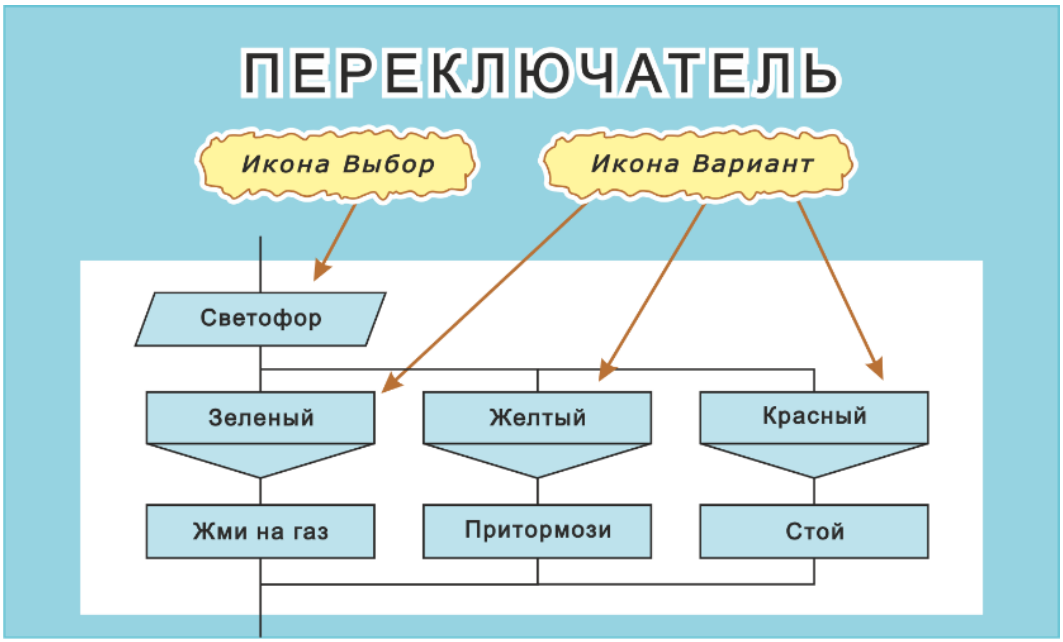
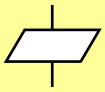


Рис. 10. Развилка на три направления, построенная с помощью переключателя

Что такое переключатель

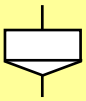
- Это часть алгоритма, имеющая один вход и один выход, внутри которой алгоритм разветвляется на несколько дорожек.
- Переключатель содержит икону Выбор и несколько икон Вариант.

Что такое икона Выбор



- Икона, которую рисуют в начале переключателя
- В ней пишут вопрос или приказ, имеющий два и более ответов

Что такое икона Вариант



Икона переключателя, в которой пишут один ответ на вопрос

ВЕТКА «ПОКУПКА ПОПУГАЯ»

Итак, мы выяснили, что переключатель — полезная и удобная вещь. Вернемся к нашим попугаям на рисунке 11 и прочитаем первые две команды:

- Посети питомник или зоомагазин.
- Посоветуйся с продавцом.

Далее следует переключатель, где в иконе Выбор записано: «Выбери возраст». Эту фразу можно преобразовать в вопрос. Какую птицу вы желаете приобрести: молодую или взрослую?

Ответ дан в иконах Вариант, где предлагается выбрать одно из двух по принципу либо – либо:

- либо птенца,
- либо взрослую птицу.

Обратите внимание, бегунок может пройти через переключатель только по одному маршруту. Либо по левому (через икону «Птенец»), либо по правому (через икону «Взрослая птица»), но не по обоим сразу.

После того как решение о возрасте принято, бегунок спускается вниз и попадает в следующий переключатель. В этом месте предлагаются на выбор три вида попугаев:

- жако,
- какаду,
- волнистый попугай.

Здесь есть определенная условность, потому что наш список попугаев далеко не полный. На самом деле попугаев очень много:

- ара,
- амазон,
- розелла,
- неразлучник,
- корелла,

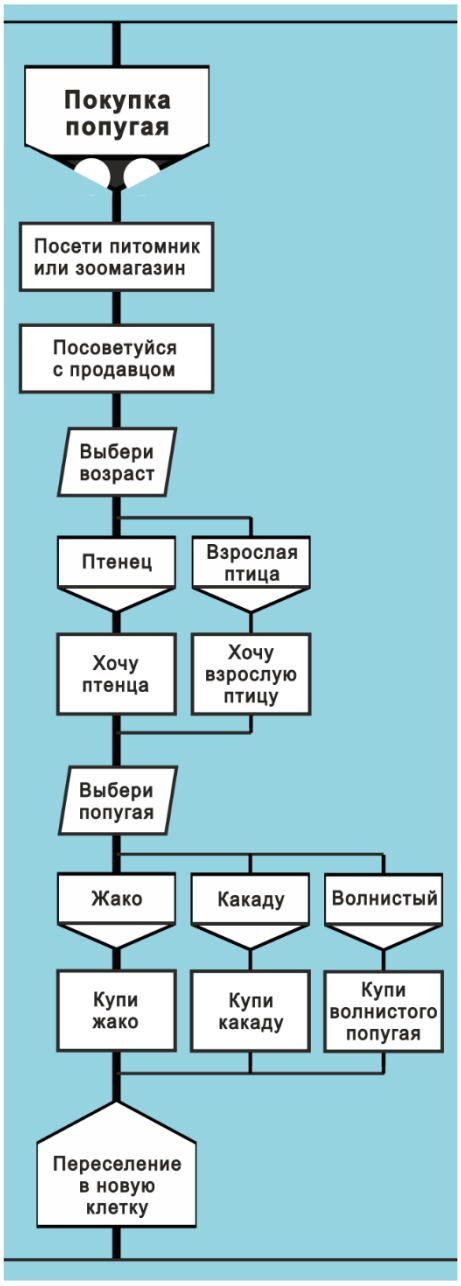


Рис. 11. Ветка «Покупка попугая»

- александрийский попугай,
- лори и др.

Для экономии места мы о них умалчиваем. Таким образом, в нашем переключателе представлены всего три вида (жако, какаду, волнистый), выбранные из длинного списка.

И последнее. В самом низу ветки на рис. 11 имеется икона Адрес «Переселение в новую клетку», которая управляет переходом бегунка в следующую ветку.

КАКИЕ ИКОНЫ МЫ УЖЕ ЗНАЕМ. ЗОЛОТАЯ ДЮЖИНА

ДРАКОН — графический язык. Он имеет графические буквы. Необходимо запомнить их. Нужно знать, как они выглядят и как называются. Это нетрудно.

К этому моменту мы познакомились с «золотой дюжиной» графических фигур языка ДРАКОН.

Проверьте себя. Закройте названия на рис. 12 рукой или газетой и, глядя на фигуры, постарайтесь вспомнить, как они называются. После этого просмотрите предыдущие рисунки (рис. 1 – 11) и найдите на них все 12 фигур.

Учтите, что в состав золотой дюжины входят десять икон (1 – 10) и две составные иконы, или макроиконы (11 и 12).

Самые ходовые среди них — иконы Действие и Вопрос.

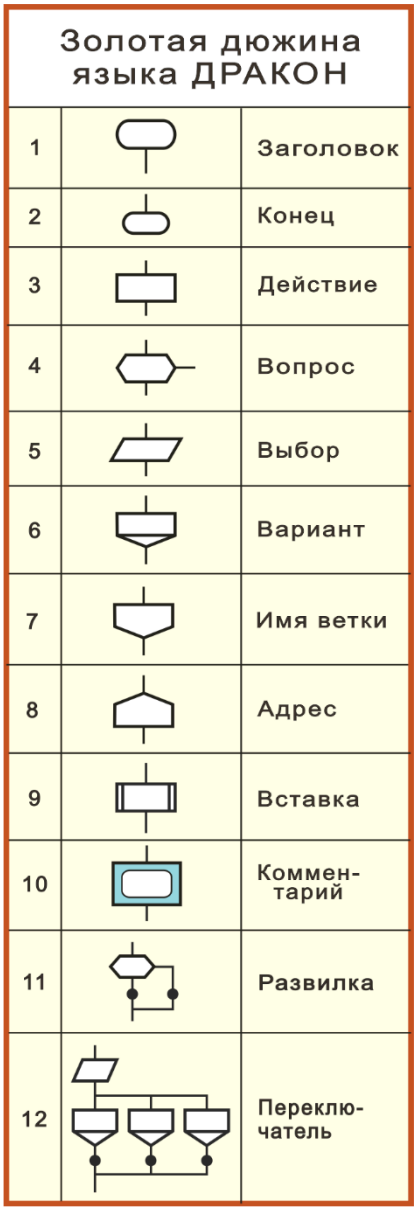


Рис. 12. Часто используемые фигуры в алгоритмах

ПОДРОБНЫЙ ПЛАН ГРАФИЧЕСКОГО РАССКАЗА О ПОПУГАЯХ

Наш алгоритм состоит из 19 веток (рис. 14). Перечислим названия веток:

- Пояснение к алгоритму.
- Подготовка к покупке попугая.
- Покупка попугая.
- Переселение в новую клетку.
- Приручение попугая.
- Учимся брать корм с ладони.
- Усложняем задачу.
- Учимся сидеть на руке.
- Учимся сидеть на плече.
- Как уберечь попугая.
- Новые заботы.
- Летаем по комнате.
- Учим свое имя.

- Учим другие слова.
- Учим первую фразу.
- Учим другие фразы.
- Попугай разговорился.
- Поиск доброго человека.
- Завершение.

Изучив поочередно все ветки и уяснив связи между ними, вы научитесь использовать алгоритмическую конструкцию силуэт для составления больших и сложных алгоритмов.

КАК ЧИТАТЬ АЛГОРИТМ НА РИСУНКЕ 14

Вспомним, что силуэт делится на 19 веток, точно так же, как повесть делится на 19 глав. Каждую ветку читаем сверху вниз. Прочитав первую ветку (главу), переходим ко второй, потом к третьей и так далее.

Повесть на рис. 14 построена так, что главы (то есть ветки) расположены последовательно, друг за другом, слева направо, что облегчает чтение.

Мы подробно рассмотрели три первые ветки, представленные на рисунках 7, 8 и 11. Благодаря этому мы познакомились с графическим языком, который нужно усвоить, чтобы понимать прочитанное.

Приглашаем читателей пуститься в свободное плавание по рисунку 14 и самостоятельно изучить оставшиеся шестнадцать веток. Ниже мы добавим несколько пояснений.

ВЕТКА «УЧИМСЯ БРАТЬ КОРМ С ЛАДОНИ»

Четвертая ветка («Переселение в новую клетку») и пятая («Приручение попугая») не представляют трудности. Надо просто внимательно читать текст, записанный внутри икон. И все будет в порядке.

Зато шестая ветка («Учимся брать корм с ладони») вызывает неподдельный интерес. Приглядимся к ней (рис. 13).

Вот первые две команды:

- Разговаривай с попугаем нежно и ласково.
- Дождись утра.

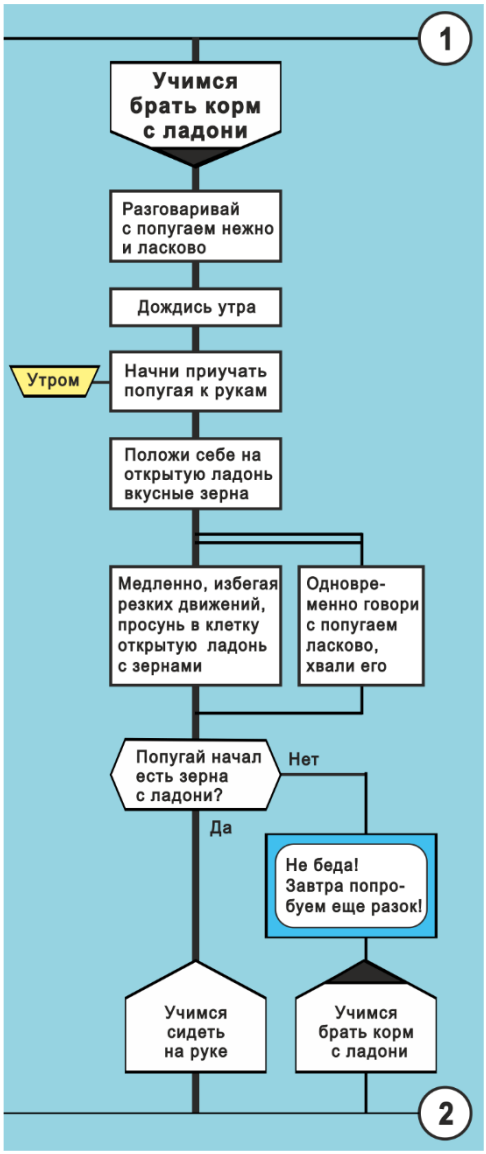


Рис. 13. Ветка «Учимся брать корм с ладони»

Для попугая важна размеренная жизнь и строгий режим дня. Заниматься с птицей нужно ежедневно, в одно и то же время — либо утром, либо вечером.

В данном случае предполагается, что хозяева предпочитают обучать птицу по утрам. Этим объясняется команда «Дождись утра».

Читаем дальше:

- Начни приучать попугая к рукам.
- Положи себе на открытую ладонь вкусные зерна.

Слева от «Начни приучать…» нарисована трапеция, а внутри сказано: «Утром». Желтая трапеция называется Синхронизатор и предназначена для указания точного отсчета времени по таймеру, например, «8 часов 30 минут». Однако в нашем случае подобная точность не нужна; поэтому написано просто: «Утром».

Бегунок скользит вниз, и мы видим два параллельных процесса, имеющих точно обозначенные границы (рис. 13).

Начало процесса показано двумя горизонтальными линиями, *конец процесса* — одной.

Вот первый процесс:

- «Медленно, избегая резких движений, просунь…»

Вот второй процесс:

- «Одновременно говори с попугаем…»

Затем появляется вопрос:

- Попугай начал есть зерна с ладони?

Если ответ «Да», бегунок идет вниз через икону Адрес и переходит в начало следующей ветки, которая называется «Учимся сидеть на руке».

Если же получен ответ «Нет», бегунок уходит вправо и через правую икону Адрес возвращается наверх к началу своей собственной ветки.

Что все это значит? Мы столкнулись с новой конструкцией, которую необходимо пояснить.

ЧТО ТАКОЕ ВЕТОЧНЫЙ ЦИКЛ

До сих пор все ветки имели одну икону Адрес, то есть один выход. Однако на рис. 13 впервые появилась ветка с двумя выходами, имеющая две иконы Адрес. Правый выход Адрес позволяет бегунку пройти по данной ветке не один раз, а больше, то есть выполнять ее повторно. Повторное выполнение ветки называется *веточным циклом.*

Мы остановились на вопросе: «Попугай начал есть зерна с ладони?» Если попугай упрямится и отворачивается, из иконы Вопрос выходим направо через «Нет». Далее после иконы Комментарий читаем икону Адрес: «Учимся брать корм с ладони». Ищем наверху такое же название. Так и есть! — это название нашей ветки. Поэтому бегунок прыгает наверх в начало ветки. И начинает выполнять *веточный цикл.*

ЧЕРНЫЕ ТРЕУГОЛЬНИКИ

Веточный цикл отмечен двумя черными треугольниками (рис. 13).

Какова роль черных треугольников? Для работы алгоритма они совершенно не нужны. Они ни на что не влияют. Если их убрать, ровным счетом ничего не изменится — алгоритм будет работать по-прежнему.

Зачем же появились черные треугольники? Только для того, чтобы служить удобным зрительным ориентиром. Если бы треугольников не было, пришлось бы долго разбираться, чтобы понять, где «спрятался» цикл. А треугольники сразу бросаются в глаза. (Веточный цикл возникает, если в одной из икон Адрес записан адрес собственной ветки).

Что такое веточный цикл

- Это цикл, при котором в иконе Адрес написано имя своей собственной (или более левой) ветки.
- Чтобы легко найти веточный цикл, его помечают двумя черными треугольниками.

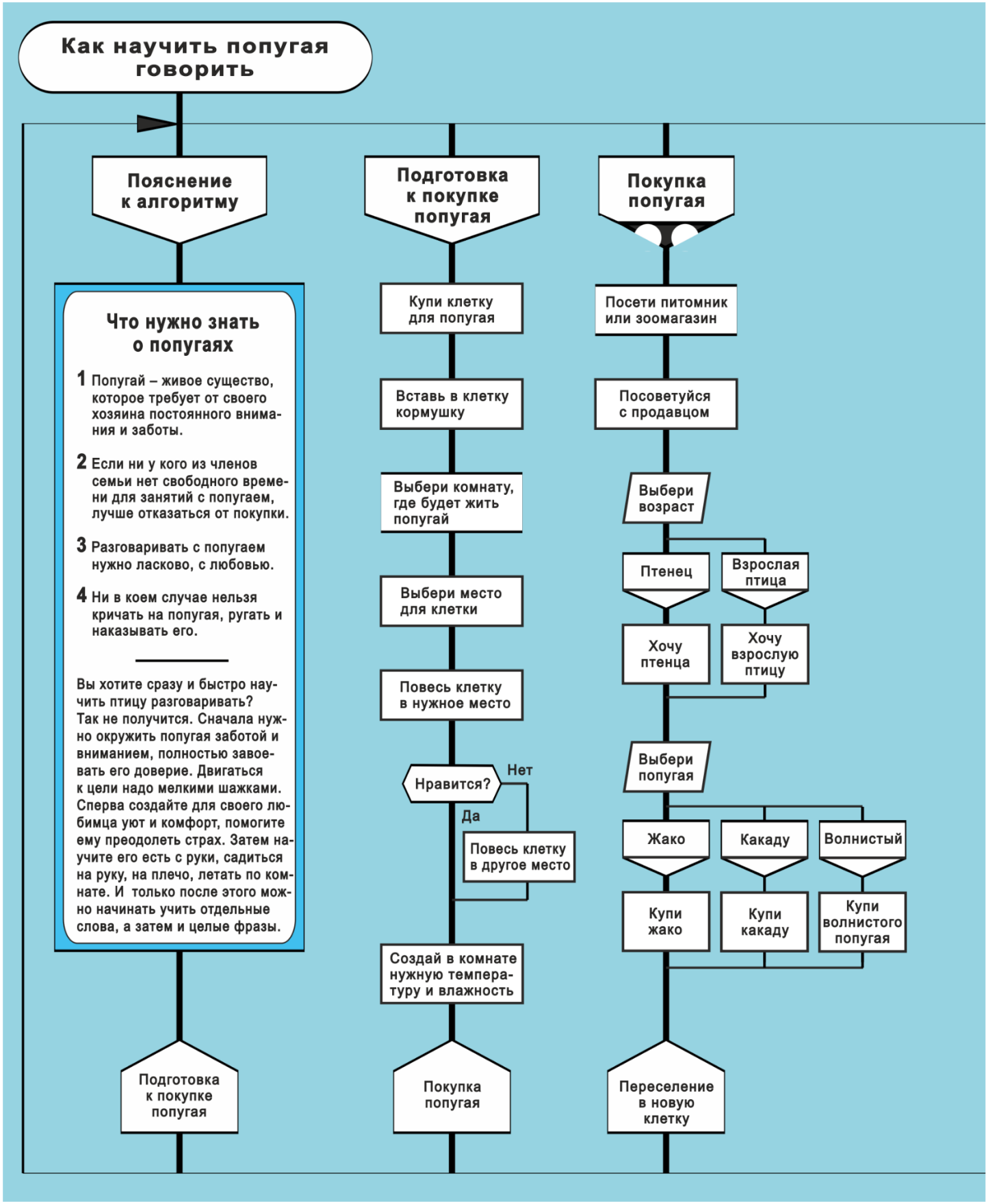
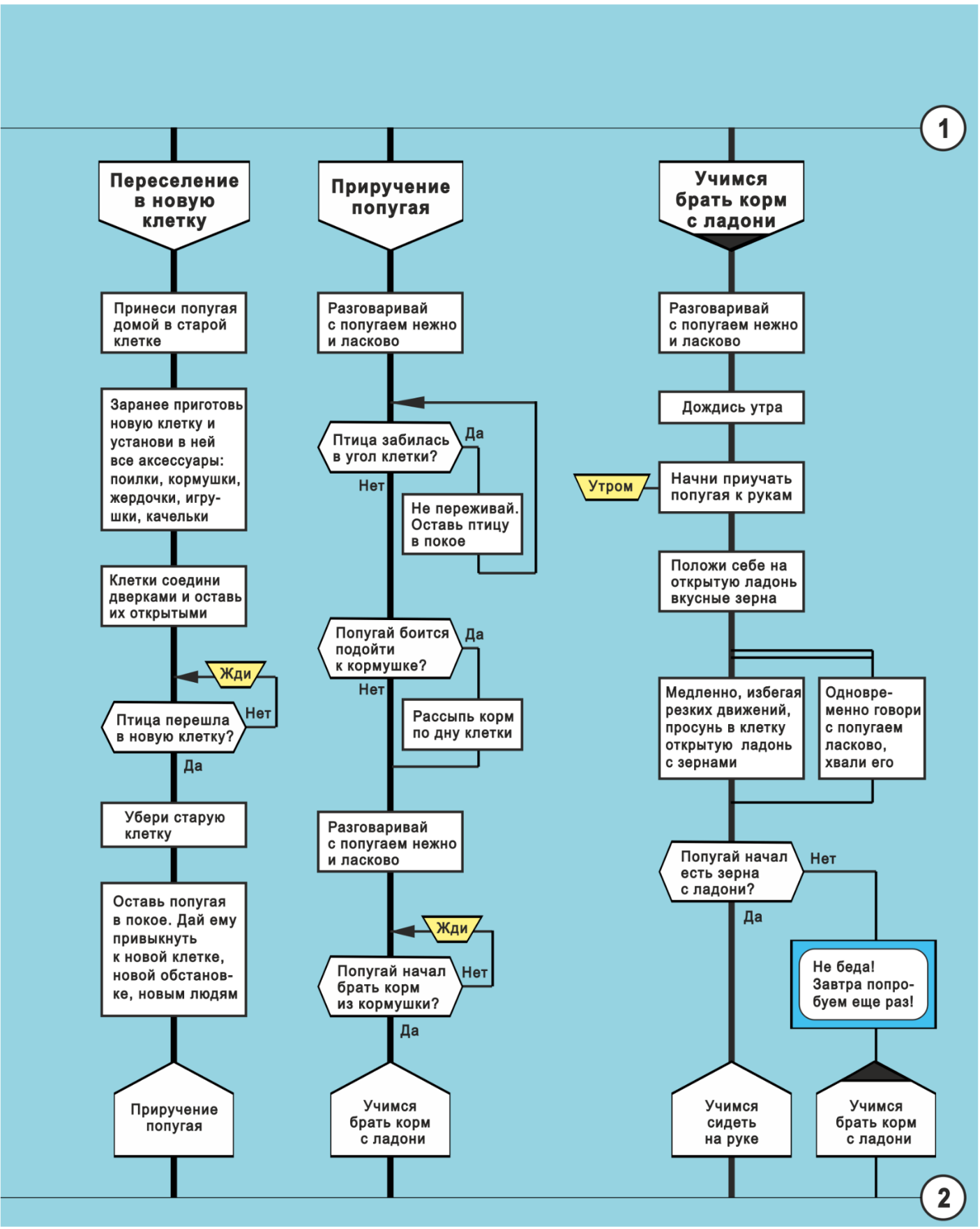


Рис. 14. Алгоритм «Как научить попугая говорить». *Часть 1.*


1
Переселение в новую клетку
Принеси попугая домой в старой клетке
Заранее приготовь новую клетку и установи в ней все аксессуары: поилки, кормушки, жердочки, игрушки, качельки
Клетки соедини дверками и оставь их открытыми
Жди
Птица перешла в новую клетку?
Нет
Да
Убери старую клетку
Оставь попугая в покое. Дай ему привыкнуть к новой клетке, новой обстановке, новым людям
Приручение попугая
Приручение попугая
Разговаривай с попугаем нежно и ласково
Птица забилась в угол клетки?
Да
Нет
Не переживай. Оставь птицу в покое
Попугай боится подойти к кормушке?
Да
Нет
Рассыпь корм по дну клетки
Разговаривай с попугаем нежно и ласково
Жди
Попугай начал брать корм из кормушки?
Нет
Да
Учимся брать корм с ладони
Учимся брать корм с ладони
Разговаривай с попугаем нежно и ласково
Дождись утра
Утром
Начни приучать попугая к рукам
Положи себе на открытую ладонь вкусные зерна
Медленно, избегая резких движений, просунь в клетку открытую ладонь с зернами
Одновременно говори с попугаем ласково, хвали его
Попугай начал есть зерна с ладони?
Нет
Да
Не беда! Завтра попробуем еще раз!
Учимся сидеть на руке
Учимся брать корм с ладони
2

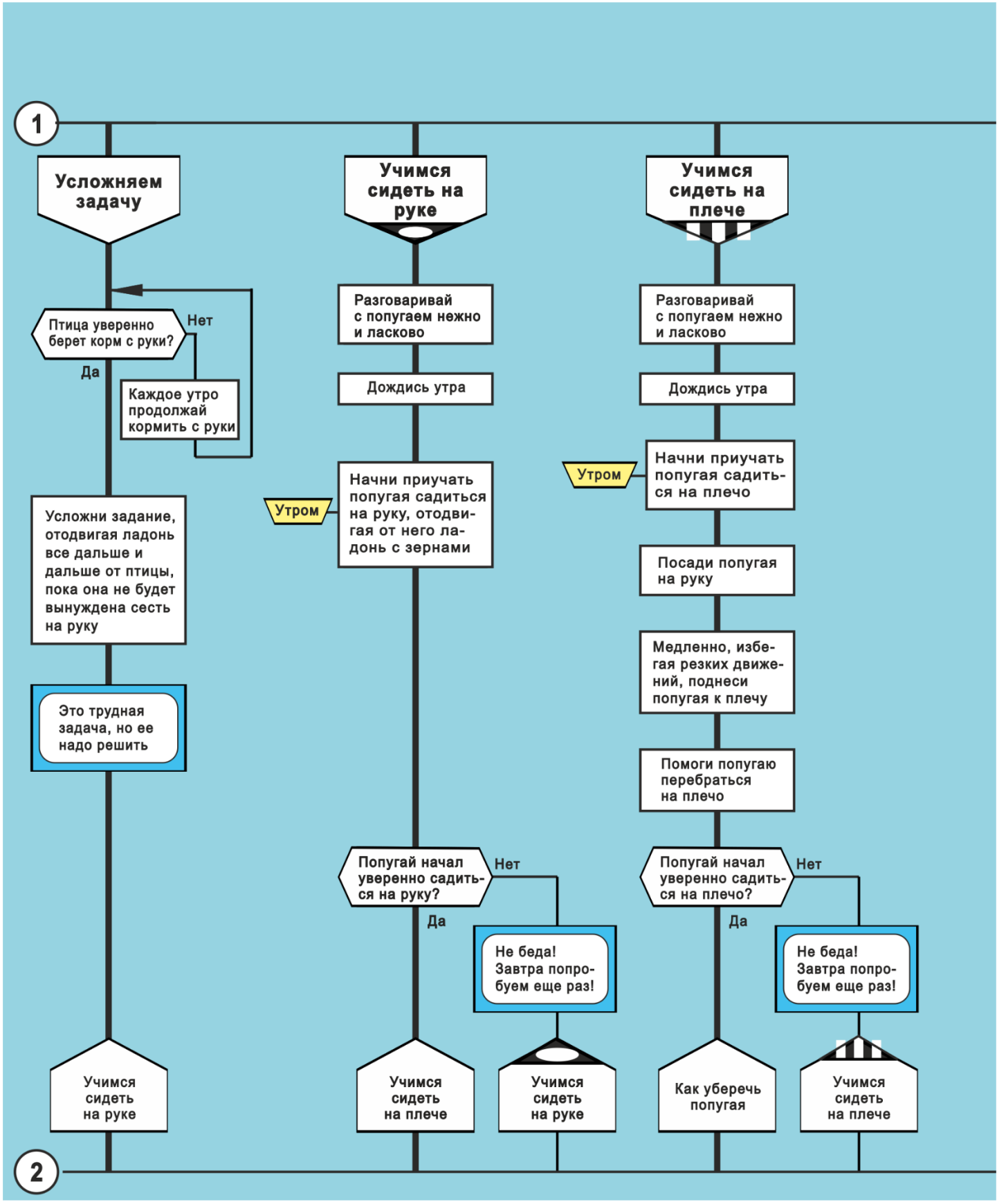
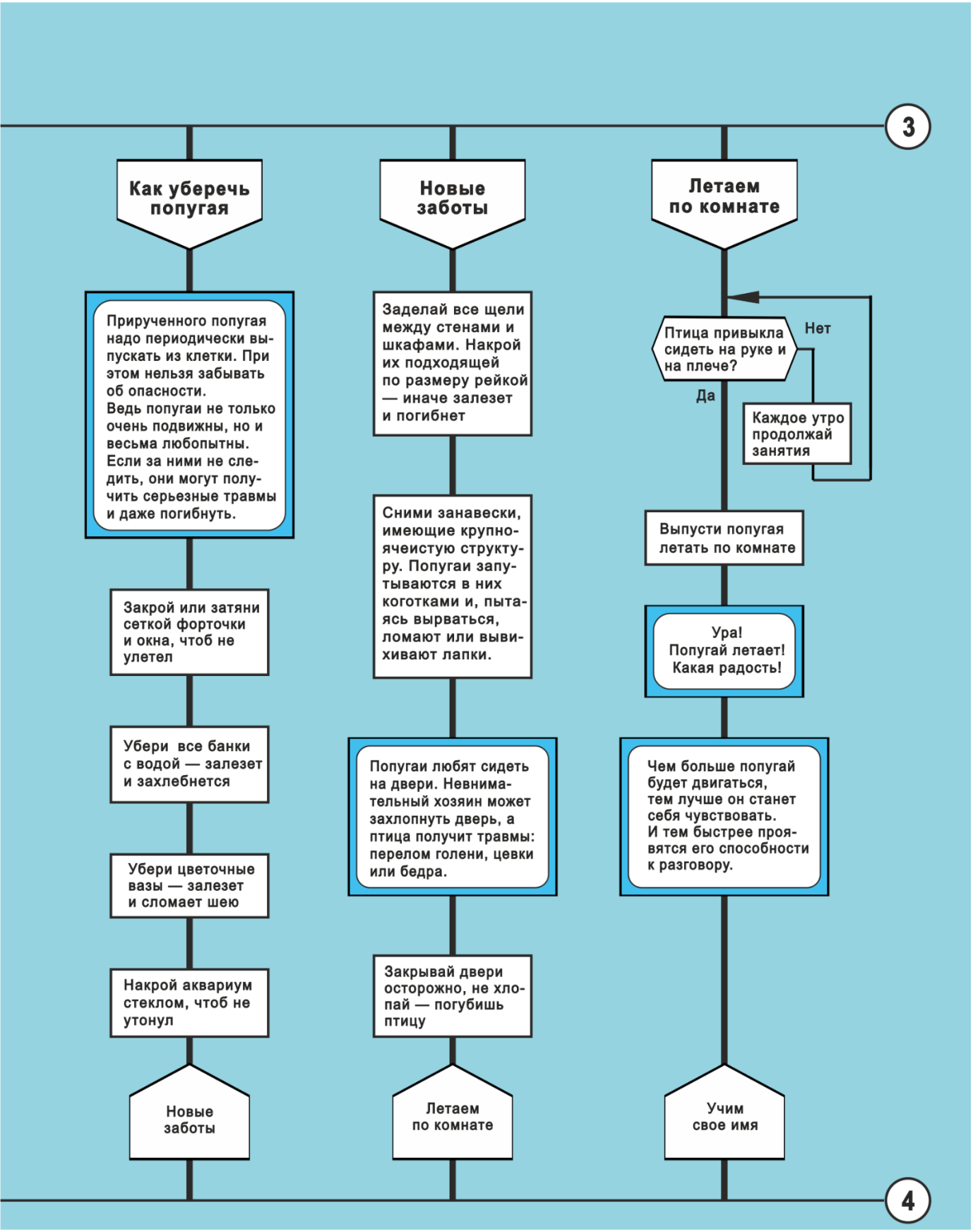


Рис. 14. Алгоритм «Как научить попугая говорить». *Часть 2.*


3
Как уберечь попугая
Приученного попугая надо периодически выпускать из клетки. При этом нельзя забывать об опасности.
Ведь попугаи не только очень подвижны, но и весьма любопытны. Если за ними не следить, они могут получить серьезные травмы и даже погибнуть.
Закрой или затяни сеткой форточки и окна, чтоб не улетел
Убери все банки с водой — залезет и захлебнется
Убери цветочные вазы — залезет и сломает шею
Накрой аквариум стеклом, чтоб не утонул
Новые заботы
Новые заботы
Заделай все щели между стенами и шкафами. Накрой их подходящей по размеру рейкой — иначе залезет и погибнет
Сними занавески, имеющие крупноячеистую структуру. Попугаи запутываются в них коготками и, пытаясь вырваться, ломают или вывихивают лапки.
Попугаи любят сидеть на двери. Невнимательный хозяин может захлопнуть дверь, а птица получит травмы: перелом голени, цевки или бедра.
Закрывай двери осторожно, не хлопай — погубишь птицу
Летаем по комнате
Летаем по комнате
Птица привыкла сидеть на руке и на плече?
Нет
Да
Каждое утро продолжай занятия
Выпусти попугая летать по комнате
Ура! Попугай летает! Какая радость!
Чем больше попугай будет двигаться, тем лучше он станет себя чувствовать. И тем быстрее проявятся его способности к разговору.
Учим свое имя
4

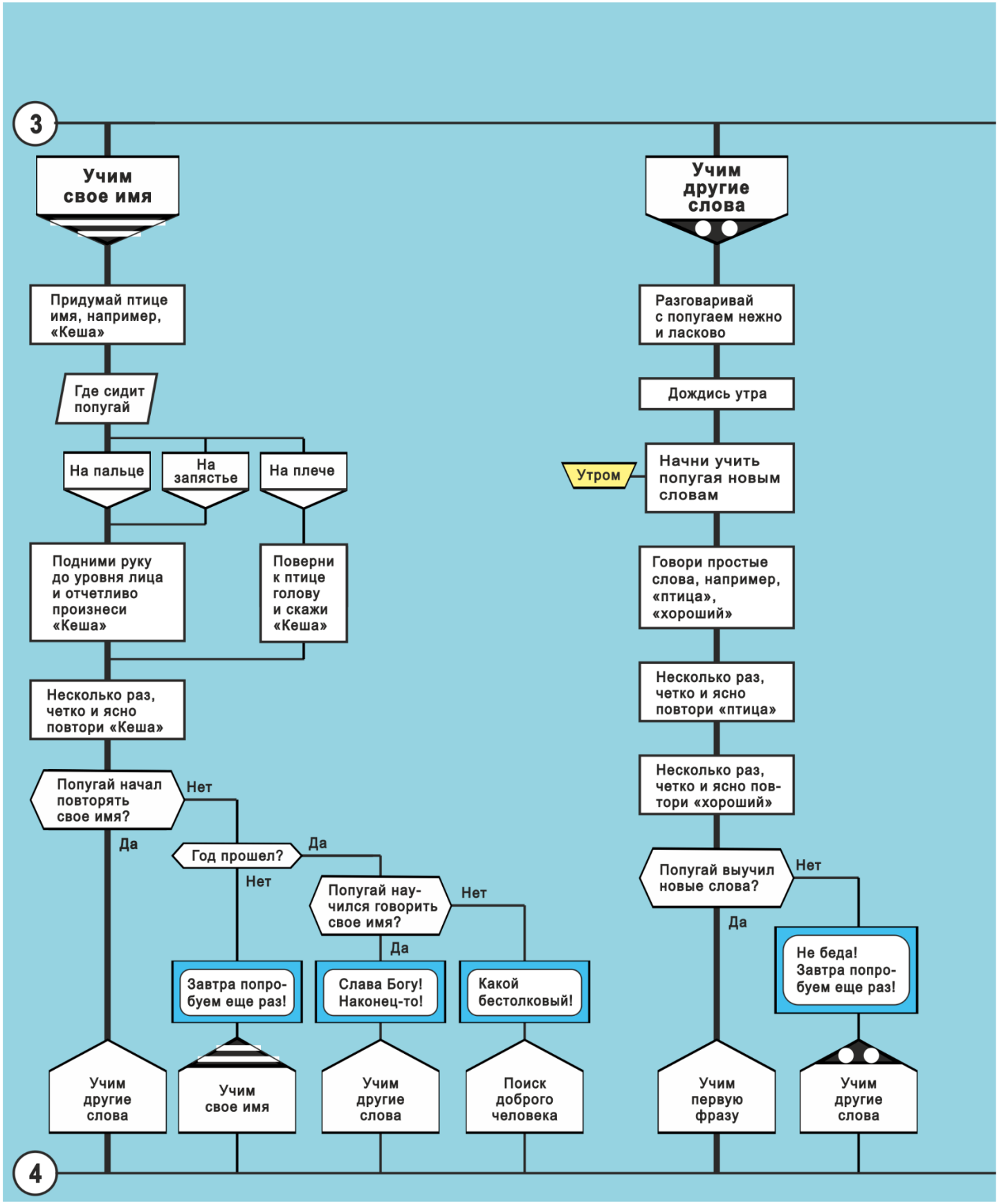
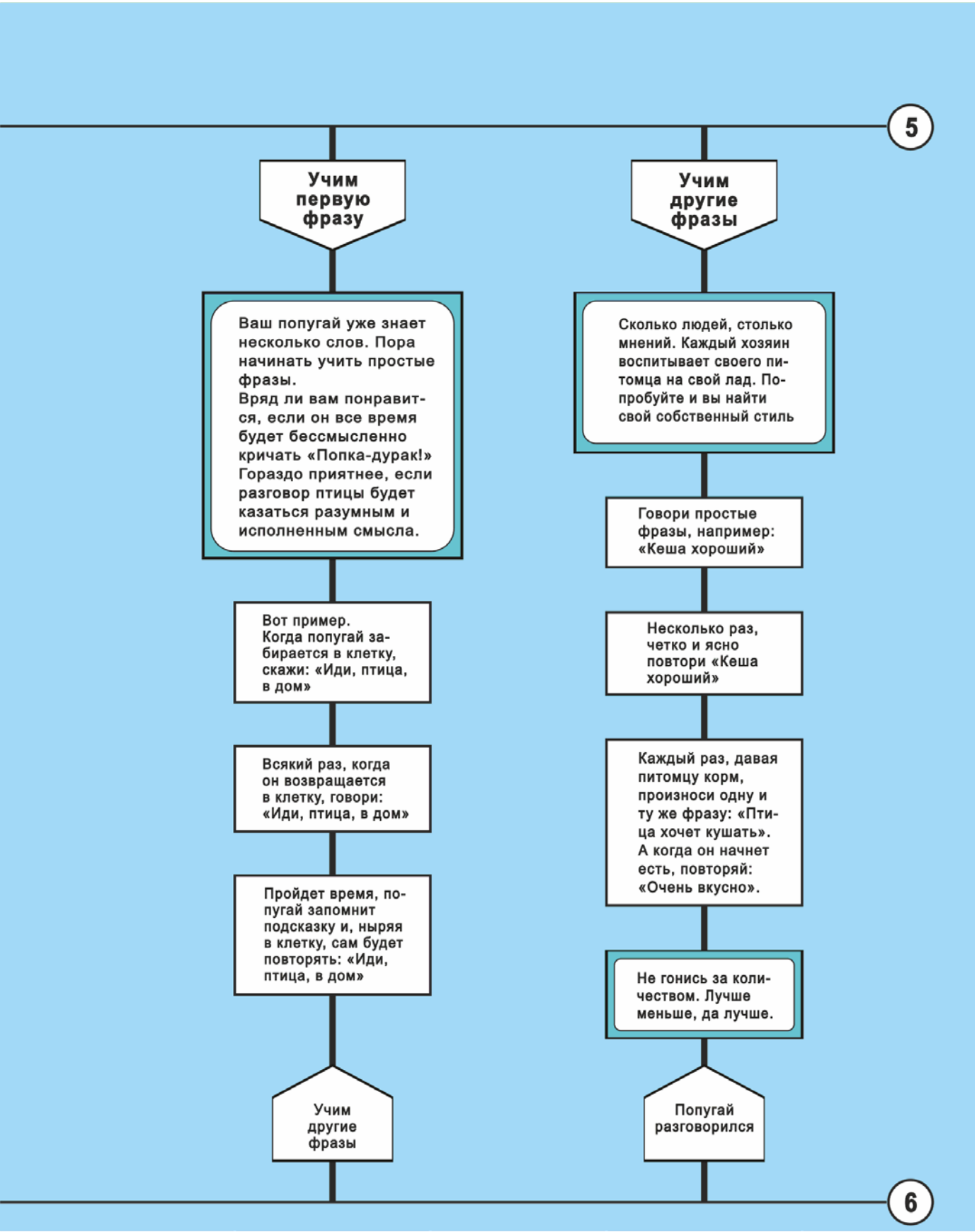


Рис. 14. Алгоритм «Как научить попугая говорить». *Часть 3.*


5
Учим первую фразу
Ваш попугай уже знает несколько слов. Пора начинать учить простые фразы.
Вряд ли вам понравится, если он все время будет бессмысленно кричать «Попка-дурак!» Гораздо приятнее, если разговор птицы будет казаться разумным и исполненным смысла.
Вот пример. Когда попугай забирается в клетку, скажи: «Иди, птица, в дом»
Всякий раз, когда он возвращается в клетку, говори: «Иди, птица, в дом»
Пройдет время, попугай запомнит подсказку и, ныряя в клетку, сам будет повторять: «Иди, птица, в дом»
Учим другие фразы
Учим другие фразы
Сколько людей, столько мнений. Каждый хозяин воспитывает своего питомца на свой лад. Попробуйте и вы найти свой собственный стиль
Говори простые фразы, например: «Кеша хороший»
Несколько раз, четко и ясно повтори «Кеша хороший»
Каждый раз, давая питомцу корм, произноси одну и ту же фразу: «Птица хочет кушать». А когда он начнет есть, повторяй: «Очень вкусно».
Не гонись за количеством. Лучше меньше, да лучше.
Попугай разговорился
6

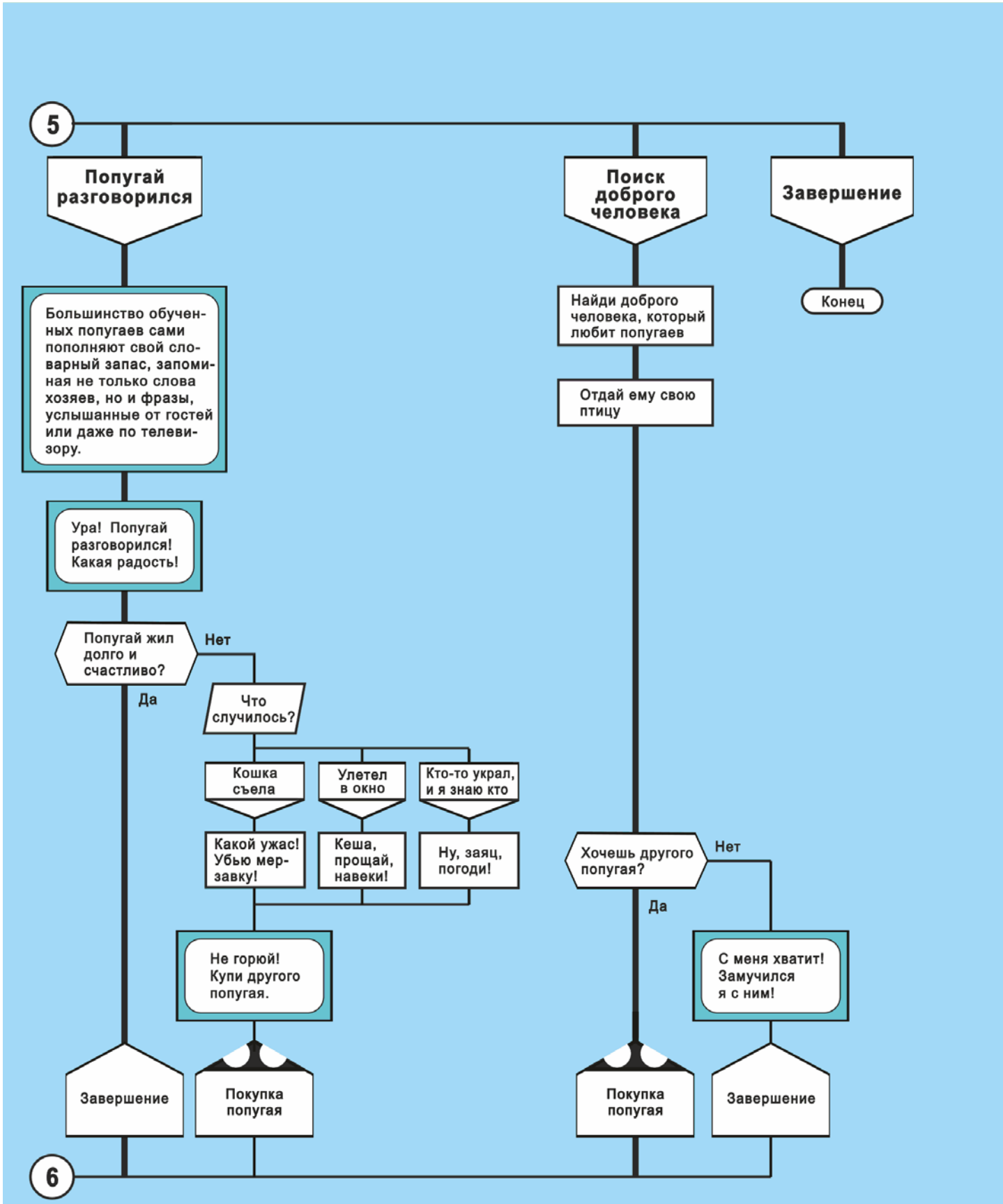


Рис. 14. Алгоритм «Как научить попугая говорить». *Часть 4.*

ВЕТКА «УЧИМ СВОЕ ИМЯ»

Данная ветка интересна тем, что является четырехадресной, т. е. имеет четыре иконы Адрес. В зависимости от условий она может передавать управление в три разных ветки (рис. 15).

Если попугай за один день выучил свое имя, значит, все в порядке, и мы переходим к следующей ветке.

Если же бедняга за один день не справился, завтра ему придется зубрить то же самое. Это значит, что в дело вступает веточный цикл. Бегунок прыгает в начало собственной ветки и снова начинает ее читать сверху вниз.

Наконец, если попугай оказался бестолковым и за целый год так ничего и не усвоил, дело совсем плохо. В этом случае правая икона Адрес заставляет бегунок отправиться на «Поиск доброго человека», которому можно подарить нашу птицу.

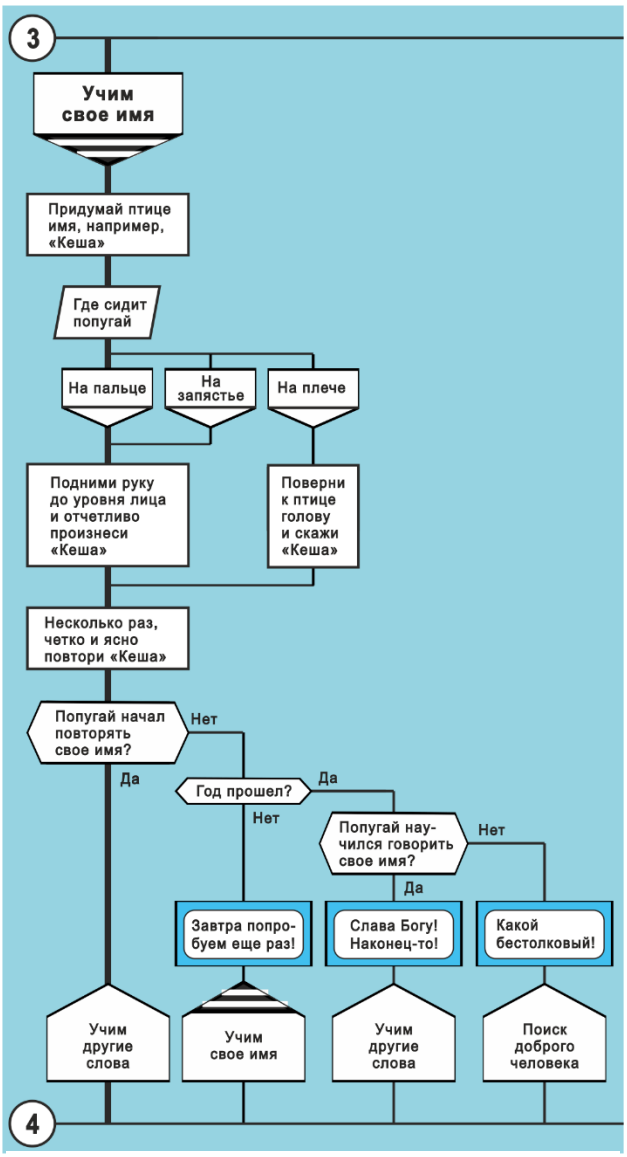


Рис. 15. Ветка «Учим свое имя»

ЗАЧЕМ НУЖНЫ МАРКЕРЫ

Внимательный читатель уже заметил, что в рассказе о попугаях на рис. 14 веточный цикл используется шесть раз. Если все веточные циклы будут носить одинаковую одежду в виде черных треугольников, их будет трудно различить между собой.

По этой причине нужно каждый веточный цикл раскрасить по-своему — с помощью различных маркеров. Для этого нам потребуются шесть маркеров, показанных на рис. 16. Маркеры называются так:

- треугольник,
- белый овал,
- вертикаль,
- горизонталь,
- глаза,
- арка.

Напомним еще раз, что маркеры никак не влияют на работу алгоритма. Однако они приносят большую пользу, позволяя читателю легко ориентироваться в алгоритме и быстро найти каждый веточный цикл.

ТРИ ЗАДАЧИ ВЕТОЧНОГО ЦИКЛА

Веточный цикл позволяет решать три задачи:

- повторное выполнение одной (своей собственной) ветки;

- переход назад в ветку, расположенную левее данной;
- переход назад в одну точку из разных веток.

Повторное исполнение собственной ветки показано на рис. 14 в пяти случаях. Для этой цели веточный цикл организован в следующих ветках:

- Учимся брать корм с ладони *(маркер «Треугольник»);*
- Учимся сидеть на руке *(маркер «Белый овал»);*
- Учимся сидеть на плече *(маркер «Вертикаль»);*
- Учим свое имя *(маркер «Горизонталь»);*
- Учим другие слова *(маркер «Глаза»).*

Переход назад с помощью веточного цикла использован в ветке «Попугай разговорился», где производится возврат в ветку «Покупка попугая».

Второй пример касается ветки «Поиск доброго человека», откуда также делается переход назад в ветку «Покупка попугая».

Таким образом, на рис. 14 возврат управления в ветку «Покупка попугая» выполняется из двух разных мест:

- из ветки «Попугай разговорился»
- и из ветки «Поиск доброго человека».

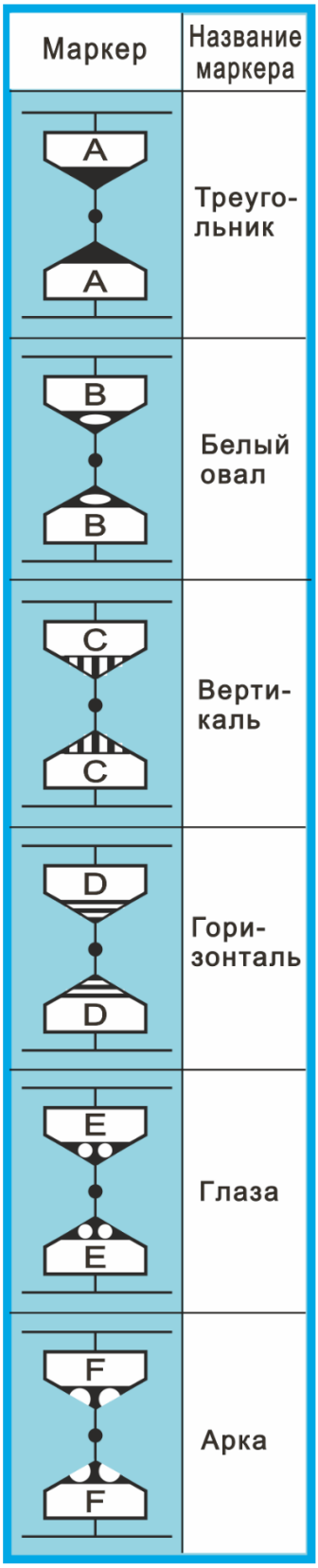


Рис. 16. Маркеры веточных циклов

ПОСЛЕДОВАТЕЛЬНОСТЬ ВЫПОЛНЕНИЯ ВЕТОК

В силуэте имеет место следующий порядок работы:

- Первой работает крайняя левая ветка, последней – крайняя правая;
- Остальные ветки передают управление друг другу слева направо (при этом может случиться так, что некоторые ветки будут пропущены);
- Иногда образуется веточный цикл. Это происходит, когда в иконе Адрес указано имя собственной или одной из левых веток. На рис. 14 веточные циклы помечены маркерами с черными треугольниками.

ШАМПУРЫ КАК ПУТЕВОДИТЕЛЬ

Давайте проследим маршруты, ведущие от начала (икона Заголовок) до конца (икона Конец) на рис. 14. Таких маршрутов много, так как в иконах Вопрос и переключателях могут появляться ответвления, которые увеличивают число маршрутов.

Однако путаница не возникает, потому что в алгоритме есть направляющая нить в виде последовательности шампуров. Шампуры нарисованы легко различимой жирной линией. Они играют роль путеводителя, роль надежной опоры, сверяясь с которой, можно сразу найти правильную дорогу.

ВЫВОДЫ

1. Алгоритм-силуэт делится на ветки, подобно тому, как повесть делится на главы.
2. Каждая ветка имеет только один шампур.
3. Число веток всегда равно числу шампуров.
4. Икона Вставка обозначает алгоритм, который изображен в другом месте.
5. Переключатель содержит икону Выбор и несколько икон Вариант.
6. Веточный цикл есть повторное выполнение ветки.
7. Чтобы создать веточный цикл, во второй иконе Адрес пишут имя своей собственной (или более левой) ветки.
8. Веточный цикл отмечают двумя черными треугольниками.
9. В силуэте может быть до шести веточных циклов.
10. Чтобы их различить, используют шесть различных маркеров.

Глава 3

ИКОНА СОЕДИНИТЕЛЬ И ВЫПУСК ДОКУМЕНТАЦИИ

ПЕЧАТЬ БОЛЬШОГО АЛГОРИТМА НА БУМАЖНОМ НОСИТЕЛЕ

Если алгоритм большой, его печатают на нескольких листах. Алгоритм делят на части, чтобы каждая часть размещалась на отдельном листе, например, формата А4 или А3. Соединительные линии, идущие с листа на лист, сочленяются между собой с помощью иконы Соединитель. Она представляет собой кружок, внутри которого пишут номер (рис. 17).

КАК РАЗМЕЩАТЬ БОЛЬШИЕ АЛГОРИТМЫ В ДОКУМЕНТАХ, КНИГАХ И УЧЕБНИКАХ

В предыдущей главе на рис. 14 показан большой алгоритм про попугаев. Алгоритм разбит на четыре части, занимающие три с половиной книжных разворота.

Соединители нумеруют, начиная с 1. При этом номера входных и выходных соединителей должны совпадать. На рис. 17 показано, что выходные соединители (слева) имеют номера 1 и 2, которые повторяются во входных иконах (справа).

Желательно, чтобы две страницы, образующие разворот, не имели между собой соединительных линий. Исключением являются межстраничные связи через верхнюю и нижнюю шины силуэта, как показано на рис. 14.

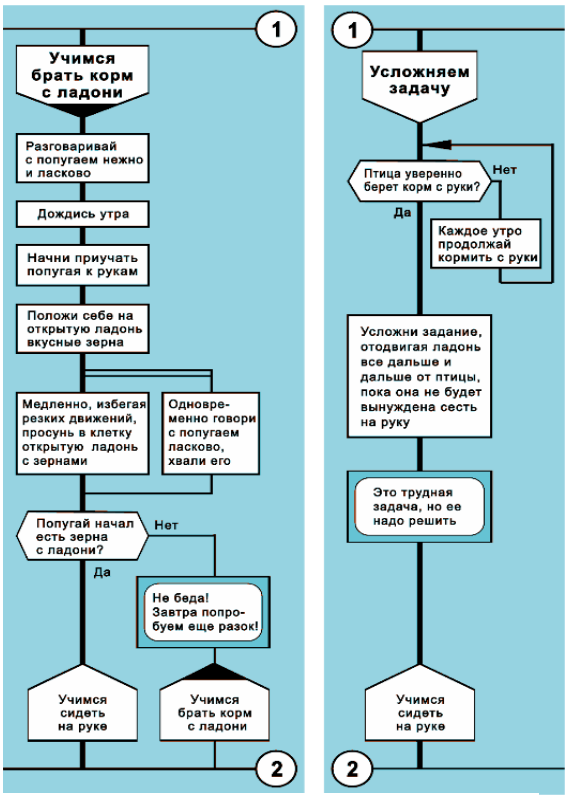


Рис. 17. Переход с листа на лист осуществляется с помощью двух икон Соединитель

ПАМЯТКА

Большие алгоритмы следует размещать в виде силуэта на двух страницах, образующих книжный разворот.

- Если одного разворота мало, следует продолжить силуэт вправо по горизонтали и занять два, три (и более) соседних разворота.
- Соединение двух смежных страниц разворота между собой производится с помощью двух горизонтальных линий (верхней и нижней шины силуэта).
- Сочленение двух разворотов осуществляется с помощью икон Соединитель, как показано на рис. 17 (по два соединителя с каждой стороны).
- Для издания книги, содержащей алгоритмы, желательно использовать книжный формат 70×100 1/16.
- Не следует использовать слишком мелкий шрифт внутри икон. Размер буквы (кегль) должен быть не меньше 6 или 7 типографских пунктов.
- Рекомендуется использовать шрифт без засечек. Например, Arial или Arial Narrow.

КОМПАКТНЫЙ НАБОР ЧЕРТЕЖЕЙ

Желательно иметь компактный набор иллюстраций, демонстрирующих публикацию больших алгоритмов. Такой набор представлен на рис. 18 в виде четырех частей. Рисунок 18 получен из рис. 14 путем уменьшения в четыре раза.

Части силуэта располагаются последовательно, друг за другом. Порядок создается путем нумерации соединителей. Левые соединители разворота называются входными, правые — выходными.

Первый разворот (рис. 18а) имеет только выходные соединители с номерами 1 (вверху) и 2 (внизу).

Второй разворот получил входные номера 1 и 2, а выходные 3 и 4 (рис. 18б).

Третий разворот также имеет четыре соединителя: входные номера 3 и 4, выходные 5 и 6 (рис. 18в).

Последняя часть алгоритма поместилась на одной странице. Она получила лишь входные соединители с номерами 5 и 6 (рис. 18г).

Рассматривая поочередно все четыре части рисунка 18, можно заметить, что они подчиняются правилу: *«Номера выходных соединителей предыдущего разворота совпадают со входными номерами следующего разворота».*

КАК РАЗДЕЛИТЬ РАЗВОРОТ НА ДВЕ СТРАНИЦЫ

Воображаемая вертикальная линия делит книжный разворот пополам, обозначая границу между двумя смежными страницами. Назовем эту линию «переплет».

Как расположить дракон-схему на книжном развороте? Если пустить дело на самотек, алгоритм может получиться некрасивым — из-за того, что линия переплета будет пересекать иконы и соединительные линии. Это плохо. Желательно подобные пересечения исключить. Для этого нужно сдвинуть ветки по горизонтали, чтобы линия переплета оказалась на свободном месте — в промежутке между ветками. Тогда линия переплета не будет касаться веток и пересекать соединительные линии алгоритма.

Переплет может имеет пересечения лишь с двумя горизонтальными линиями — верхней и нижней шиной силуэта (это допустимо).

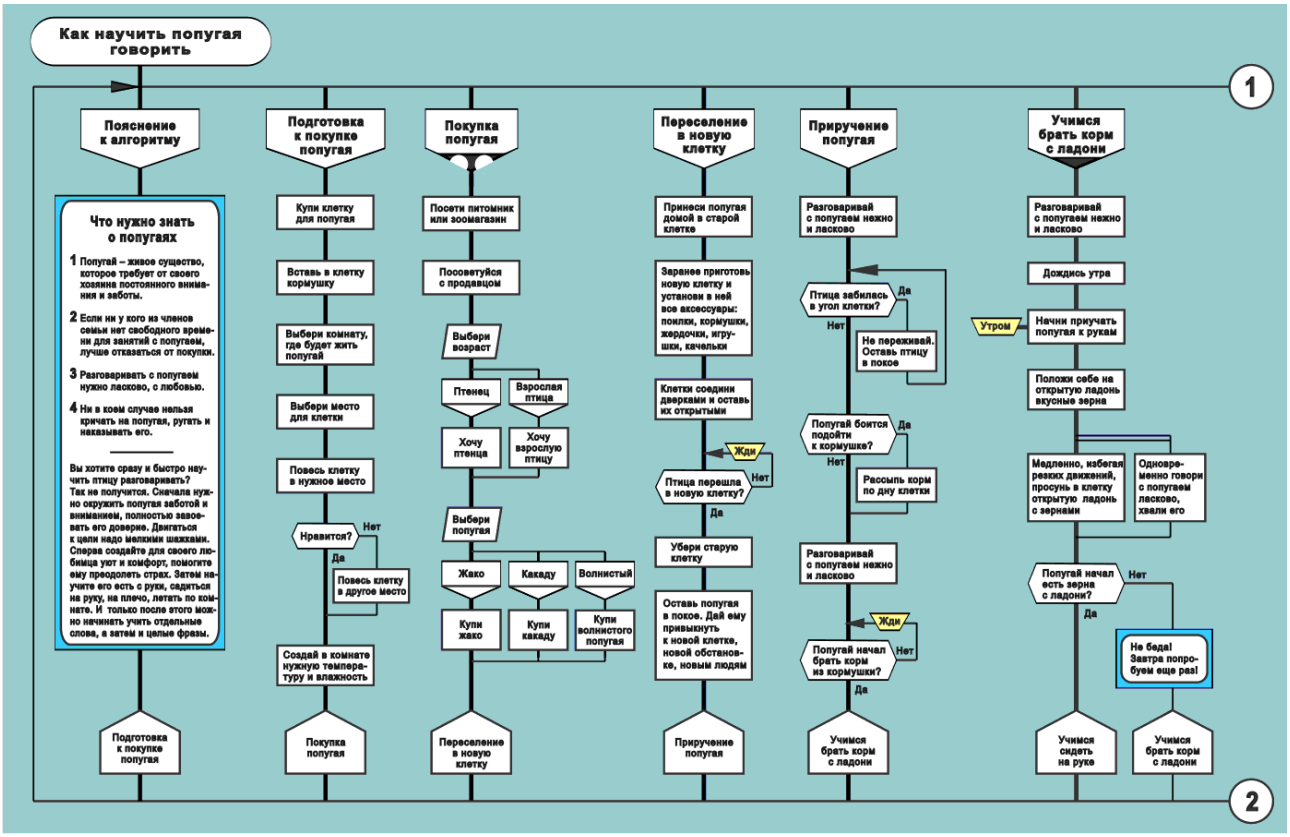


Рис. 18а. *Часть 1.* Алгоритм, подготовленный для публикации на книжном развороте, снабженный 2-мя соединителями (справа)

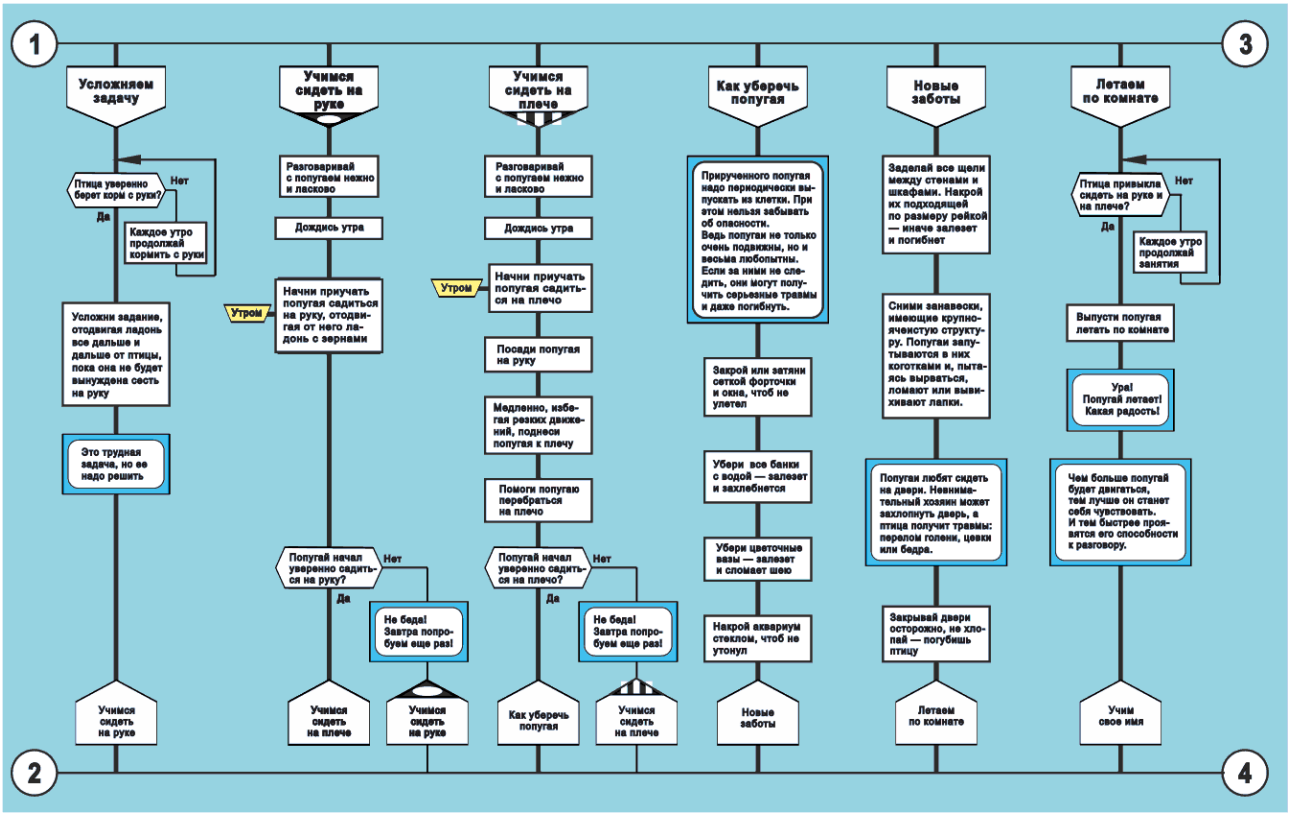


Рис. 18б. *Часть 2.* Алгоритм, подготовленный для публикации на книжном развороте, снабженный 4-мя соединителями (слева и справа)

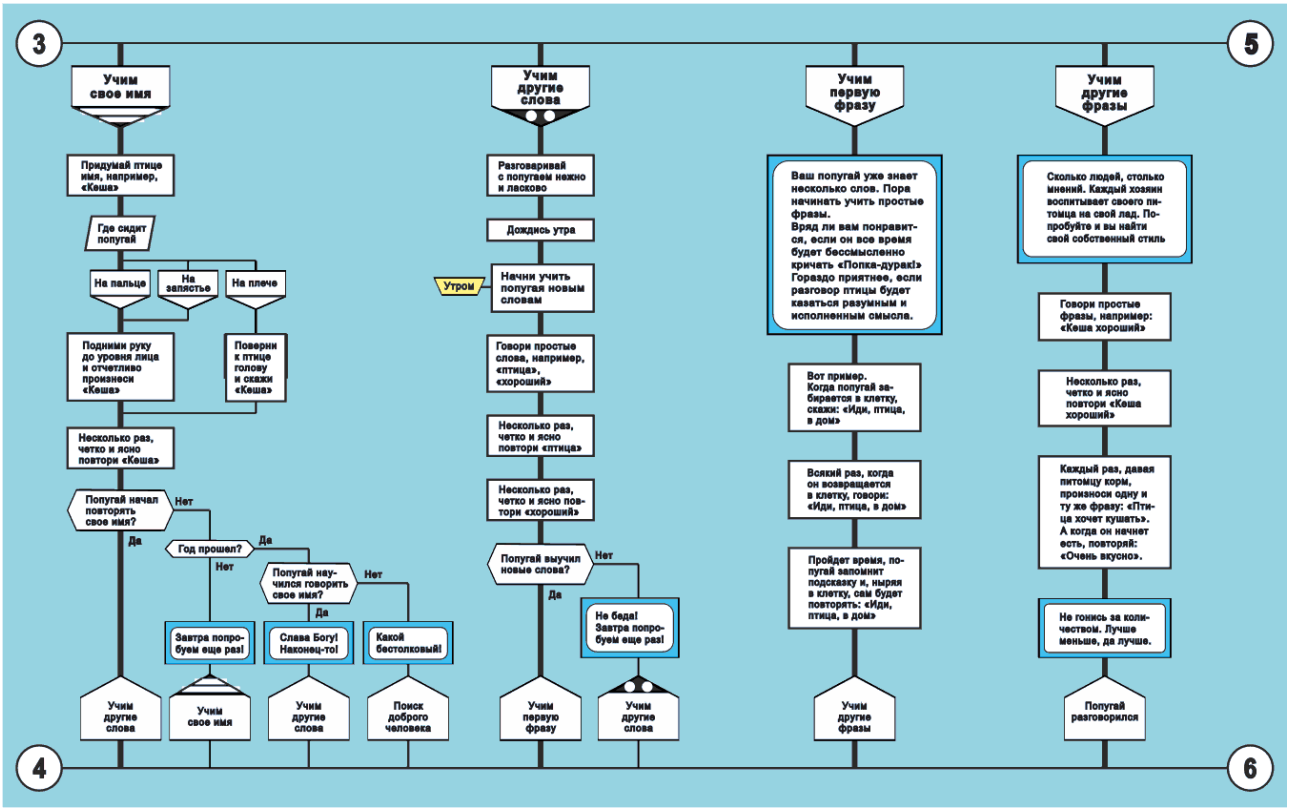


Рис. 18в. *Часть 3.* Алгоритм, подготовленный для публикации на книжном развороте, снабженный 4-мя соединителями (слева и справа)

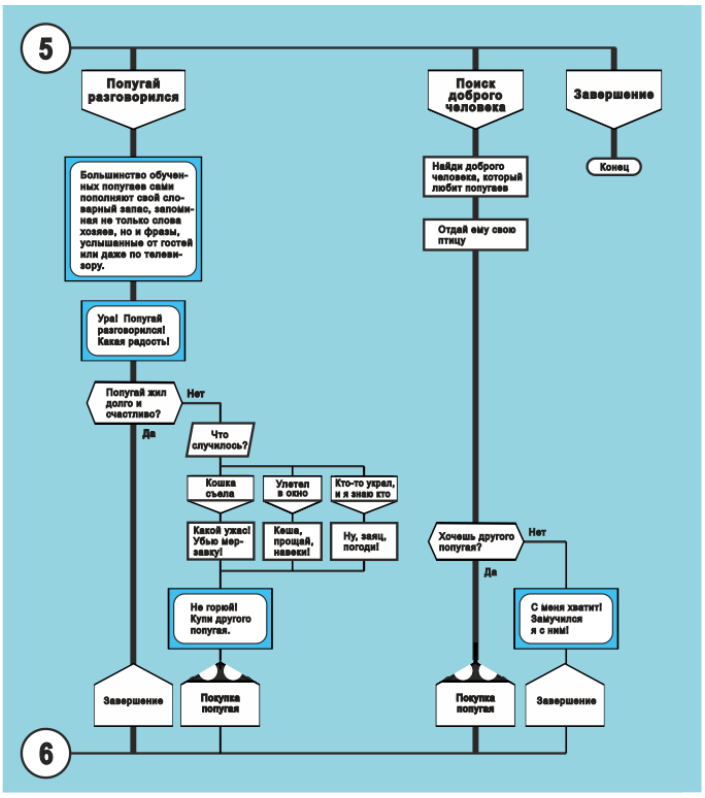


Рис. 18. *Часть 4.* Алгоритм, подготовленный для публикации на книжном развороте, снабженный 2-мя соединителями (слева)

ЧТО МЕШАЕТ ПОНИМАНИЮ

В литературе опубликовано большое количество важных, интересных и полезных алгоритмов. К сожалению, алгоритмы обычно пишут на сложных языках, понятных только избранным, что отпугивает многих читателей. Алгоритмы — большая ценность, однако они далеко не всегда могут преодолеть барьеры непонимания и найти дорогу к сердцам потребителей.

Язык ДРАКОН призван изменить сложившуюся практику и заменить сложный язык простым. Это позволит проложить новые, удобные и широкие дороги для распространения алгоритмических знаний.

Публикация алгоритмов на языке ДРАКОН значительно расширит круг посвященных и приобщит к «тайному знанию» массовую аудиторию. Алгоритмическое мышление — наконец-то! — станет доступным для всех образованных людей.

ПОНИМАЕМОСТЬ АЛГОРИТМА

Понятность алгоритма для человека (который его изучает) — это удобочитаемость, ясность, доходчивость, пригодность для быстрого понимания.

В ГОСТе используется термин «понимаемость алгоритма». Понимаемость есть совокупность свойств алгоритма, характеризующая затраты усилий пользователя на понимание логической концепции этого алгоритма[1].

ВЫВОДЫ

1. Цель языка ДРАКОН — облегчить и ускорить понимание алгоритмов.
2. Понимаемость алгоритма — свойство алгоритма минимизировать интеллектуальные усилия, необходимые для его понимания при зрительном восприятии алгоритма.
3. Традиционная запись алгоритма трудна для понимания. Глядя на сложный алгоритм, понять его трудно. А быстро понять — невозможно.
4. ДРАКОН стремится упорядочить и представить решение сложной императивной проблемы в виде наглядных чертежей, выполненных по принципу «Взглянул — и сразу понял!»
5. Большие алгоритмы следует размещать в документации в виде силуэта на двух страницах, образующих книжный разворот.
6. Если места не хватает, нужно продолжить алгоритм-силуэт вправо и занять два или более соседних разворота.
7. Соединение двух страниц книжного разворота производится с помощью верхней и нижней шины силуэта.
8. Сочленение двух соседних разворотов осуществляется с помощью икон Соединитель.
9. Номера выходных соединителей предыдущего разворота должны совпадать со входными номерами следующего разворота.

---

[1] Термин «понимаемость» (understandability) определен в стандарте «ГОСТ 28806-90 Качество программных средств. Термины и определения» [96].

Глава 4

СПРАВОЧНИК: ГРАФИЧЕСКИЕ ФИГУРЫ ЯЗЫКА ДРАКОН

ЗАЧЕМ НУЖЕН СПРАВОЧНИК

При анализе дракон-алгоритмов иногда возникает необходимость вспомнить, как называется та или иная фигура. И зачем она нужна.

Для этого служит справочник фигур, представленный в главе.

ИКОНЫ ЯЗЫКА ДРАКОН

ДРАКОН — графический язык. Буквами языка являются геометрические фигуры, которые называются «иконы». Всего имеется 31 икона. Назначение икон показано на рис. 19 и 20.

Иконы должны быть заполнены текстом. Если текст отсутствует (икона пустая), значит, это ошибка.

Чаще всего используются две иконы: *Действие* и *Вопрос*.

В иконе Действие пишут команду в повелительном наклонении, например:

- Включи двигатель.
- Приготовь борщ.
- Обеспечь проходимость дыхательных путей.

В иконе Вопрос пишут да-нетный вопрос, т. е. вопрос, имеющий только два ответа: Да и Нет. Например:

- В кошельке есть деньги?
- Есть ли реакция на прикосновение?
- Есть ли дефибриллятор?

Двенадцать часто используемых фигур показаны на рис. 12.

КОММЕНТАРИИ

Для комментариев используют три иконы:

- Комментарий (рис. 20, п. 29);
- Выноска (рис. 20, п. 30);
- Пояснение (рис. 19, пункт 11).

С иконой Комментарий мы хорошо знакомы — ее мы уже видели на рис. 5-7, 13-15, 17, 18. Икона Выноска размещается на любом свободном месте дракон-схемы, указывая своим острием на икону-хозяина.

Икона Пояснение играет роль правого комментария. Она присоединяется справа от иконы, которая нуждается в дополнительной информации или примечании. Если же ее прицепить к иконе Заголовок, она называется «Формальные параметры», как это принято в программировании.

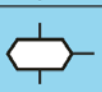
	Икона	Название иконы	Пояснение
1		Заголовок	В иконе «Заголовок» пишут полное название алгоритма, например: «Аварийное выключение ракетного двигателя»
2		Конец	В этой иконе пишут слово «Конец»
3		Действие	Указывают действие или команду, которую должен выполнить компьютер или человек
4		Вопрос	Да-нетный вопрос, т. е. вопрос, на который можно ответить либо Да, либо Нет. Все другие ответы запрещены
5		Выбор	Фраза (или вопрос), приглашающая выбрать один из вариантов
6		Вариант	Здесь пишут один из вариантов. (Число рассматриваемых вариантов равно числу икон «Вариант»)
7		Имя ветки	Эта икона обозначает начало ветки. В ней находится название ветки. (Ветка — это структурная часть алгоритма)
8		Адрес	Икона «Адрес» обозначает конец любой ветки, кроме последней. Она показывает, в какую следующую ветку надо идти
9		Вставка	Икона «Вставка» говорит, что в этом месте из алгоритма вынут «кусок», который перенесён в другое место. В иконе пишут название этого «куска»
10		Полка	Сверху пишут указание, например: «Установить признак», снизу пояснение, например: «Можно ехать налево»
11		Формальные параметры	Икона присоединяется справа к иконе Заголовок
		Пояснение	Любые сведения, поясняющие икону, находящуюся слева от данной
12		Начало цикла ДЛЯ	Для цикла for пишут переменную цикла, ее начальное и конечное значения, шаг. Иконы 12 и 13 используются также в цикле foreach
13		Конец цикла ДЛЯ	Внутри иконы пишут «Конец цикла». Запрещено оставлять икону пустой
14		Вывод	Вывод данных, передача информации. Выдача команд на исполнительные органы в системах реального времени
15		Простой вывод	Икона «Простой вывод» позволяет упростить дракон-схему при необходимости
16		Ввод	Ввод данных, прием информации
17		Простой ввод	Икона «Простой ввод» позволяет упростить дракон-схему при необходимости
18		Пауза	Пауза задерживает выполнение следующего действия. Время задержки пишут внутри иконы.
19		Период	Период повторения действия, которое выполняется в цикле ЖДАТЬ
20		Таймер	Если в иконе написано A = 0, это значит, что данная икона создает, обнуляет и запускает таймер A

Рис. 19. Иконы языка ДРАКОН

	Икона	Название иконы	Пояснение
21		Синхронизатор	Останавливает процесс, ожидая появления заданного момента времени
		Время	Если в иконе Время написано «10 секунд», это значит, что действие в иконе справа следует выполнить за 10 секунд или меньше
22		Время группы	Длительность выполнения группы действий. Не одного действия, а именно группы. Группа состоит из двух и более действий
23		Время группы *справа*	Икона «Время группы» присоединяется справа
24		Начало контрольного срока	В иконе Начало пишут контрольное время критической процедуры. Например, «30 сек».
25		Конец контрольного срока	Указывают окончание контрольного времени. Например, «Прошло 30 сек».
26		Начало параллельных действий	Означает НАЧАЛО нескольких действий, которые могут начинаться и выполняться как синхронно, так и в любом порядке
27		Конец параллельных действий	Означает момент времени, когда закончилось самое последнее действие из числа параллельных действий
28		Параллельный процесс	Осуществляет пуск, останов, приостановку и рестарт параллельного процесса
29		Комментарий	Комментарий — это не действие. Это различные пояснения и подсказки, которые помогают быстрее понять алгоритм
30		Выноска	Выноска содержит комментарий и может присоединяться к любой иконе с любой стороны
31		Соединитель	Икона «Соединитель» используется при переходе с листа на лист (когда алгоритм размещается на нескольких листах)

Рис. 20. Иконы языка ДРАКОН. *Продолжение*

ИКОНЫ РЕАЛЬНОГО ВРЕМЕНИ

Язык ДРАКОН позволяет управлять процессами реального времени. Для этого имеется ряд фигур, обозначающих время.

1. Вот первая группа икон:

- Пауза (рис. 19, п. 18);
- Период (рис. 19, п. 19);
- Таймер (рис. 19, п. 20);

2. Синхронизатор и Время — два разных названия для одной и той же иконы (рис. 20, п. 21). Причина в том, что данная икона может выполнять существенно разные функции. Но недоразумений не возникает, так как в иконе Синхронизатор пишут латинскую букву, обозначающую таймер, и знака =.

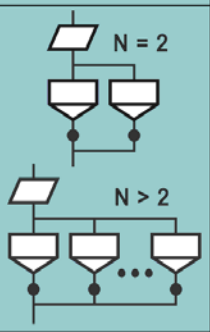
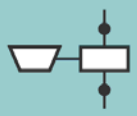
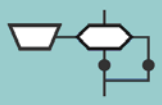
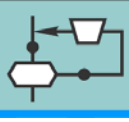
	Макроикона	Название макроиконы	Пояснение
1		Заголовок с параметрами	Заголовок с формальными параметрами
2		Развилка	Развилка на два направления. Черные точки — это валентные точки. В эти точки можно вставлять иконы, например, икону «Действие». Хотя бы одна валентная точка должна быть заполнена.
3		Переключатель (число вариантов 2 и больше)	Переключатель — это часть алгоритма, имеющая один вход вверху и один выход внизу. Внутри переключателя алгоритм разветвляется на несколько дорожек. Число дорожек равно двум и более. Переключатель строится с помощью иконы «Выбор» и нескольких икон «Вариант». Под каждой иконой «Вариант» имеется валентная точка.
4		Цикл со стрелкой	Цикл нужен для того, чтобы повторять действия. Повторение прекращается, когда будет выполнено условие. Условие записывают в иконе «Вопрос».
5		Веточный цикл	Веточный цикл нужен для того, чтобы повторять действия, расположенные в одной или нескольких ветках.
6		Действие с заданной длительностью	Справа нарисована икона «Действие». Слева к ней прицеплена икона «Время». Макроикона 6 показывает, что время действия не должно превышать заданную величину
7		Решение с заданной длительностью	Справа нарисована «Развилка». Слева к ней прицеплена икона «Время». Макроикона 7 показывает, что человек должен принять решение за указанное время.
8		Длительность группы действий. *Время слева*	Справа нарисовано не одно действие, а группа, состоящая из двух (и более) действий. Слева к этой группе присоединена икона «Время группы». Макроикона 8 показывает, что время группы действий жестко задано.
9		Длительность группы действий. *Время справа*	То же самое, что в пункте 8. Отличие в том, что икона «Время группы» присоединена не слева, а справа.
10		Параллельная работа	Означает синхронную или не синхронную работу включая: — начало параллельных работ, — выполнение параллельных работ, — конец параллельных работ.
11		Цикл ДЛЯ	Цикл for (цикл со счетчиком) или цикл foreach (совместный цикл, или цикл по коллекции)
12		Цикл ЖДАТЬ	Цикл периодически проверяет условие в ожидании события. Когда событие наступило, происходит выход из цикла

Рис. 21. Макроиконы языка ДРАКОН

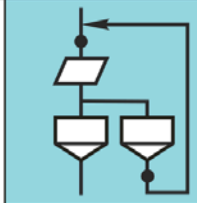
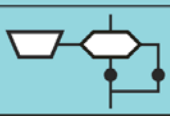
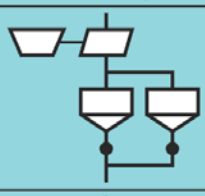
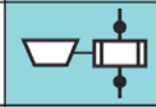
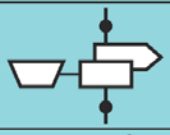
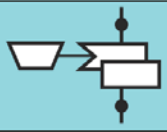
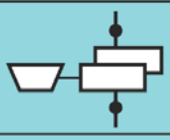
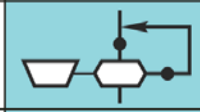
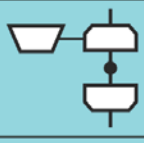
	Макроикона	Название макроиконы	Пояснение
13		Переключающий цикл	Цикл, построенный с помощью переключателя
14		Действие по таймеру	Справа нарисована икона «Действие». Слева к ней прицеплена икона «Синхронизатор», который отсчитывает время по таймеру. Макроикона 14 задерживает выполнение действия до момента времени, указанного в синхронизаторе
15		Развилка по таймеру	Макроикона 15 задерживает выполнение развилки до момента времени, указанного в синхронизаторе
16		Переключатель по таймеру	Макроикона 16 задерживает выполнение переключателя до момента времени, указанного в синхронизаторе
17		Вставка по таймеру	Макроикона 17 задерживает выполнение вставки до момента времени, указанного в синхронизаторе
18		Вывод по таймеру	Макроикона 18 задерживает выполнение вывода (выдачи команды на исполнительный орган) до момента времени, указанного в синхронизаторе
19		Ввод по таймеру	Макроикона 19 задерживает выполнение ввода до момента времени, указанного в синхронизаторе
20		Параллельный процесс по таймеру	Макроикона 20 задерживает выполнение параллельного процесса до момента времени, указанного в синхронизаторе
21		Цикл со стрелкой по таймеру	Макроикона 21 задерживает выполнение цикла со стрелкой до момента времени, указанного в синхронизаторе
22		Цикл ДЛЯ по таймеру	Макроикона 22 задерживает выполнение цикла ДЛЯ (for и foreach) до момента времени, указанного в синхронизаторе
23		Цикл ЖДАТЬ по таймеру	Макроикона 23 задерживает выполнение цикла ЖДАТЬ до момента времени, указанного в синхронизаторе

Рис. 22. Макроиконы языка ДРАКОН. *Продолжение*

При программировании используют Синхронизатор (но не Время). Внутри иконы Синхронизатор пишут, например: A = 3мин20с (3 минуты 20 секунд). Здесь буква A (или B, или C) обозначает имя таймера. Синхронизатор задерживает выполнение иконы-хозяина до тех пор, пока таймер не отсчитает 3 минуты 20 секунд.

При записи нестрогого алгоритма (жизнеритма) используют Время (но не Синхронизатор). Разница в том, что латинская буква A (или B, или C), а также знак =

отсутствуют. Если внутри иконы Время написано 10с, это значит, что действие, указанное в иконе-хозяине, следует выполнить за время 10 секунд (не больше). Таким образом, в иконе Время указывают лимит времени, отведенный на данную операцию. Это важно, например, для врачей, выполняющих реанимацию.

3. Иногда нужно указать длительность не одного действия, а группы действий. Причем группа может состоять из двух, трех и более действий, выполняемых последовательно, друг за другом. В таком случае используют иконы:

- Время группы (рис. 20, п. 22);
- Время группы справа (рис. 20, п. 23).

4. Бывает так, что необходимо задать лимит времени для последовательности действий, но начало и конец подобной процедуры находятся на дракон-схеме далеко друг от друга. В таком случае нужно иметь две иконы, обозначающих начало и конец процедуры. Для этого служат иконы:

- Начало контрольного срока (рис. 20, п. 24);
- Конец контрольного срока (рис. 20, п. 25).

МАКРОИКОНЫ ЯЗЫКА ДРАКОН

ДРАКОН имеет не только мелкие фигурки (иконы), но и крупные, составные; они называются *макроиконы.*

Подобно тому, как слова состоят из букв, макроиконы (графические слова) состоят из икон (графических букв). Язык ДРАКОН имеет 23 макроиконы (рис. 21 и 22).

Иконы и макроиконы — это строительные блоки, из которых составляются дракон-алгоритмы.

ВАЛЕНТНЫЕ ТОЧКИ

Важной частью дракон-схем служат *валентные точки* (на рис. 21 и 22 они показаны как черные кружки). В эти точки последовательно вводятся иконы и макроиконы, которые в совокупности образуют графический узор. После заполнения икон текстом графический узор превращается в дракон-алгоритм.

МАРКЕРЫ ЯЗЫКА ДРАКОН

Маркеры нужны для того, чтобы легко различать веточные циклы при зрительном восприятии.

В сложном медицинском алгоритме могут появиться несколько веточных циклов. Как отличить их друг о друга?

Желательно иметь признак, позволяющий моментально, *по внешнему виду* обнаружить, где один цикл, а где другой. Примерно так же, как мы, не задумываясь, отличаем кошку от собаки.

Отличительным признаком служит маркер. Предусмотрены шесть маркеров, показанных на рис. 16.

Приятным сюрпризом служит тот факт, что пользователь не должен заботиться о маркерах. Они появляются в дракон-алгоритме автоматически. Расстановку маркеров выполняет программа «ДРАКОН-конструктор».

ВЫВОДЫ

1. Данная глава представляет собой справочник. При анализе дракон-алгоритмов у читателя могут возникнуть вопросы: как называется та или иная фигура. Глава позволит быстро разобраться и сэкономить драгоценное время.
2. Графические фигуры ДРАКОНа делятся на три части:
    - иконы.
    - макроиконы.
    - маркеры.
3. Иконы и макроиконы — это строительные блоки, из которых собирают дракон-алгоритмы.
4. Маркеры никак не влияют на правильность алгоритмов. Они облегчают чтение и зрительное восприятие сложных графических чертежей.

Глава 5

ПРИМИТИВ И СИЛУЭТ

ЧТО ТАКОЕ ПРИМИТИВ

В языке ДРАКОН есть два логически законченных алгоритмических чертежа.

- силуэт (о котором шла речь выше);
- примитив.

Примитив

- Это составная алгоритмическая структура, у которой иконы Заголовок и Конец лежат на одной вертикали.
- В отличие от силуэта примитив не имеет веток.

Примитивы показаны на рис. 1, 2, 23-26.

ШАМПУР ПРИМИТИВА

Шампур примитива

- Это вертикальная линия, соединяющая иконы Заголовок и Конец.
- Между ними на той же линии помещается одна или несколько других икон.

Правило для примитива

Выход иконы Заголовок и вход иконы Конец должны находиться на одной вертикали (на шампуре)

Если правило выполняется, дракон-схема становится упорядоченной, эргономичной, легкой для чтения. И наоборот, нарушение правила делает схему корявой, изломанной, неудобной для глаза.

ЧТО ТАКОЕ ГЛАВНЫЙ МАРШРУТ

Вспомним, что маршрут — это путь, ведущий от начала до конца алгоритма.

На рис. 1 показан неразветвленный алгоритм. В нем всего один маршрут. В разветвленном алгоритме на рис. 23 таких маршрутов четыре. Если тропинок несколько, среди них можно выделить главный и боковые маршруты.

Главный маршрут алгоритма

Это путь от начала до конца алгоритма, который ведет к наибольшему успеху (happy path)

На рис. 23 главный маршрут проходит по левой вертикали. В самом деле, левая вертикаль означает, что дела идут хорошо, ибо человек здоров. Остальные маршруты описывают нежелательную или даже плохую ситуацию.

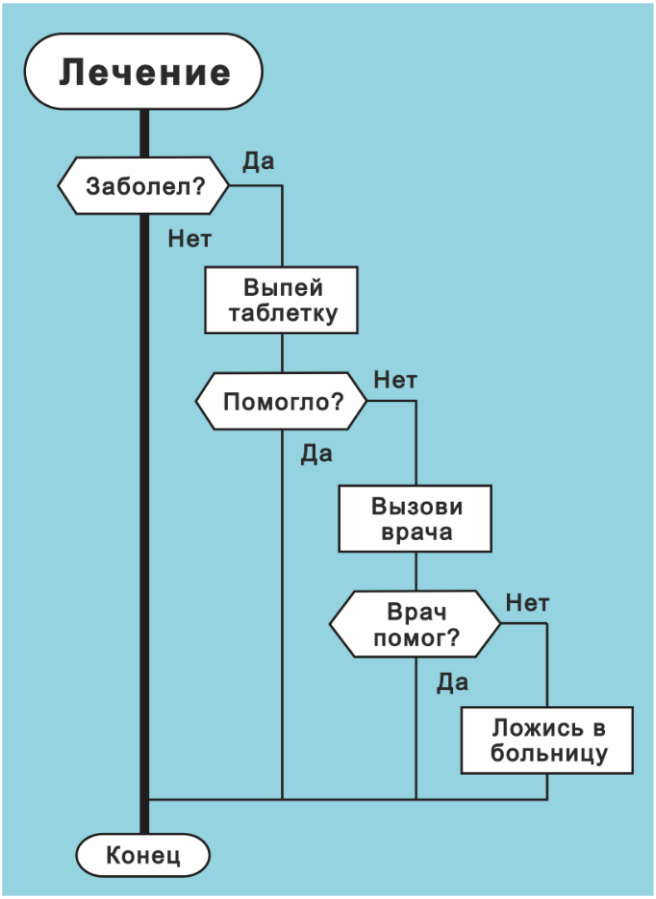


Рис. 23. Маршруты упорядочены слева направо

ПРАВИЛО ГЛАВНОГО МАРШРУТА

Рассмотрим задачу. В запутанном лабиринте, соединяющем начало и конец сложного алгоритма, нужно выделить один-единственный маршрут — царскую дорогу, путеводную нить. С ней можно зрительно сравнивать все прочие маршруты, чтобы легко сориентироваться в задаче и не заблудиться в путанице развилок.

Путеводная нить должна быть визуально легко различима. Бросив беглый взгляд на дракон-схему, мы должны обнаружить четкие ориентиры, позволяющие сразу и безошибочно увидеть «царский» маршрут и упорядоченные относительно него остальные маршруты. Для этого служит правило главного маршрута. Оно гласит: *главный маршрут должен идти по шампуру.*

Это значит, что царский маршрут не может оказаться где-то на задворках дракон-схемы, где его днем с огнем не сыскать. Нет, он всегда должен находиться на самом почетном месте — на крайней левой вертикали. Соблюдение правила делает схему зрительно упорядоченной, предсказуемой и интуитивно ясной.

Если правило главного маршрута по каким-то причинам оказалось нарушенным (рис. 25), нужно поменять местами слова «Да» и «Нет» в развилках, а также присоединенные к ним гирлянды икон. Действуя таким путем, всегда можно добиться, чтобы на царском пути оказался тот выход иконы Вопрос, который ведет к наибольшему успеху (рис. 24).

Правило главного маршрута	Главный маршрут алгоритма должен идти по шампуру

ПРАВИЛО БОКОВЫХ МАРШРУТОВ

Боковые маршруты нужно рисовать справа от шампура по принципу: *«Чем правее, тем хуже».*

Дракон-алгоритм превращает хаос в порядок. Для наглядности эта мысль развернута на рис. 23. Все маршруты упорядочены согласно правилу: «Чем правее расположен маршрут, тем более неприятную ситуацию он описывает».

Самая первая, крайняя слева вертикаль — это шампур. По ней идет главный маршрут, имеющий оценку «хорошо», потому что человек здоров (рис. 23). Вторая

вертикаль описывает легкое недомогание, которое можно снять таблеткой. Третья вертикаль говорит: самочувствие ухудшилось, нужен врач. Наконец, крайняя правая вертикаль отражает самую неблагоприятную ситуацию – пришлось лечь в больницу.

Что такое боковой маршрут	Это любой маршрут разветвленного алгоритма за исключением главного
Правило боковых маршрутов	Боковые маршруты алгоритма нужно рисовать справа от шампура по принципу: *«Чем правее — тем хуже».*

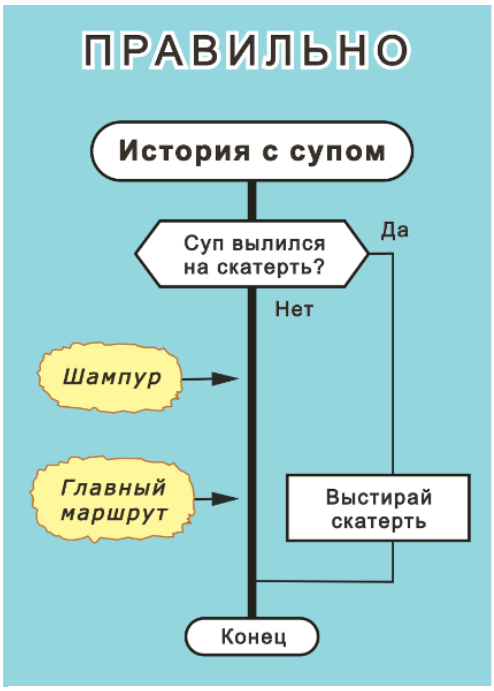


Рис. 24. Хорошая (эргономичная) схема. Главный маршрут идет по шампуру

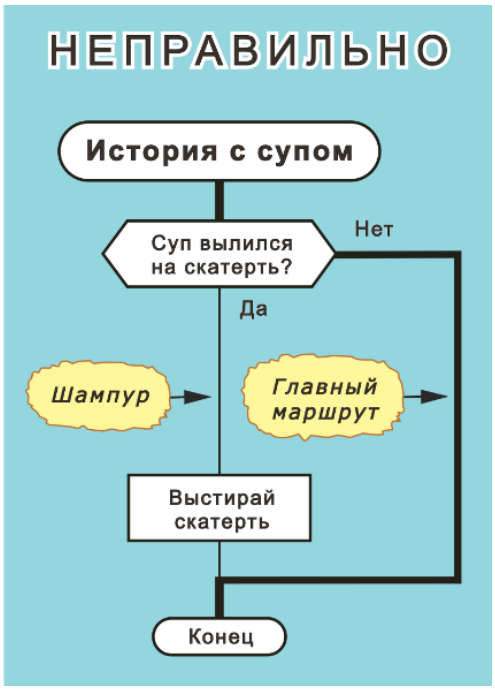


Рис. 25. Плохая схема. Нарушено правило «Главный маршрут должен идти по шампуру»

В ЛЮБОМ ДЕЛЕ НУЖЕН ПОРЯДОК

Чтобы алгоритм был понятным, он должен быть стройным, красивым, упорядоченным, то есть эргономичным. Он не должен содержать непредсказуемые и хаотичные хитросплетения линий и икон.

Язык ДРАКОН был разработан, в частности, потому, что традиционные блок-схемы алгоритмов, с эргономической точки зрения, не выдерживают критики. Они напоминают лабиринт, в котором легко запутаться.

ДРАКОН выгодно отличается тем, что его графический узор подчиняется жестким и тщательно продуманным правилам, которые дисциплинируют мышление, облегчают умственный труд.

Чтобы убедиться в этом, взглянем еще раз на рис. 23 и проведем взглядом по всем вертикалям слева направо. Мы обнаружим не хаос, а строгий порядок. Потому что маршруты нарисованы не как попало, а по правилам. В результате чертеж алгоритма обретает четкую зрительно-смысловую структуру, которая облегчает работу мысли. Читая такую схему, человек не станет плутать в потемках, так как при ее составлении использовано мудрое правило: *«Все хорошее — слева, все плохое — справа».*

Про такой алгоритм можно сказать: «Все ясно, как на ладони!»

ЧТО ДЕЛАТЬ, ЕСЛИ ПРИНЦИП «ЧЕМ ПРАВЕЕ, ТЕМ ХУЖЕ» НЕ РАБОТАЕТ

Что ж, бывает и такое. Ничего страшного. Надо придумать какой-нибудь другой принцип, подходящий к вашей задаче. Например: чем правее, тем ниже, или, наоборот, тем выше. Идея проста. Смещение вправо от главного маршрута должно быть не произвольным и хаотичным, а продуманным и логичным.

Годятся любые правила, позволяющие сделать схему упорядоченной. Для разных задач могут понадобиться разные правила. Но у всех правил есть общая черта. В схеме должен быть не хаос, а порядок. Здесь действует

Правило хорошей хозяйки	Если постараться, порядок всегда можно навести

ЧТО ЛУЧШЕ: ПРИМИТИВ ИЛИ СИЛУЭТ?

Конечно, силуэт. Силуэт является мощным и чрезвычайно эффективным орудием языка ДРАКОН.

Примитив применяют в самых простых случаях, когда число икон в дракон-схеме не превышает 10 – 15 штук. Во всех остальных случаях следует использовать силуэт.

ПРИМИТИВ ХОРОШ ДЛЯ ОБЪЯСНЕНИЙ

В данной книге мы используем примитив для удобства читателей, чтобы на простых примерах объяснить правила языка ДРАКОН.

Вот пример. Ветку силуэта (не весь силуэт, а лишь одну ветку) можно сравнить с примитивом. Что для этого нужно?

На рисунке 14 мы видели силуэт из 19 веток. Из них только 11 веток имеют одну икону Адрес. Выберем из них ветку «Подготовка к покупке попугая» (рис. 8).

А теперь сделаем следующее. На рис. 8 удалим две иконы: Имя ветки и Адрес, а вместо них вставим иконы Заголовок и Конец. В результате ветка на рис. 8 превратится в примитив на рис. 26. Узнаете?

Мы убедились, что любая одноадресная ветка силуэта является аналогом примитива.

Подведем итоги. В языке ДРАКОН есть две конструкции: силуэт и примитив. На практике в подавляющем большинстве случаев используют силуэт для средних, сложных и сверхсложных задач. Примитив используется редко, для самых простых алгоритмов.

Рис. 26. Примитив, полученный из ветки на рис. 8

ВЫВОДЫ

1. Язык ДРАКОН имеет две алгоритмические структуры:
    - силуэт;
    - примитив.
2. Примитив — это малый алгоритм, который, в отличие от силуэта, не имеет веток.
3. Следует различать:
    - главный маршрут;
    - боковые маршруты.
4. Главный маршрут идет по шампуру.
5. Боковые маршруты находятся справа от главного. Они располагаются слева направо по принципу: «Чем правее, тем хуже».
6. Если указанный принцип нельзя применить, боковые маршруты должны подчиняться правилу: «Смещение вправо от главного маршрута должно быть не произвольным, а упорядоченным».

Глава 6

ПРЕОБРАЗОВАНИЯ, ПОЗВОЛЯЮЩИЕ УЛУЧШИТЬ ПОНЯТНОСТЬ АЛГОРИТМОВ

На ошибках мы горим! —
Мне сказал Алеха.
Непонятный алгоритм —
Это очень плохо.
Мы ошибки победим!
Надо сделать вот чего —
Надо сделать алгоритм
Ясным и доходчивым!

Юрий Примашев

ОШИБКИ И НЕПОНЯТНЫЕ АЛГОРИТМЫ

Ошибки в алгоритмах приносят большой вред. Если алгоритм трудно понять, можно не заметить затаившуюся ошибку, которая повлечет неприятности. Чтобы сократить число ошибок, нужно овладеть особым искусством — создавать алгоритмы легкие для понимания. Можно ли этому научиться?

В этой главе мы познакомимся с эффективными приемами, позволяющими улучшить понимаемость алгоритмов. В результате алгоритмы станут удобочитаемыми, легкими для понимания и удобными для работы.

УДАЛЕНИЕ ПОВТОРОВ

Иногда бывает, что в дракон-схеме иконы повторяются. Например, на рис. 27 икона «Съешь кашу» встречается три раза. Это нехорошо. Навязчивые повторения раздражают читателя, которому приходится тратить время и несколько раз читать одно и то же.

К счастью, некоторые повторы можно устранить. Такая возможность появляется, если одинаковые иконы находятся рядом, причем их выходы соединены между собой (рис. 27). В этом случае действует

Правило. Повторы запрещены

Устранение повторов производится с помощью горизонтальной линии и называется *горизонтальное объединение* (рис. 28).

При этой операции несколько икон объединяются в одну. Для этого нужно:

- выделить две или более одинаковых икон, выходы которых присоединены к нижней горизонтальной линии;
- объединить входы упомянутых икон горизонтальной линией,
- удалить все одинаковые иконы, кроме крайней левой.

ВЕРТИКАЛЬНОЕ ОБЪЕДИНЕНИЕ

На рис. 28 есть недостаток — повторяются два белых квадрата с развилкой «Есть можно?» Чтобы убрать дефект, надо использовать другой прием — на этот раз уже не горизонтальное, а *вертикальное объединение.*

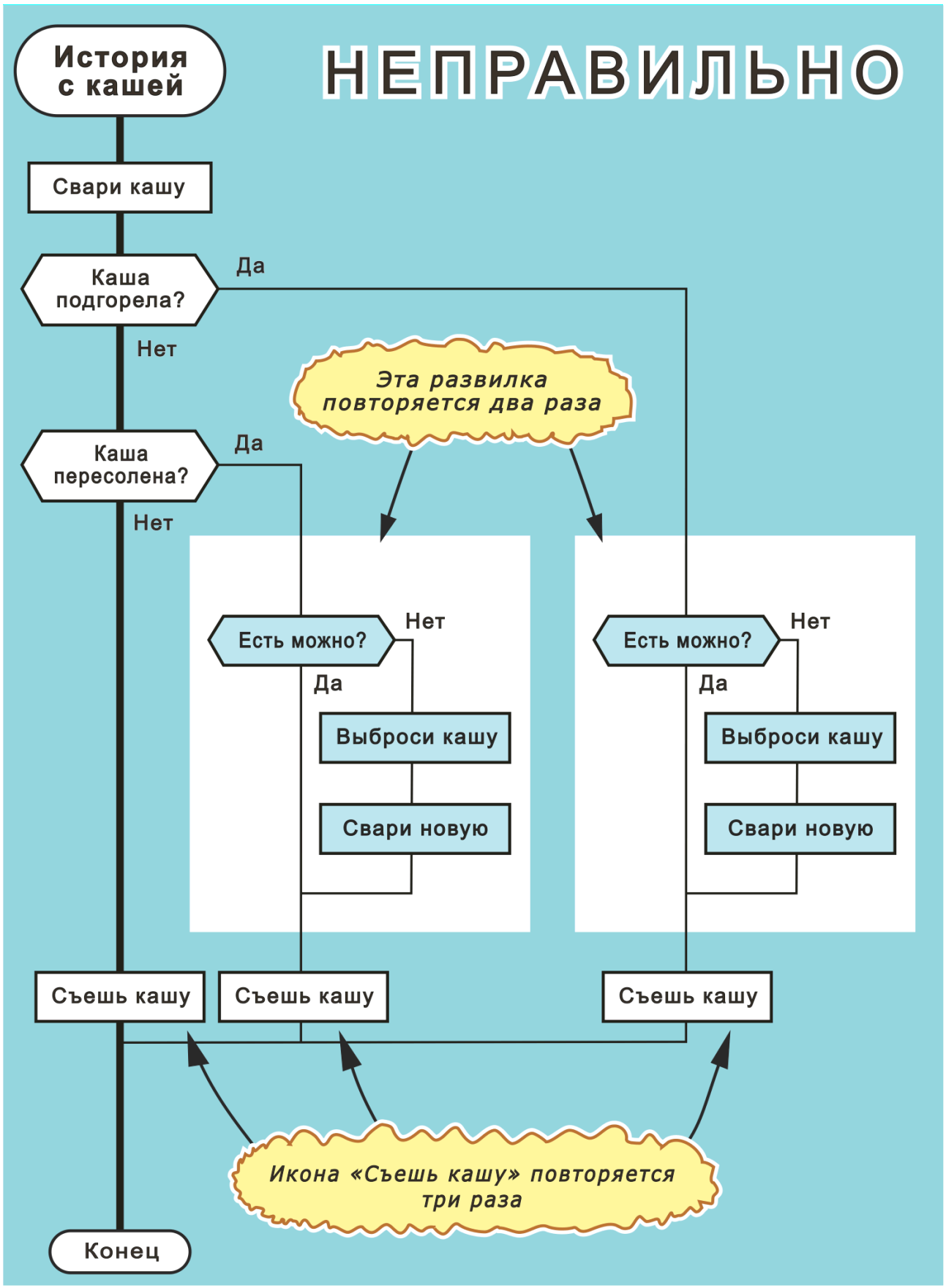


Рис. 27. Очень плохая дракон-схема. Правило «Повторы запрещены» нарушено в двух местах

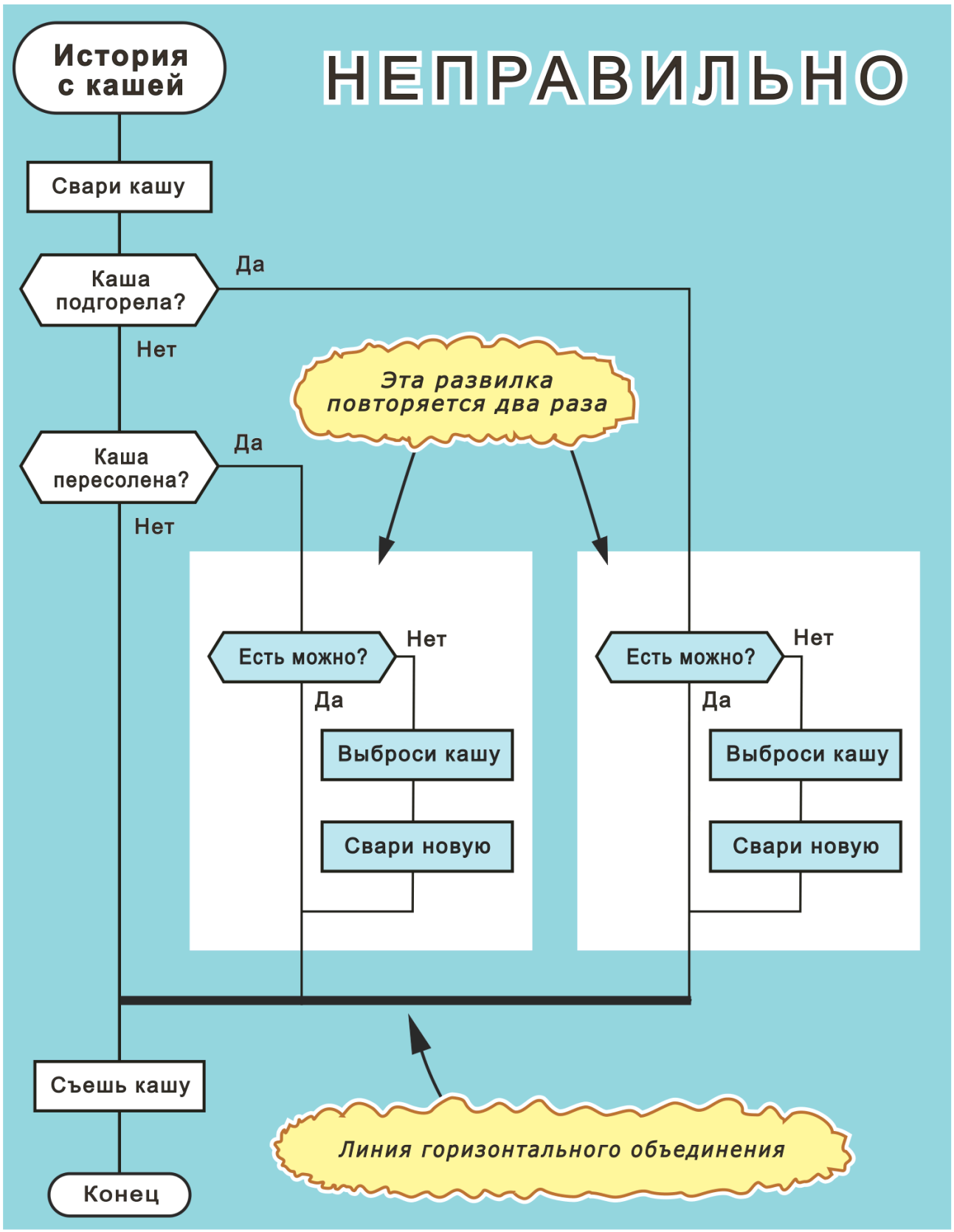


Рис. 28. Данная схема получена из рис. 27 в результате объединения трех икон «Съешь кашу».
Однако работа еще не закончена. Нужно устранить оставшиеся повторы

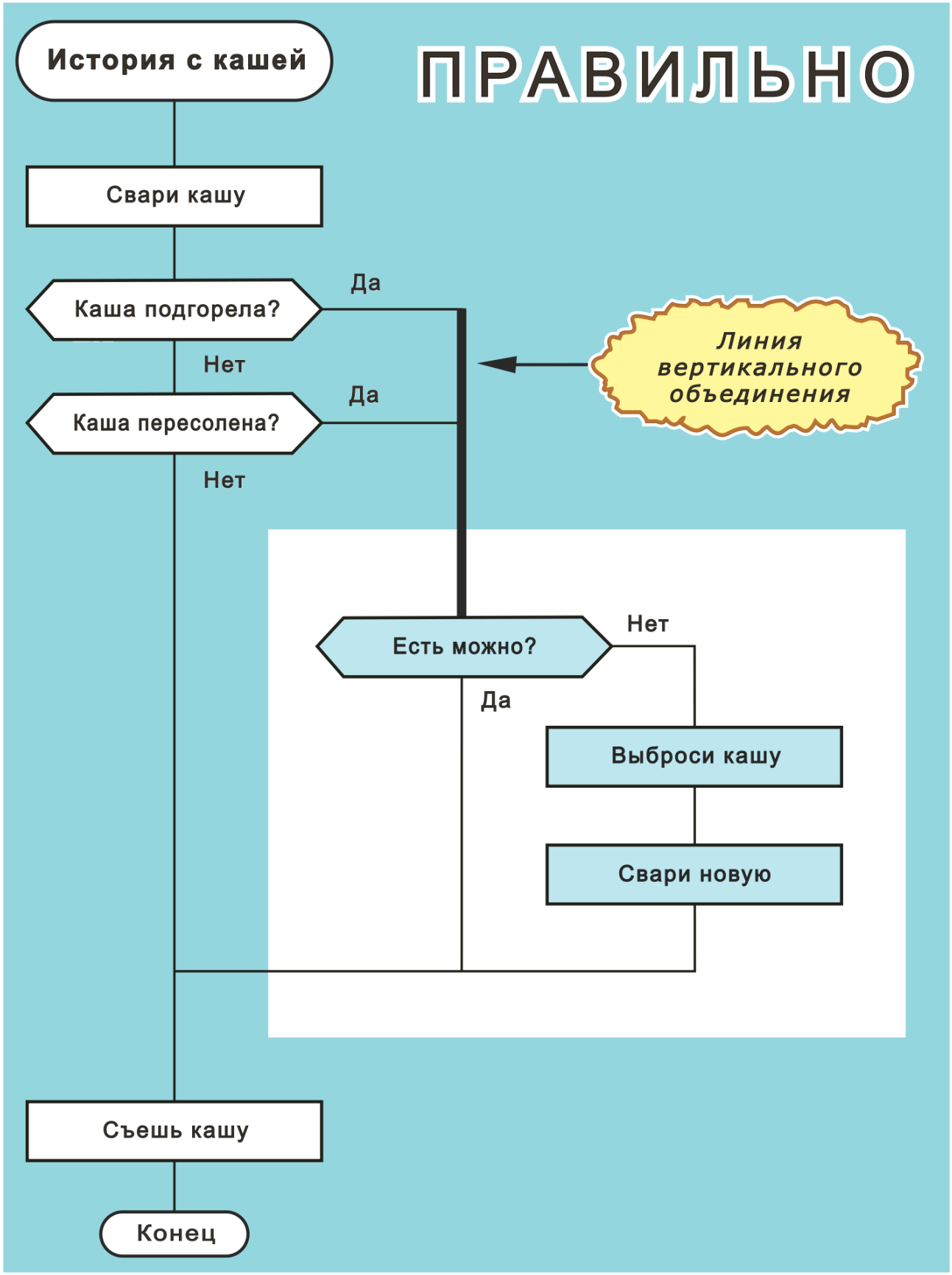


Рис. 29. Хороший (ясный и наглядный) дракон-алгоритм.
Все повторы устранены. Данная схема получена из рис. 28 в результате объединения двух развилок «Есть можно?»

При этой операции несколько блоков объединяются в один. Это делается так:

- выделяются два или более одинаковых блока, выходы которых присоединены к нижней горизонтальной линии;
- входы упомянутых блоков объединяются вертикальной линией,
- удаляются все одинаковые блоки, кроме крайнего левого.

Окончательный результат показан на рис. 29. Легко заметить, что схема на рис. 29 более удобна и занимает меньше места, чем исходная схема на рис. 27. Самое главное, она стала проще и намного понятнее.

ФОРМУЛА ЛИНЕЙНОГО АЛГОРИТМА

Необходимо доказать, что алгоритмы на рис. 27, 28, 29 равносильны. Для этого введем математическое описание маршрутов. Будем считать, что алгоритмы равносильны, если их маршруты попарно совпадают, то есть имеют одинаковое математическое описание.

Маршрут можно описать с помощью формулы, которая представляет собой последовательность букв, обозначающих иконы. Все иконы, включая одинаковые, обозначаются разными буквами.

Линейный (неразветвленный) алгоритм имеет один маршрут и одну формулу. Например, алгоритм на рис. 30 описывается формулой

ABCDEF

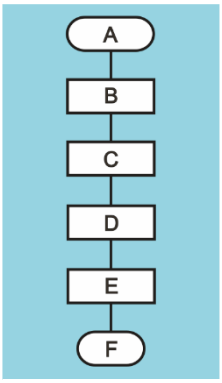


Рис. 30. Линейный алгоритм

ФОРМУЛЫ РАЗВЕТВЛЕННОГО АЛГОРИТМА

Разветвленный алгоритм имеет несколько (два и более) маршрутов. У каждого маршрута своя, отличная от других формула. В формулах разветвленных алгоритмов наряду с буквами, обозначающими иконы, используются слова «да» и «нет», отделяемые пробелами.

Разветвленный алгоритм на рис. 31 имеет два маршрута и описывается двумя формулами, как показано в таблице 1.

Таблица 1

Маршрут	Формула
Маршрут 1	ABC нет DEF
Маршрут 2	ABC да GF

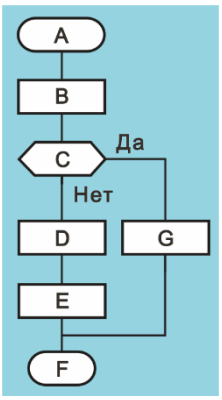


Рис. 31. Разветвленный алгоритм

НАБОР ФОРМУЛ

Набор формул нужен, чтобы описать маршруты разветвленного алгоритма. Набор формул алгоритма на рис. 31 показан в таблице 1. Он состоит из двух формул.

Чтобы написать набор формул данного алгоритма, необходимо:

- выявить все маршруты;
- для каждого маршрута написать формулу.

Совокупность таких формул — это и есть набор формул алгоритма на рис. 31.

РАВНОСИЛЬНЫЕ АЛГОРИТМЫ

Два алгоритмы называются равносильными, если они имеют одинаковый набор формул. Как доказать, что алгоритмы *X и Y* равносильны?

Для этого нужно:

- выявить маршруты алгоритма *X* и для каждого маршрута записать формулу;
- сделать то же самое для алгоритма *Y;*
- убедиться, что два набора формул совпадают.

ДОКАЗАТЕЛЬСТВО РАВНОСИЛЬНОСТИ АЛГОРИТМОВ «ИСТОРИЯ С КАШЕЙ»

На рис. 27, 28, 29 видно, что три изображенных на них алгоритма имеют один и тот же смысл, так как они выполняют одни и те же действия.

Докажем, что эти алгоритмы равносильны. Прежде всего нужно доказать, что горизонтальное объединение есть равносильное преобразование алгоритмов.

Рассмотрим левый алгоритм на рис. 32. Он имеет три маршрута и описывается тремя формулами, как показано в таблице 2.

Таблица 2

Маршрут	Формула
Маршрут 1	A да B да CQ
Маршрут 2	A да B нет CQ
Маршрут 3	A нет CQ

Для нас важно, что таблица 2 (в правом столбце) содержит *набор формул* данного алгоритма.

Затем составим такую же таблицу для другого алгоритма на рис. 32. Получим набор формул для правого алгоритма. Сравним *наборы формул* левого и правого алгоритмов на рис. 32. Легко видеть, что они совпадают.

О чем это говорит? О том, что два алгоритма, представленные на рис. 32 (слева и справа), имеют одинаковые маршруты и одинаковый набор формул. Следовательно, они являются равносильными. Что и требовалось доказать.

Отсюда проистекает важный вывод: операция «горизонтальное объединение» является равносильным преобразованием алгоритмов.

Далее. Точно так же можно доказать, что «вертикальное объединение» есть равносильное преобразование алгоритмов. В самом деле, два алгоритма на рис. 33 имеют одинаковый набор маршрутов, как показано в таблице 3.

Таблица 3

Маршрут	Формула
Маршрут 1	A да B да P да Q
Маршрут 2	A да B да P нет R
Маршрут 3	A да B нет R
Маршрут 4	A нет R

На этом вторая часть доказательства заканчивается. Мы убедились, что преобразования алгоритмов, показанные на рис. 27, 28, 29, являются математически строгими, так как соответствующие наборы формул совпадают между собой.

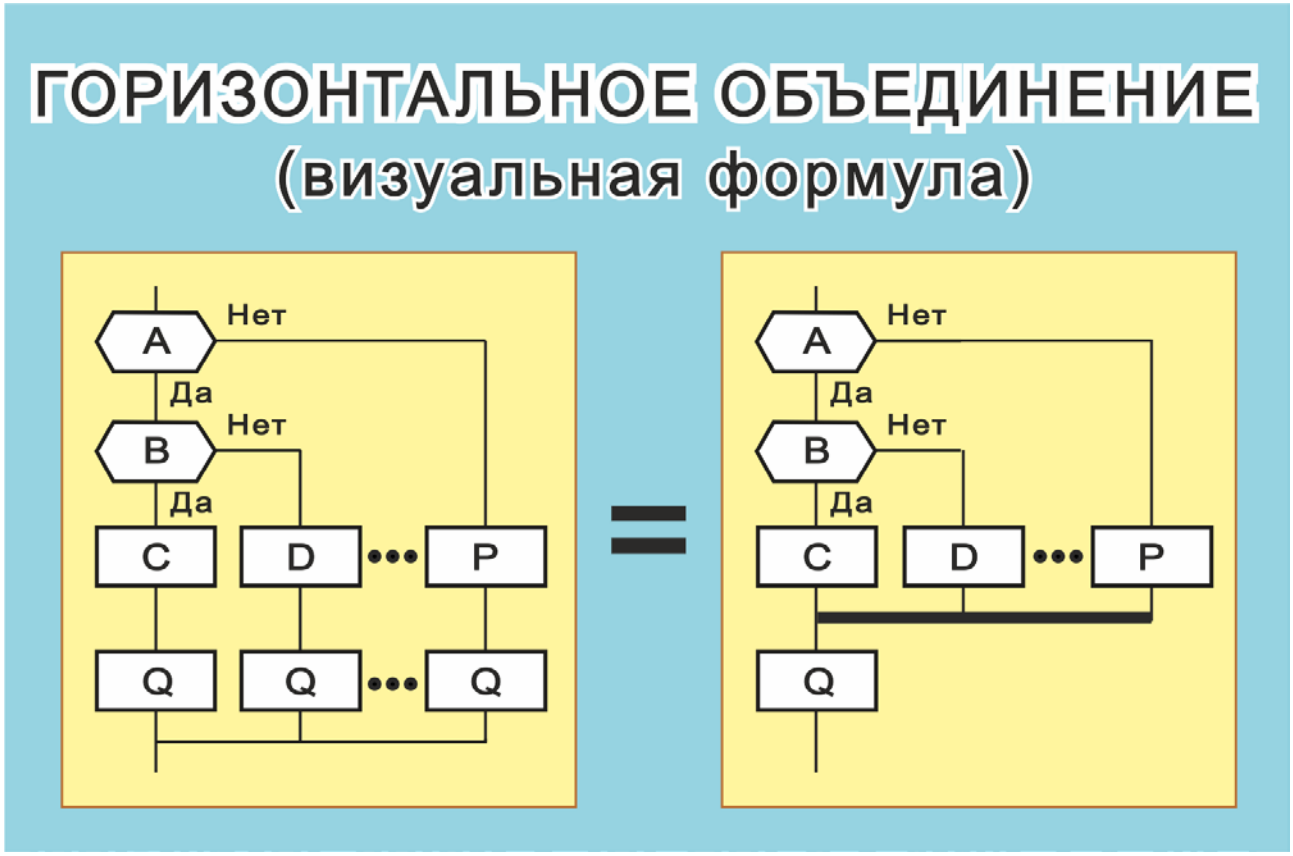


Рис. 32. Преобразование дракон-схемы «Горизонтальное объединение»

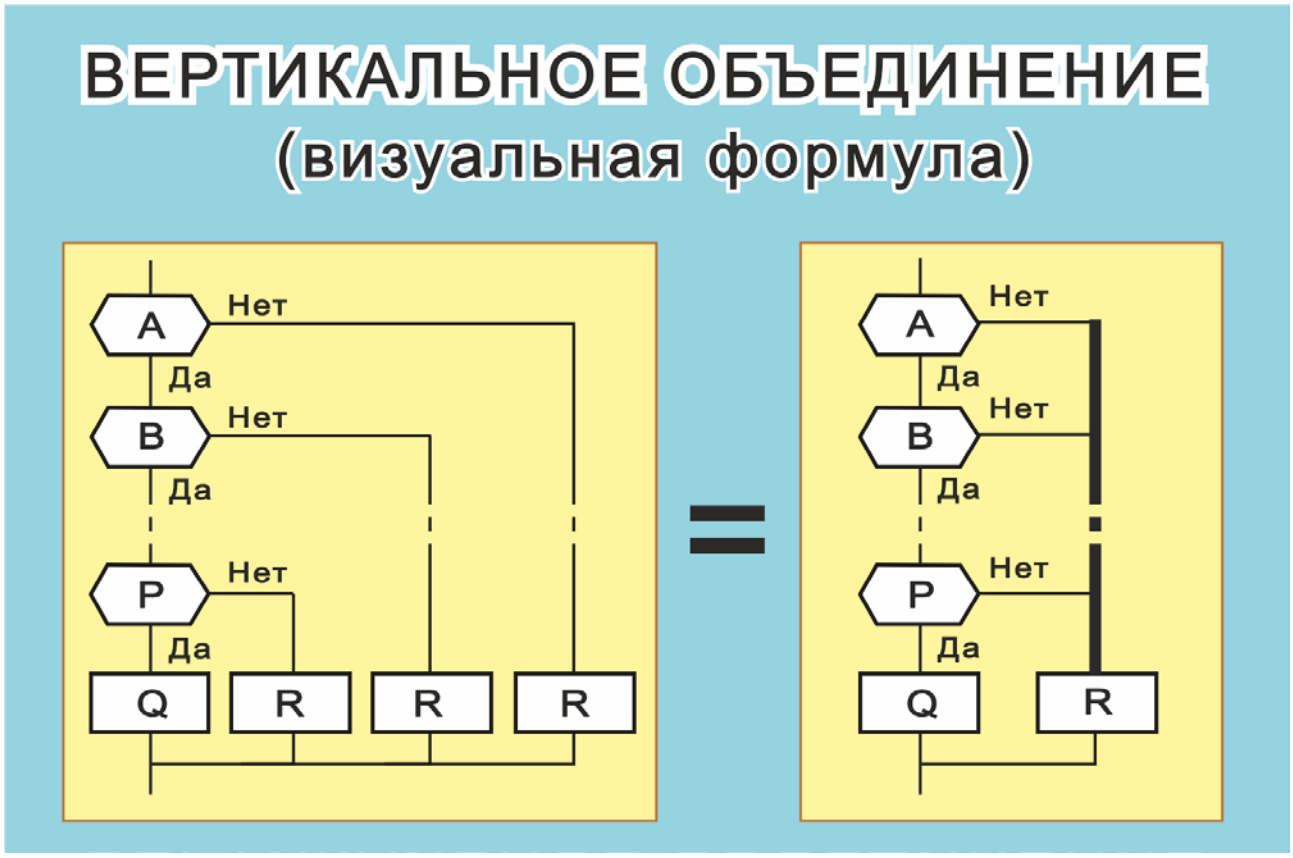


Рис. 33. Преобразование дракон-схемы «Вертикальное объединение»

ЧТО ТАКОЕ ПЛЕЧО

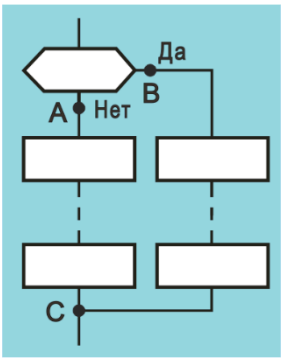


Рис. 34. Левое и правое плечи развилки

Суть дела можно понять из рисунка 34.

Левое плечо — это маршрут от нижнего выхода иконы Вопрос (точка *A*) до точки слияния *C.* Оно содержит ответ «Нет» (или Да), иконы (если они есть) и соединительные линии.

Правое плечо — такой же маршрут, идущий от правого выхода (точка *B*) до точки слияния *C.* Оно содержит ответ «Да» (или Нет), иконы (если они есть) и линии.

Левое плечо идет вертикально вниз, а правое образует ломаную линию.

РОКИРОВКА

Рокировка — это преобразование алгоритма, при котором левое и правое плечи развилки меняются местами. При этом слова «Да» и «Нет» также меняются местами.

Пример рокировки показан на рис. 35.

Что такое рокировка?
Это плеч перестановка!

РОКИРОВКА МОЖЕТ УЛУЧШИТЬ ЭРГОНОМИЧНОСТЬ АЛГОРИТМОВ

Рокировка обладает важным свойством. Она позволяет превратить плохую дракон-схему в хорошую. И улучшить понимаемость алгоритма.

На рис. 35 показана неудачная схема, в ней нарушено правило «Чем правее, тем хуже». Почему? Голод не тетка. Если кошка голодная, это плохо; если сытая, хорошо.

Кроме того, на рис. 35 главный маршрут отклонился от шампура. Рокировка устраняет оба недостатка: выпрямляет кривой главный маршрут и переносит голодную кошку на боковой маршрут (рис. 36).

Таким образом, дефектную схему на рис. 35 можно с помощью рокировки улучшить и преобразовать в корректную, эргономичную схему на рис. 36.

Рокировка обладает удивительным свойством. Она может исправить плохое и создать хорошее, переделать брак в образец.

Рокировка — операция, которая видоизменяет внешний облик алгоритма, не меняя его по существу. При рокировке смысл алгоритма не меняется.

Операция «рокировка» относится не ко всему алгоритму, а только к одной развилке (как показано на рис. 35, 36).

ТЕОРЕМА РОКИРОВКИ

Теорема. Если к алгоритму *X,* содержащему развилку, применить операцию «рокировка», получим алгоритм *Y*, равносильный алгоритму *X.*

Доказательство. Рассмотрим алгоритм на рис. 37. Он имеет два маршрута и две формулы, как показано в таблице 4.

Таблица 4

Маршрут	Формула
Маршрут 1	A да B
Маршрут 2	A нет C

Составим такую же таблицу для алгоритма на рис. 38. Сравнив два *набора формул,* убеждаемся, что они совпадают. Следовательно, алгоритмы на рис. 37 и 38 равносильны.

Отсюда вытекает, что операция «рокировка» является равносильным преобразованием алгоритмов.

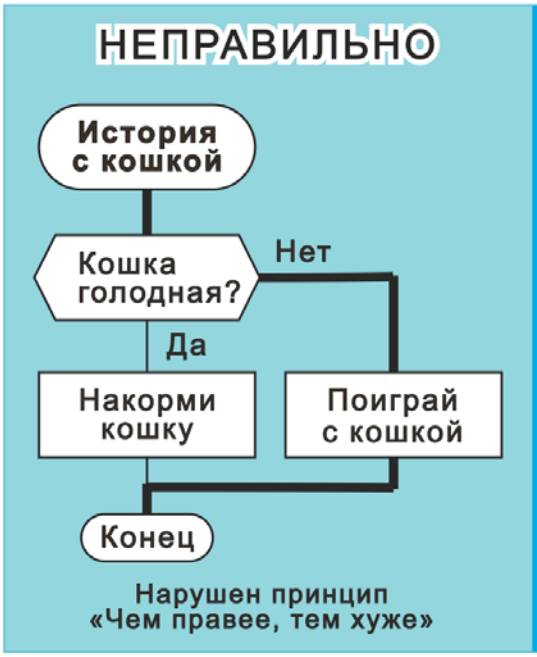


Рис. 35. Плохая схема. Нарушено правило «Главный маршрут должен идти по шампуру»

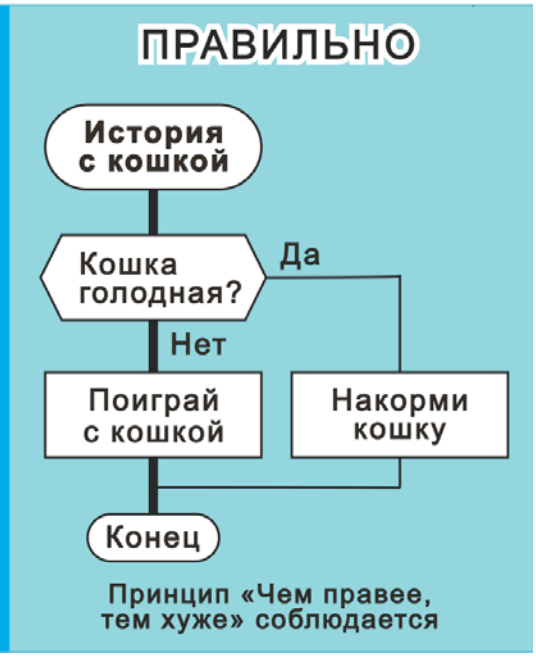


Рис. 36. Рокировка исправила схему на рис. 35, выпрямила главный маршрут и пустила его по шампуру

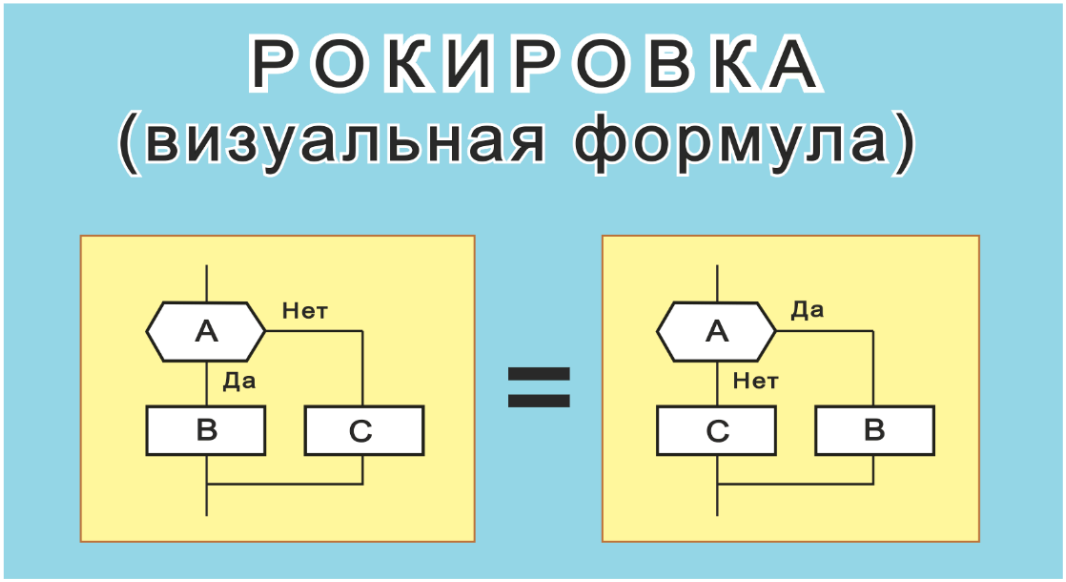


Рис. 37. Маршрут *A да B* расположен слева; маршрут *A нет C* — справа

Рис. 38. Рокировка поменяла маршруты местами.

ИСПОРЧЕННЫЙ ГЛАВНЫЙ МАРШРУТ. КАК ЕГО ИСПРАВИТЬ?

Мы старались показать, что рокировка полезная вещь. Осталось выяснить, как с пользой для дела использовать ее на практике.

Рассмотрим более сложную задачу. Не одну-единственную развилку, а солидный алгоритм «Обед в ресторане», содержащий шесть развилок.

На рис. 39 показана плохая, неэргономичная дракон-схема. Согласно правилу, главный маршрут (жирная линия) должен быть прямым, как стрела, и идти точно по шампуру. А он вместо этого превратился в ломаную линию, которая постоянно делает скачки и сбивает с толку читателя. Чтобы исправить промах, нужно несколько раз сделать рокировку.

Заранее можно предсказать, что для исправления всех дефектов операцию «рокировка» придется выполнить шесть раз — по числу имеющихся в схеме развилок.

Начнем по порядку.

Первый раз делаем рокировку в развилке «В меню есть ваш любимый салат?», второй — в развилке «Есть хоть какой-то салат?». Двойная рокировка позволяет пустить главный маршрут по шампуру на верхнем участке схемы. Однако внизу главный маршрут по-прежнему совершает неоправданные скачки и зигзаги.

Затем делаем рокировку в развилке «Борщ очень вкусный?». И еще раз в развилке «Борщ пересолен?». Тем самым улучшаем среднюю часть дракон-алгоритма.

Нам осталось выпрямить главный маршрут на нижнем участке. Для этого переставляем плечи у двух развилок: «Жаркое как подошва?» и «Другая порция еще хуже?».

В результате шести рокировок намеченный план полностью выполнен. Запутанная схема на рис. 39 превратилась в хорошую (эргономичную) схему на рис. 40.

Подведем итоги. Выпрямляя главный маршрут, мы делаем алгоритм более наглядным и легким для понимания. Главный маршрут — это путеводная нить алгоритма, позволяющая быстрее уяснить суть дела и не заблудиться в путанице развилок.

А теперь – самое интересное. Мы осуществили выпрямление главного маршрута не случайно, не методом проб и случайного успеха, а на основании строгого математического закона — *закона рокировки.* Напомним суть закона: рокировка – равносильное преобразование алгоритма. При рокировке смысл алгоритма не меняется.

Полученный результат чрезвычайно важен. Мы на практике убедились, что рокировка позволяет улучшить наглядность, доходчивость и эргономичность алгоритма.

> Закон рокировки придает нашим эргономическим действиям (позволяющим выпрямить дефектный главный маршрут) математическую строгость и точность

В заключение сделаем замечание. Разработчик дракон-алгоритма должен стараться с самого начала проектирования соблюдать два правила:

- главный маршрут должен идти по шампуру;
- чем правее, тем хуже.

Если это удалось, то разработчику не придется прибегать к рокировке. И наоборот, если выяснится, что правила нарушены, ошибку всегда можно исправить с помощью рокировки.

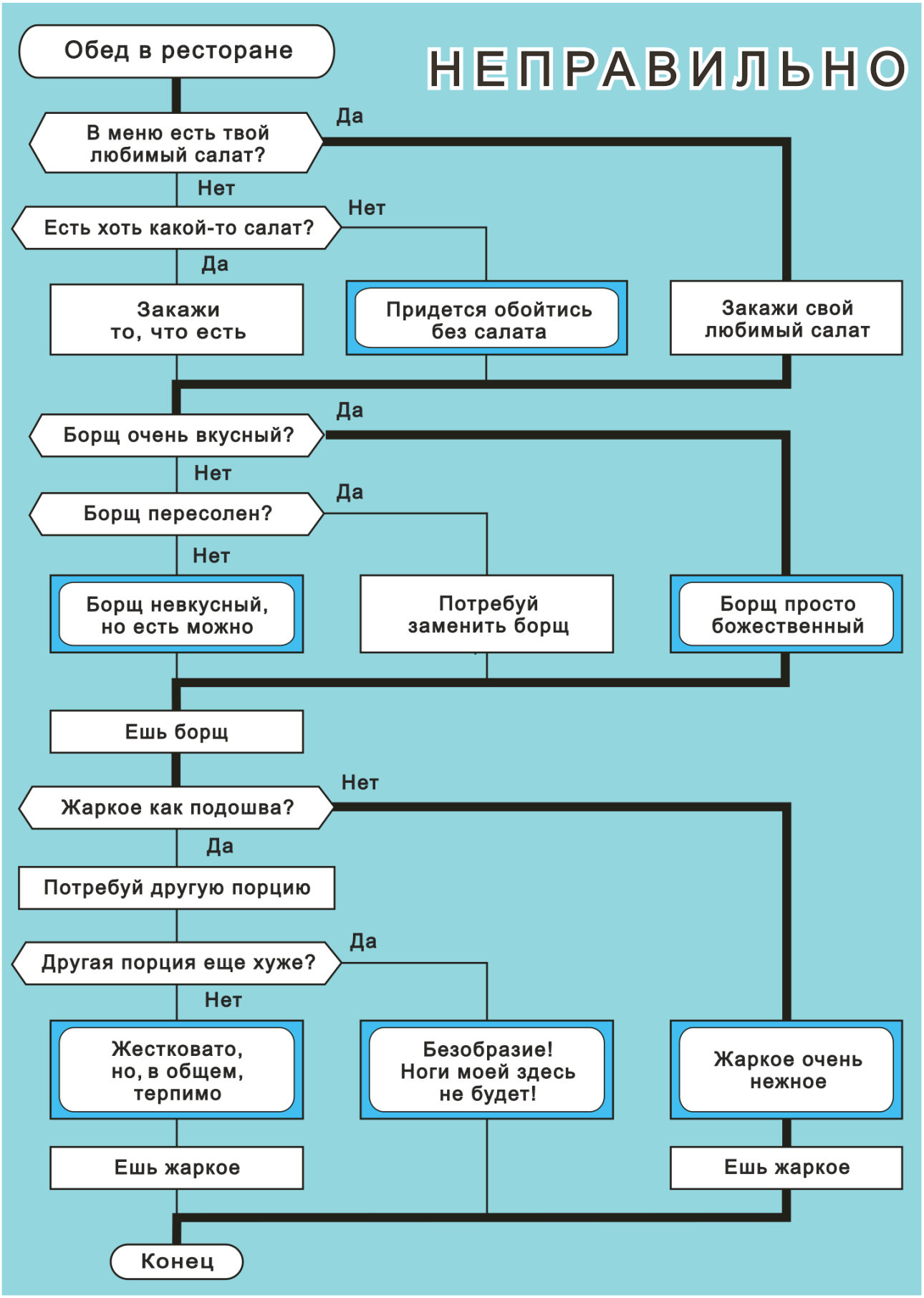


Рис. 39. Плохой дракон-алгоритм. Главный маршрут (жирная линия) все время петляет и делает зигзаги. Его трудно проследить взглядом

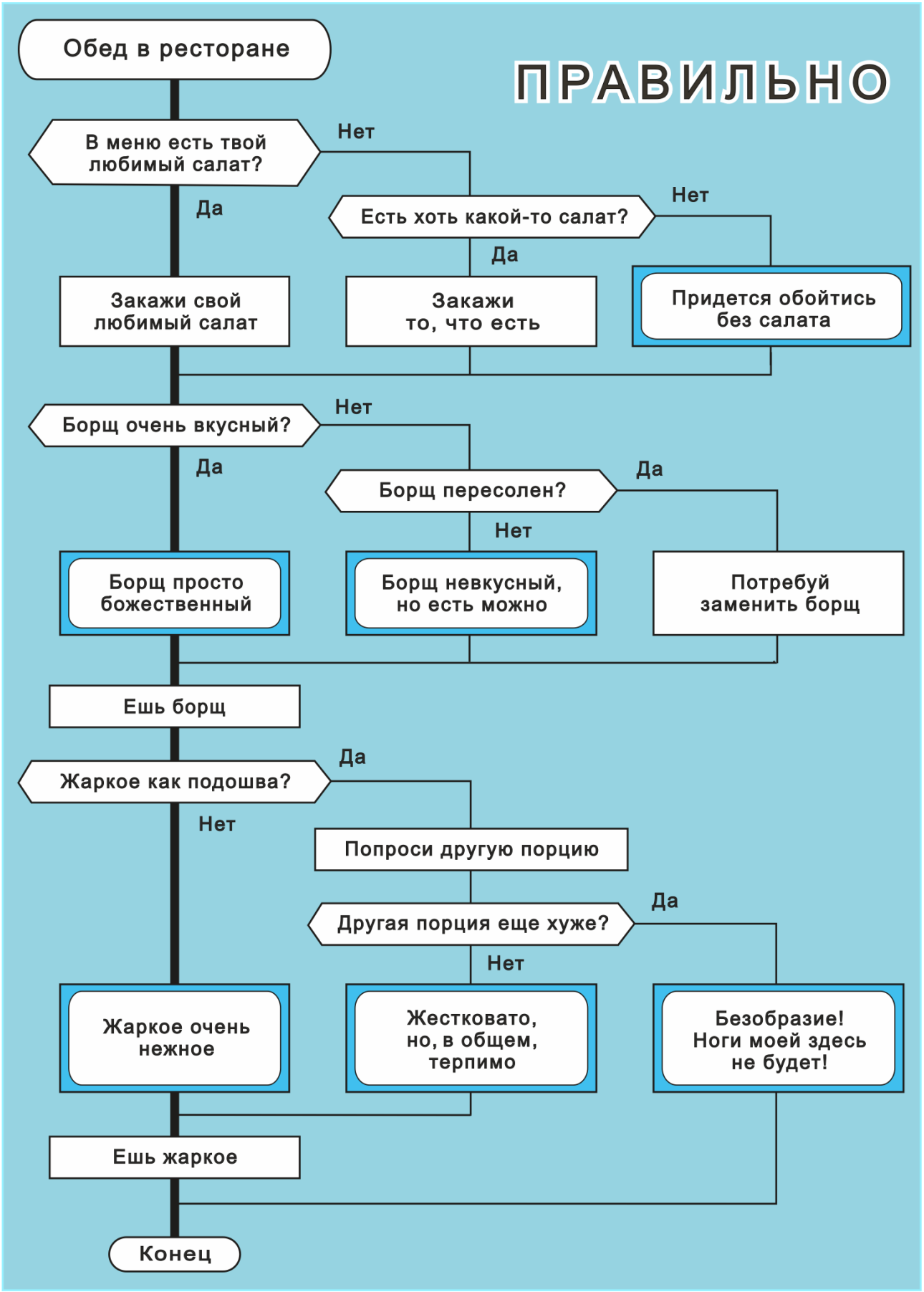


Рис. 40. Хороший дракон-алгоритм. Он получен в результате улучшения схемы на рис. 39. Главный маршрут прямой как стрела. Выполняется правило «Чем правее, тем хуже».

КАКОВА ЦЕЛЬ

Мы познакомились с тремя равносильными преобразованиями алгоритмов:

- горизонтальное объединение,
- вертикальное объединение.
- рокировка.

Зачем нужны эти преобразования? Какую пользу они приносят? К какой цели мы стремимся?

Выше говорилось, что цель — улучшить понимаемость алгоритмов, повысить удобочитаемость, устранить трудности понимания. Но что это значит? От чего зависит понимаемость?

Если говорить простым языком, цель состоит в том, чтобы устранить путаницу и навести порядок. В качестве образца возьмем порядок на географической карте.

КАРТОГРАФИЧЕСКИЙ ПРИНЦИП ПРИМИТИВА

Чем хороша географическая карта? Тем, что она упорядочена. Вверху север, внизу юг. Смотрим наверх — видим север, смотрим вниз — видим юг. Двигаясь по меридиану сверху вниз, мы перемещаемся с севера на юг.

Важнейшая цель дракон-алгоритма — упорядочить алгоритмическую картину и устранить путаницу. Для этого он выстроен по образцу географической карты.

Дракон-алгоритм «примитив» на рис. 40 нарисован не как попало, а по строгим правилам. Вверху начало алгоритма и начало времени. Внизу — конец алгоритма и конец времени.

Картографический принцип говорит о том, что движение взора по карте имеет четкий смысл. Глаза учитывают и отслеживают два направления: север – юг и запад – восток. Смотрим налево — видим запад, смотрим направо — видим восток.

У примитива точно так же. Смотрим налево — видим шампур. Переводим взгляд направо — наблюдаем ситуацию «Чем правее, тем хуже».

КАРТОГРАФИЧЕСКИЙ ПРИНЦИП СИЛУЭТА

Выше мы описали картографический принцип для примитива, т. е. для малого алгоритма. Он полностью сохраняет силу для ветки. Принцип имеет локальный характер, действует внутри одной ветки и не распространяется за ее пределы.

Сделаем следующий шаг и определим порядок взаимодействия веток. Тем самым распространим картографический принцип на весь силуэт.

Для этого нужно упорядочить ветки по горизонтали. Лучше всего расположить их слева направо в той последовательности, в какой они включаются в работу. Для этого служит правило: «Чем правее, тем позже».

Силуэт, нарисованный по правилам, считается хорошим, эргономичным (рис. 5, 14). Схемы, где это правило нарушается, объявляются плохими. Их использование запрещено.

В разрешенных (эргономичных) алгоритмах имеет место следующий порядок действий (рис. 5, 14):

- первой работает крайняя левая ветка, последней — крайняя правая;
- остальные ветки передают эстафету друг другу слева направо (при этом может случиться так, что некоторые ветки будут пропущены).

Отсюда следует, что время в силуэте перемещается не по одной, а по двум осям — и по вертикали, и по горизонтали.

- Двигаясь по ветке сверху вниз, мы перемещается во времени от начального момента до конечного (на данном отрезке). Время движется по ветке вертикально вниз.
- Двигаясь по веткам слева направо, мы тоже перемещаемся во времени. По разным веткам время движется горизонтально слева направо (при отсутствии веточных циклов).

При этом никакой путаницы не возникает, все учтено и хорошо продумано в соответствии с картографическим принципом. По одной ветке время бежит всегда вниз, по разным веткам — всегда направо (кроме циклов).

МОЖНО ЛИ НАВЕСТИ ПОРЯДОК В АЛГОРИТМАХ

В некотором царстве-государстве были два города. Один строился без плана, вкривь и вкось: тесные улочки и переулки сплелись в змеиный клубок (рис. 41). Зато другой был образцовым: красивые площади, широкие проспекты, всюду простор и порядок (рис. 42). В каком городе легче найти дорогу?

Конечно, во втором.

А теперь представьте, что нам нужно изучить незнакомый алгоритм. Если в нем, как в первом городе, нет порядка, он превращается в лабиринт, к котором ничего нельзя понять. Если же он, как второй город, построен по хорошему плану, ситуация в корне меняется. Про такой город говорят: здесь все ясно и понятно.

Чтобы убедиться в этом, давайте посмотрим еще раз на рис. 14 и проведем взглядом по всем вертикалям слева направо. Мы обнаружим не хаос, а строгий порядок. Потому что маршруты нарисованы не случайно, а по строгим правилам.

Рис. 41. Хаотическая планировка и запутанная застройка с кривыми улочками характерна для многих старинных городов, которые стихийно разрастались на месте древних поселков

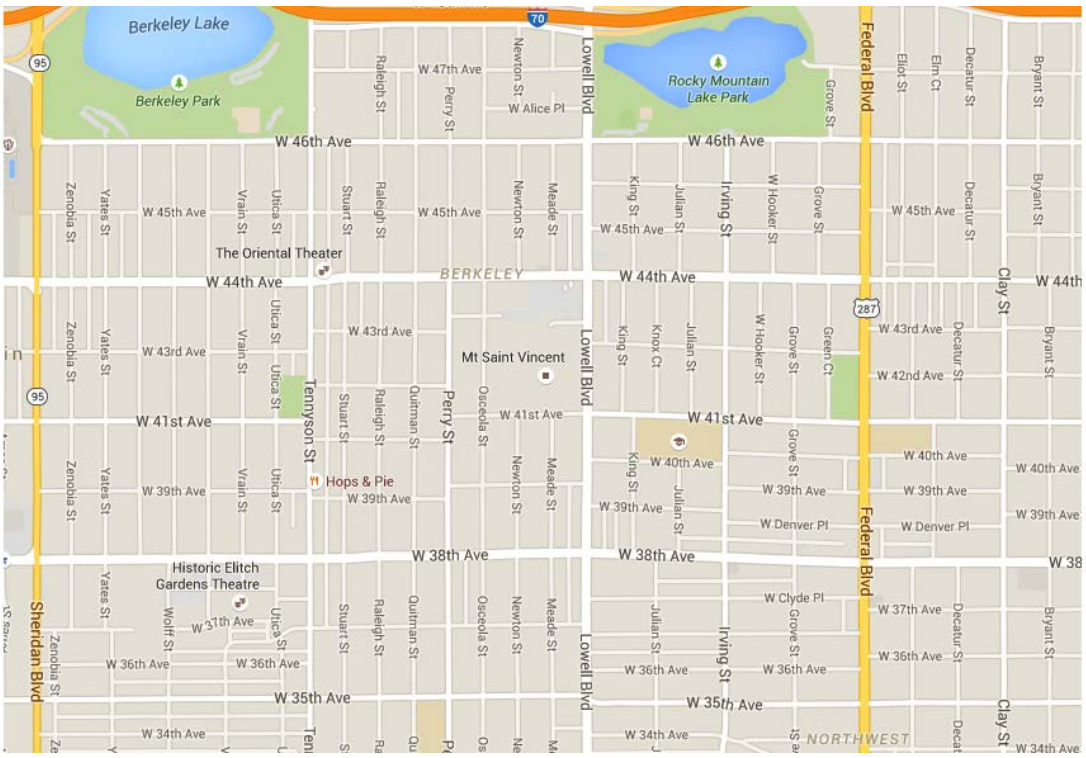


Рис. 42. Современное строительство города позволяет заранее составить четкий план и строго его соблюдать. Никаких нарушений, всюду господствует порядок

ОФИЦИАЛЬНЫЙ ДОКУМЕНТ

В официальном документе «Примерная программа дисциплины „Информатика“» имеются разделы, посвященные языку ДРАКОН [4]. В программе, в частности, говорится:

> «Синтез идей информатики и эргономики полезен тем, что процесс алгоритмизации (который во многих случаях требует значительных трудозатрат) становится менее трудоемким и более ясным. Для этого вводится понятие “эргономичный алгоритм”. Излагаются равносильные преобразования алгоритмов, способные улучшить их эргономические характеристики» [4, p. 15].

МАТЕМАТИЧЕСКИЙ ЧЕРТЕЖ АЛГОРИТМА

Чем дракон-схема отличается от блок-схемы алгоритма по ГОСТ 19.701-90? Тем, что дракон-схема упорядочена и представляет собой математический чертеж алгоритма, построенный по правилам когнитивной эргономики.

Эту мысль мы будем разъяснять на протяжении всей книги. В главах 35-37 доказывается, что «с появлением дракон-схем (drakon-charts) блок-схемы алгоритмов по ГОСТ 19.701-90 полностью потеряли свое значение, так как они во всех отношениях уступают дракон-схемам».

ВЫВОДЫ

1. В языке ДРАКОН предусмотрены три равносильных преобразования алгоритмов:
   - горизонтальное объединение,

- вертикальное объединение.
- рокировка.

2. Действует картографический принцип примитива и силуэта.
3. Картографический принцип примитива действителен для ветки.
4. Картографический принцип ветки имеет локальный характер и не распространяется за ее пределы.
5. Картографический принцип силуэта гласит:
   - в каждой ветке время направлено вертикально вниз;
   - внутри ветки вертикальные маршруты упорядочены слева направо от наиболее благоприятных до наименее благоприятных;
   - в силуэте время движется по горизонтали слева направо;
   - первой работает крайняя левая ветка, последней — крайняя правая.
6. В алгоритмах на языке ДРАКОН маршруты упорядочены и организованы в эргономичную зрительную сцену, предназначенную для облегчения понимания и запоминания алгоритмов.

Часть 2

ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ

Глава 7

ПРОСТЫЕ ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ

КАКИЕ БЫВАЮТ АЛГОРИТМЫ

Различают три типа алгоритмов: линейные, разветвленные и циклические.

Линейные алгоритмы — алгоритмы, в которых нет ни разветвлений, ни повторяющихся (циклических) участков. Примеры показаны на рис. 1, 2, 30.

Разветвленные алгоритмы — те, в которых есть разветвления, но нет циклических участков. Примеры см. на рис. 23-29, 31, 35, 36, 39, 40.

Циклические алгоритмы (сокращенно *циклы)* — алгоритмы, в которых есть повторяющиеся участки. Нам уже встречались такие алгоритмы на рис. 5, 14, 18. Пришла пора рассмотреть тему более детально.

ОБЗОР ЦИКЛОВ ЯЗЫКА ДРАКОН

В языке ДРАКОН имеется следующий ассортимент циклов:

- цикл со стрелкой *(condition-controlled loop);*
- веточный цикл;
- цикл ДЛЯ *(for loop, foreach loop);*
- цикл ЖДАТЬ.

Первые три цикла рассмотрены в этой и следующей главе, цикл ЖДАТЬ — в главе 19.

ЦИКЛ СО СТРЕЛКОЙ

Термин «Цикл со стрелкой» обозначает три типа циклов:

- цикл ДО *(do while loop),*
- цикл ПОКА *(while loop),*
- гибридный цикл *(loop with test in the middle).*

Отличить их очень легко. У цикла ДО вопрос рисуют внизу, а действие вверху (рис. 43). У цикла ПОКА — все наоборот (рис. 44). Гибридный цикл — это «помесь» цикла ДО и цикла ПОКА (рис. 45).

Взглянем на рис. 21, пункт 4. В этой макроиконе две валентные точки. Если ввести икону «Действие» в верхнюю точку, получим цикл ДО (рис. 43). Если в правую, появится цикл ПОКА (рис. 44). Если заполнить обе точки, образуется гибридный цикл (рис. 45).

Циклический алгоритм	Это алгоритм, в котором команды записываются один раз, а выполняются обычно многократно

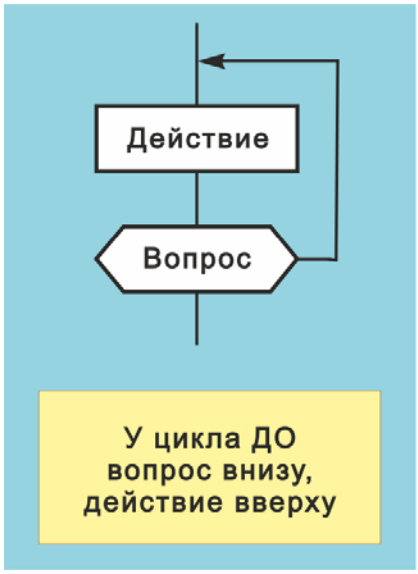


Рис. 43. Цикл ДО

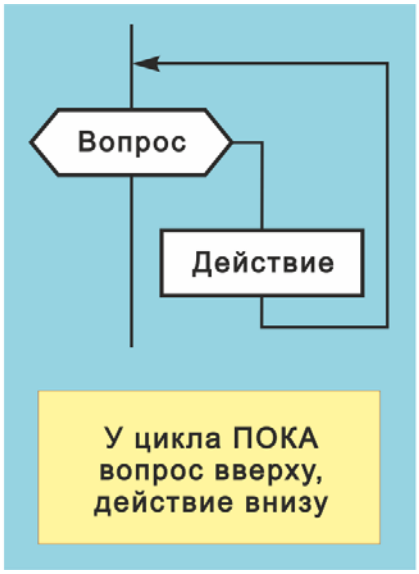


Рис. 44. Цикл ПОКА

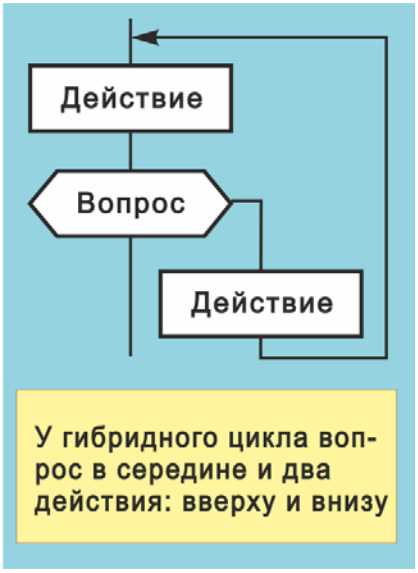


Рис. 45. Гибридный цикл

Вопрос вверху — наверняка
Перед нами цикл ПОКА.
У цикла ДО, наоборот,
Вопрос внизу сидит, как крот.

ЦИКЛ ПОКА (WHILE)

Как говорил великий Блез Паскаль, «предмет математики настолько серьезен, что полезно не упускать случая сделать его немного занимательным» [5].

Однажды Карлсон, который живет на крыше, нашел кошелек и открыл его (рис. 46). А что случилось дальше? Здесь возможны варианты.

Вариант 1. Кошелек оказался пустым. Поэтому Карлсону не удалось купить плюшку.

Вариант 2. В кошельке всего одна монета, так что Карлсон смог купить только одну плюшку.

Вариант 3. В кошельке много монет. Поэтому Карлсон купил гору плюшек и наелся до отвала.

Таким образом, цикл на рис. 46 может:

- ни разу не выполняться;
- выполняться один раз;
- выполняться много раз (два и более).

Цикл начинается с вопроса: «В кошельке есть деньги?».

Если денег нет, из иконы Вопрос бегунок выходит через «Нет», и алгоритм сразу заканчивается. Следовательно, действие «Возьми из кошелька монету» не выполняется ни разу.

Если же деньги есть, бегунок выходит через «Да» и начинает кружить по линии обратной связи. При этом выполняются действия, образующие тело цикла:

- Возьми из кошелька монету
- Купи себе плюшку
- Съешь плюшку

Когда деньги кончатся, бегунок выходит из иконы Вопрос через «Нет». И алгоритм завершается.

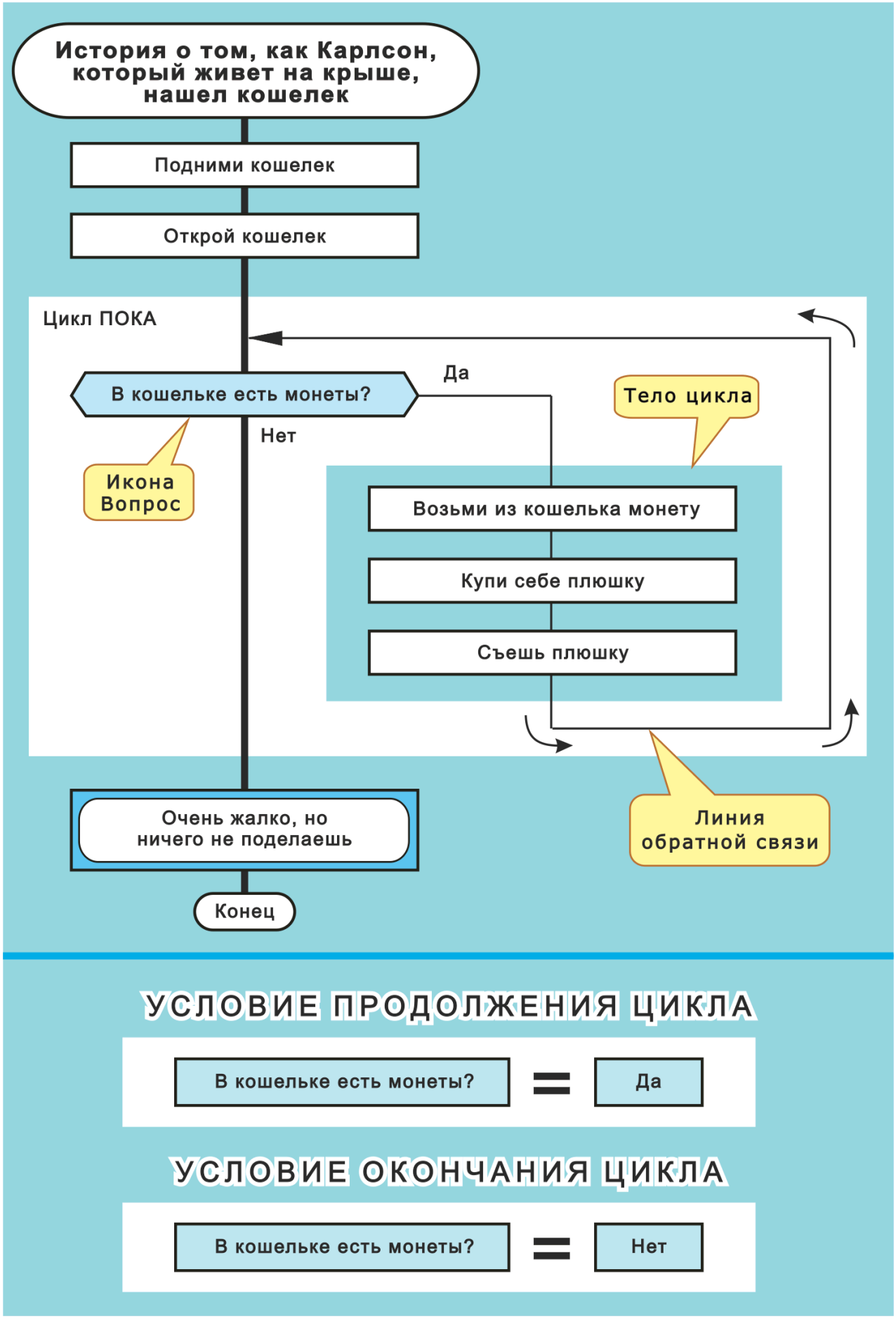


Рис. 46. Пример цикла ПОКА

ОСОБЕННОСТЬ ЦИКЛА «ДО»

В цикле ДО действие выполняется *до* вопроса. Это значит, что бегунок сначала пробегает через одну или несколько икон Действие, потом — через икону Вопрос. Например, в цикле на рис. 47 сначала выполняются два действия:

- Покрась одну доску
- Шагни вправо на ширину доски

И только после этого появляется вопрос: «Все доски покрашены?» При ответе «Нет» бегунок по стрелке возвращается к началу цикла и описанные действия выполняются снова и снова.

Когда все доски будут покрашены, бегунок выходит из иконы Вопрос через «Да». И алгоритм заканчивает работу.

СРАВНЕНИЕ ЦИКЛОВ «ПОКА» И «ДО»

В цикле ПОКА иная картина. Действие (в теле цикла) либо вообще не выполняется, либо выполняется *после* вопроса. Бегунок сначала движется через икону Вопрос, а затем (если ответ благоприятный) — через икону Действие (рис. 46).

Рассмотрим противоположный пример, связанный с циклом ДО (рис. 47). Здесь действия (в теле цикла) выполняются, как минимум, один раз.

> В цикле ДО действия (в теле цикла) обязательно выполняются хотя бы один раз. А в цикле ПОКА при некоторых условиях действие может ни разу не выполняться.

УСЛОВИЕ ПРОДОЛЖЕНИЯ И ОКОНЧАНИЯ ЦИКЛА

Уже говорилось, что в иконе Вопрос записан да-нетный вопрос, т. е. логическая переменная величина, принимающая значение «Да» или «Нет».

На рис. 46 и 47 (внизу) показаны два важных условия.

Условие продолжения цикла	Это условие, при котором цикл продолжает работать
Условие окончания цикла	Это условие, при котором цикл заканчивает работу

Условие продолжения цикла соответствует правому выходу иконы Вопрос. Условие окончания — нижнему.

Условие окончания цикла может помечаться как словом «Нет», так и словом «Да». То же самое относится и к условию продолжения.

Пример 1. На рис. 46 условие продолжения цикла имеет вид

В кошельке есть монеты? = Да

Когда деньги кончатся, логическая переменная величина изменит свое значение с «Да» на «Нет». В этот момент условие продолжения исчезнет. И появится условие окончания цикла:

В кошельке есть монеты? = Нет

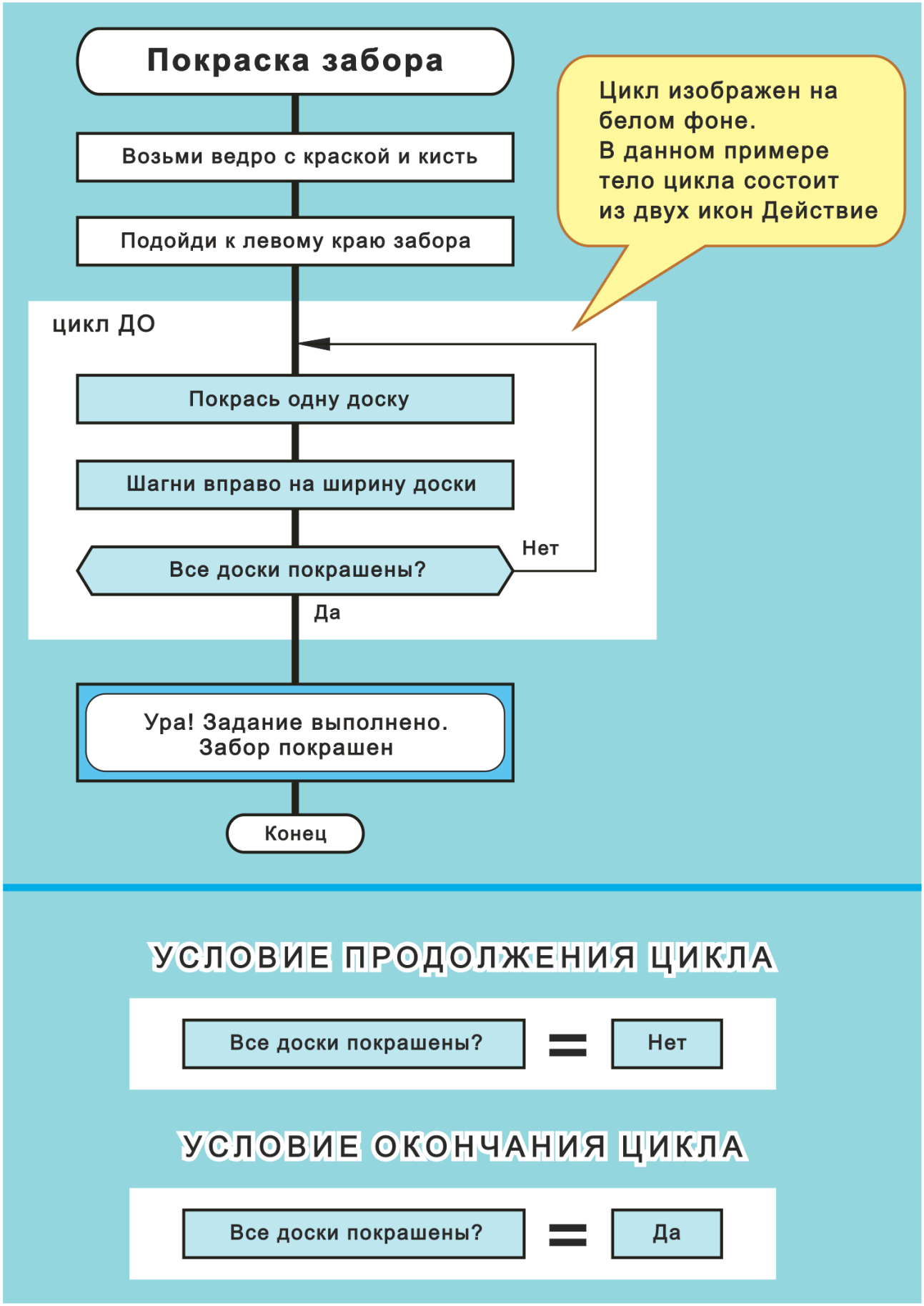
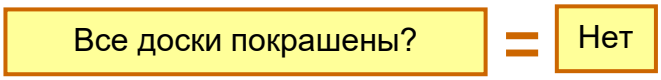
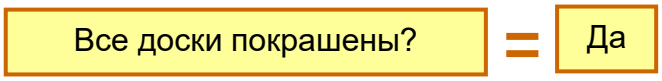


Рис. 47. Пример цикла ДО

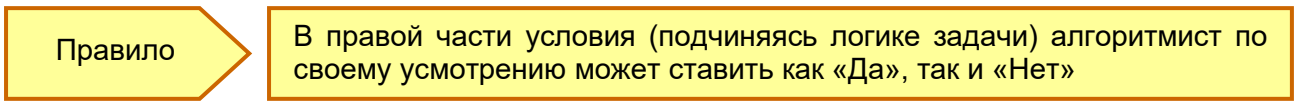
Пример 2. На рис. 47 условие продолжения цикла имеет вид



Когда все доски будут покрашены, логическая переменная изменит свое значение с «Нет» на «Да». В этот момент условие продолжения цикла исчезнет. И появится условие окончания:



Сравнивая примеры 1 и 2 (рис. 46 и 47), можно сформулировать



ЦИКЛ «СТРЕЛКА»

Цикл со стрелкой можно также называть «цикл Стрелка», или просто Стрелка.

ВЫВОДЫ

1. Различают три типа алгоритмов: линейные, разветвленные и циклические.
2. Термин «цикл Стрелка» обозначает три типа циклов:
   - цикл ДО;
   - цикл ПОКА;
   - гибридный цикл.
3. Условие *продолжения* цикла — условие, при котором цикл продолжает работать.
4. Условие *окончания* цикла — при котором цикл заканчивает работу.
5. Условие продолжения цикла соответствует правому выходу иконы Вопрос, условие окончания — нижнему выходу.
6. Условие окончания и продолжения могут помечаться как словом «Нет», так и словом «Да».
7. Существующие циклы, используемые в языках программирования, имеют недостаток. Они накладывают на творческую мысль алгоритмиста ограничения. Графика позволяет снять многие из этих ограничений. В результате алгоритмическая мысль становится более естественной и плодотворной.

Глава 8

ДОСРОЧНЫЙ ВЫХОД ИЗ ЦИКЛА

ВВЕДЕНИЕ

Цикл Стрелка — самый простой вид цикла. Он хорош тем, что позволяет легко и быстро изучить важный прием, который называется *досрочный выход из цикла* (early exit from a loop). Освоив это искусство с помощью цикла Стрелка, в следующей главе мы применим его к более сложным случаям.

ВХОДЫ И ВЫХОДЫ ЦИКЛИЧЕСКОГО АЛГОРИТМА

Цикл имеет только один вход. А выходов может быть несколько. Один выход основной, остальные — досрочные.

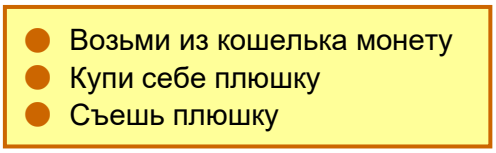
Чтобы выйти из цикла через досрочный выход, нужно выполнить особое условие.

ДОСРОЧНЫЙ ВЫХОД ИЗ ЦИКЛА «ПОКА»

Карлсон, который живет на крыше, может съесть очень много плюшек. Наверное, штук сто. Или даже двести. Но не больше! Иначе он просто лопнет. А теперь предположим, что в кошельке, на его счастье (или беду), оказалось пятьсот монет.

Как в этой ситуации будет работать алгоритм на рис. 46?

Бедный Карлсон! Ему не позавидуешь. Алгоритм заставит его съесть пятьсот плюшек. Все до единой! И он наверняка умрет от обжорства. Почему? Потому что из цикла на рис. 46 нельзя выйти раньше времени. Вспомним — в кошельке пятьсот монет. Значит каждая команда цикла будет исполнена ровно пятьсот раз. Вот три команды, образующие тело цикла:



И лишь затем на вопрос: «В кошельке есть монеты?» — мы получим ответ «Нет» и сможем уйти из цикла.

Отсюда следует вывод. Чтобы спасти Карлсона, нужно организовать *досрочный* выход из цикла. Для этого в алгоритм нужно ввести дополнительную икону Вопрос: «Карлсон уже наелся?» (рис. 48).

Что это даст? Допустим, Карлсон может съесть всего двадцать плюшек. После того как цикл повторится двадцать раз, на вопрос: «Карлсон уже наелся?» — будет получен ответ «Да». И мы благополучно выйдем из цикла.

Мы выяснили, что цикл на рис. 48 имеет не один, а два выхода: *основной* и *досрочный*. Через первый мы выходим, когда в кошельке кончились деньги. Через второй — когда Карлсон наелся. Чтобы появился досрочный выход, мы ввели в алгоритм особое условие (покажите его).

ДОСРОЧНЫЙ ВЫХОД ИЗ ЦИКЛА «ДО»

Папино слово — закон! А папа сказал: сегодня нужно покрасить забор. На рис. 47 показан случай, когда все идет по плану и работа успешно доводится до конца.

Однако в жизни вечно случается то одно, то другое. Например, ни с того ни с сего кончилась краска. Этот случай представлен на рис. 49. Алгоритм на рис. 49 имеет два выхода из цикла:

- *основной* (забор удалось покрасить)
- и *досрочный* (забор остался недокрашенным).

Однако, что значит «кончилась краска»? Одно дело, если краски нет в ведре — тогда ее можно взять в сарае. И совсем другое, если в сарае краски тоже нет. Последний случай показан на рис. 50 и 51.

ДОСРОЧНЫЙ ВЫХОД ИЗ ГИБРИДНОГО ЦИКЛА

Давайте вспомним про гибридный цикл, о котором шла речь на рис. 45. Легко видеть, что на рис. 50 и 51 представлен гибридный цикл.

Почему гибридный? Какой признак об этом говорит? Смотрите, на нижней линии обратной связи появились две иконы Действие:

- Сходи в сарай за краской
- Возьми каску в сарае

Это и есть искомый признак.

СКОЛЬКО СТРЕЛОК: ОДНА ИЛИ ДВЕ

Два рисунка 50 и 51 изображают один и тот же алгоритм. Все иконы в них совпадают. Но есть маленькое отличие. На рис. 50 изображены две стрелки, а на рис. 51 всего одна.

Почему так получилось? Проследим два маршрута на рисунке 50.

Первый маршрут выходит из иконы Вопрос «Краска в ведре кончилась?», затем идет вправо через «Нет» и превращается в нижнюю стрелку. Второй маршрут исходит из иконы «Возьми краску в сарае» и образует верхнюю стрелку.

На рисунке 51 оба маршрута начинаются точно так же, в тех же местах. Но. После этого они соединяются и превращаются в одну-единственную стрелку. О чем это говорит?

Оба алгоритма на рис. 50 и 51 работают совершенно одинаково. Число стрелок (одна или две) никак не влияет на работу алгоритма.

Какой вариант следует предпочесть? Цикл на рис. 50 содержит больше деталей: он имеет две лишние линии, лишний изгиб и лишнюю стрелку, которые загромождают зрительное поле и являются ненужными визуальными раздражителями. Поэтому рекомендуется вариант с одной стрелкой, как на рис. 51, ибо он более экономичный и более элегантный.

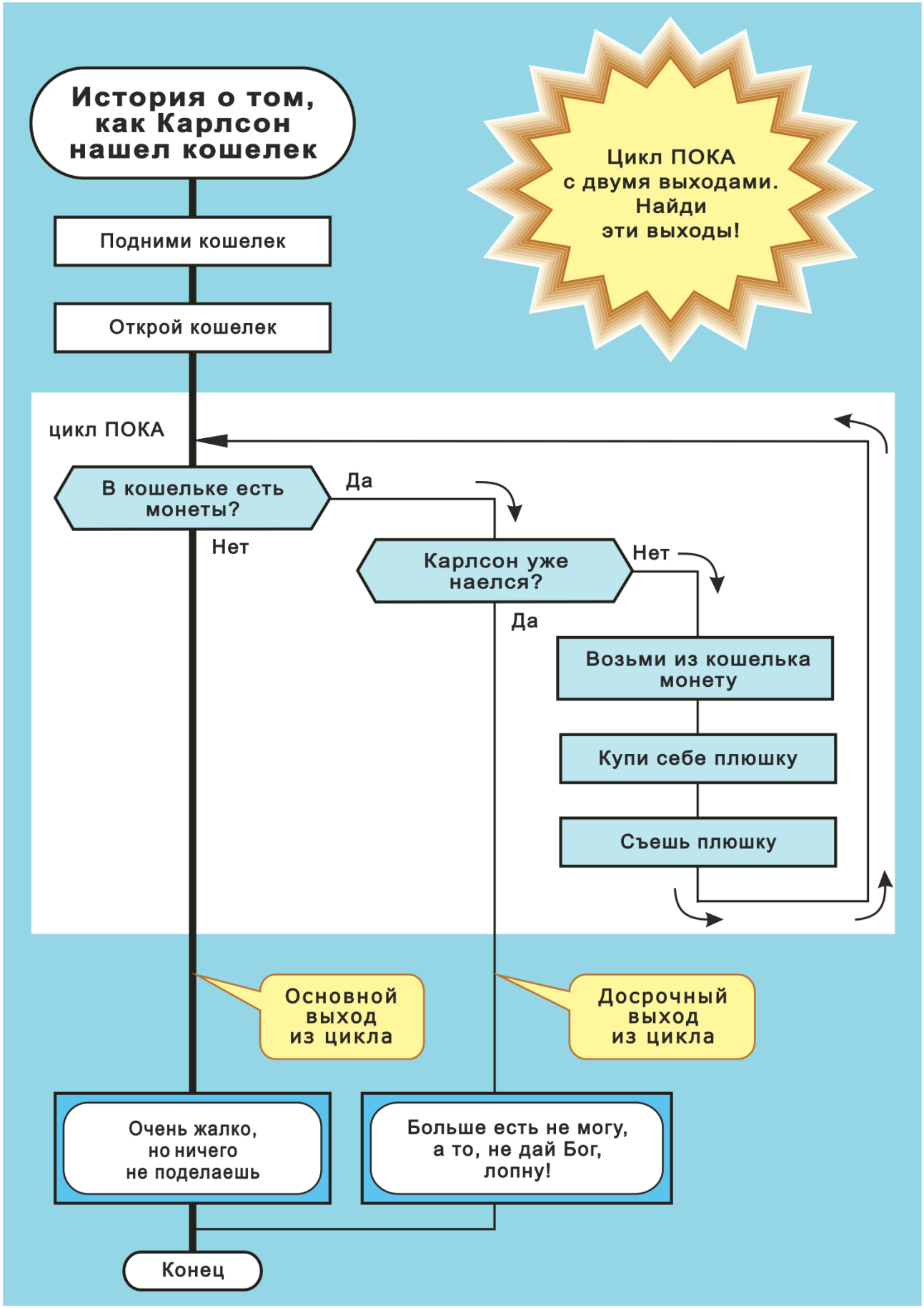


Рис. 48. Досрочный выход из цикла происходит потому, что Карлсон больше не хочет есть. Сравни с рис. 46

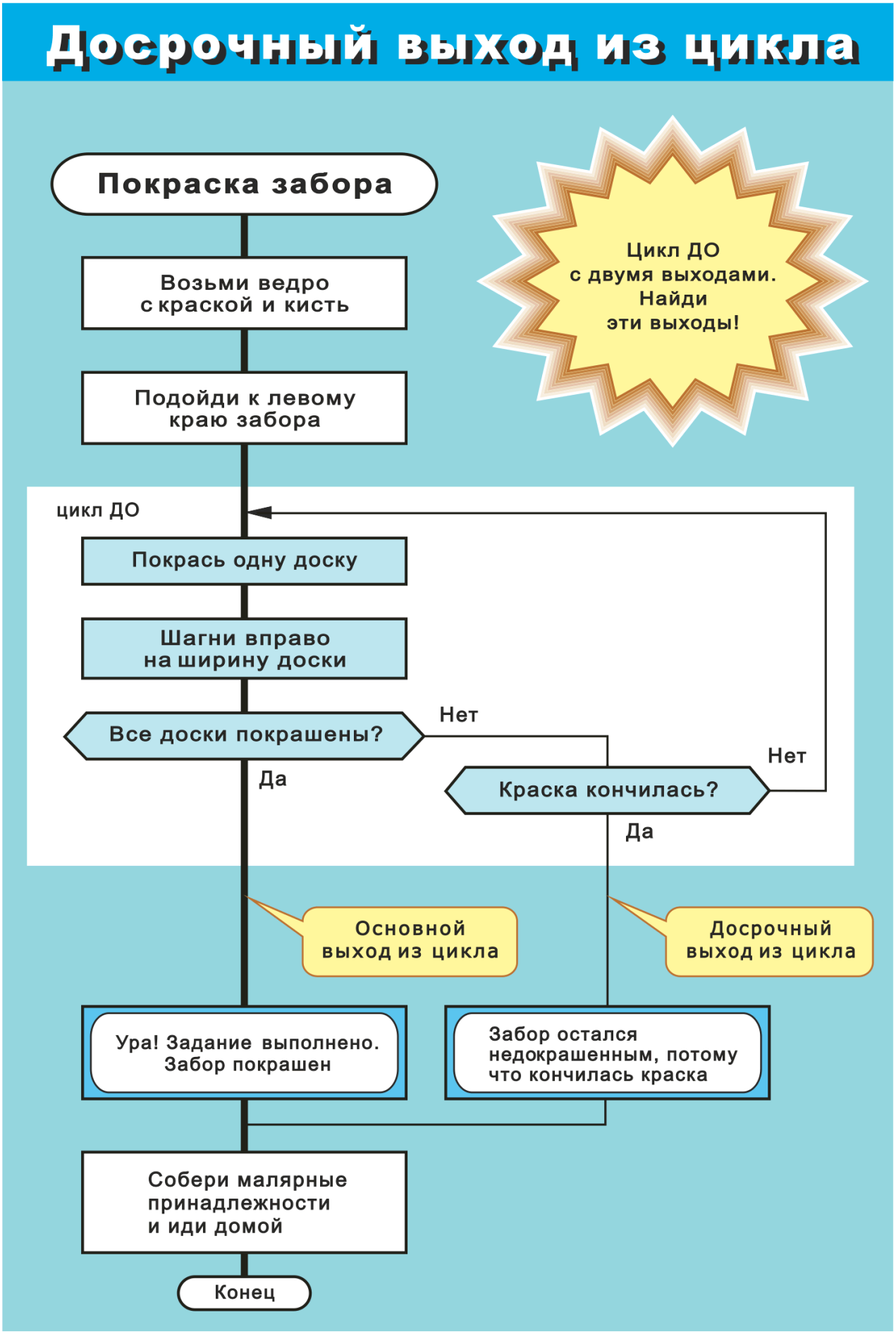


Рис. 49. Досрочный выход из цикла, потому что кончилась краска. Сравни с рис. 47

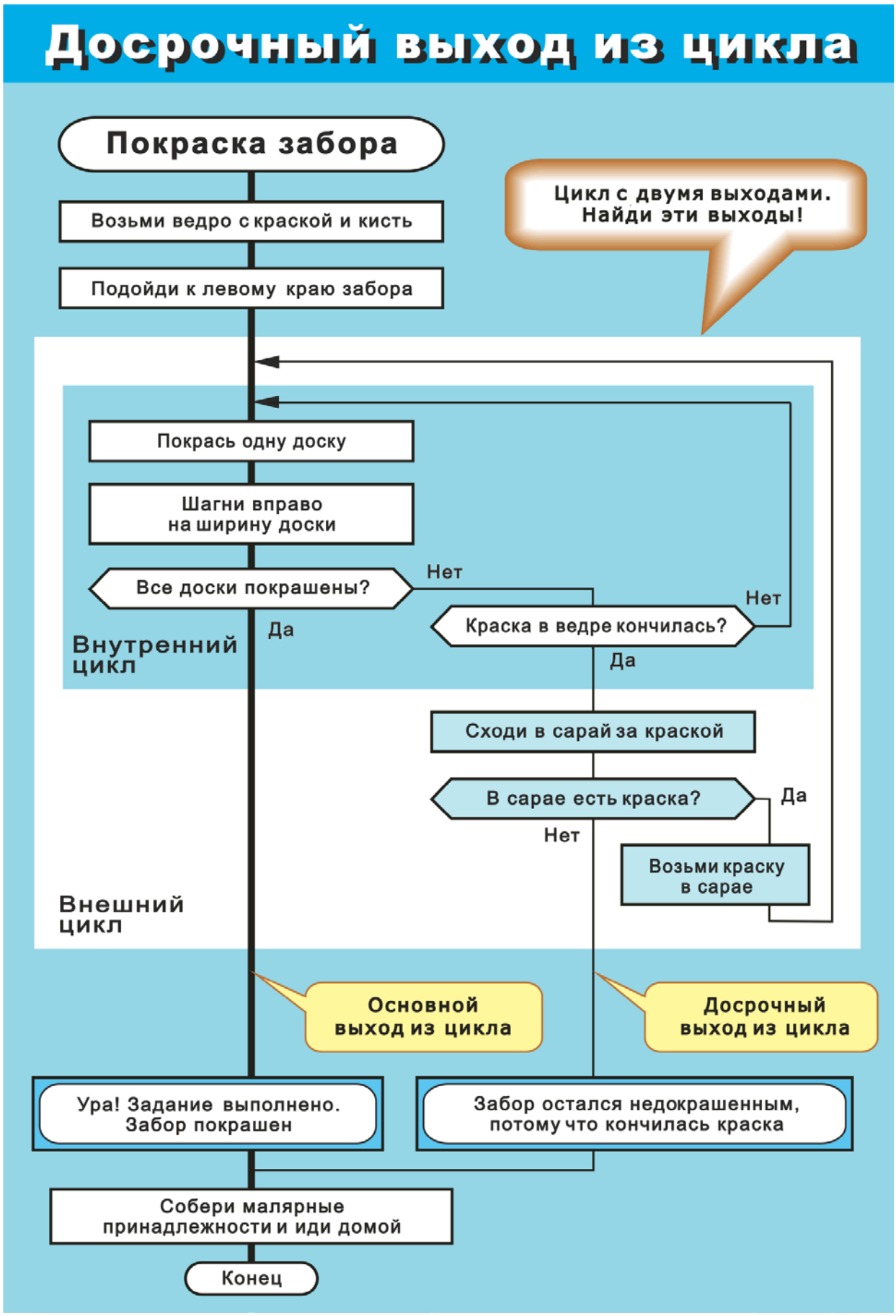


Рис. 50. Досрочный выход из цикла, потому что краска в ведре кончилась. И в сарае краски тоже нет

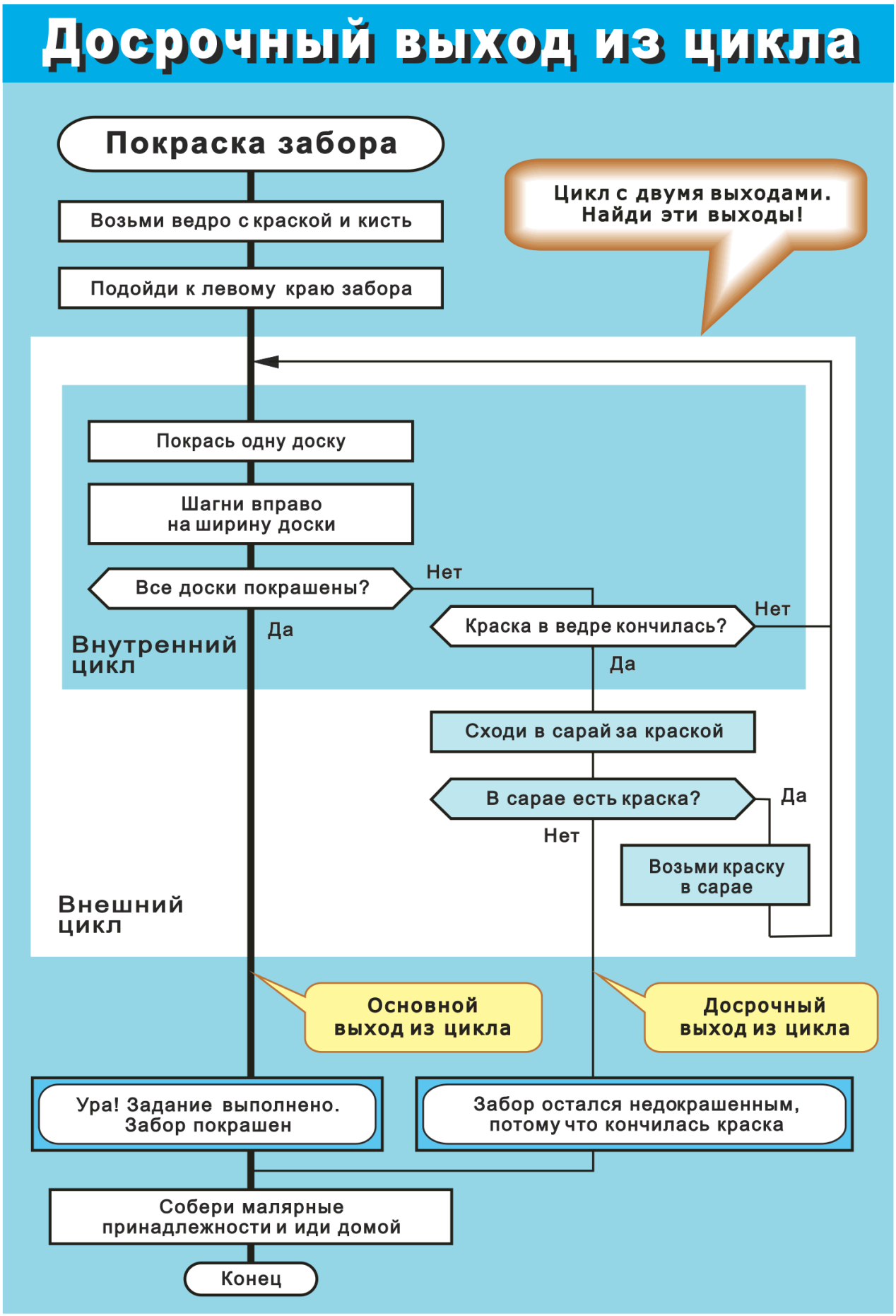


Рис. 51. Те же условия, что на рис. 50. Разница лишь в том, что две стрелки цикла объединились и превратились в одну.

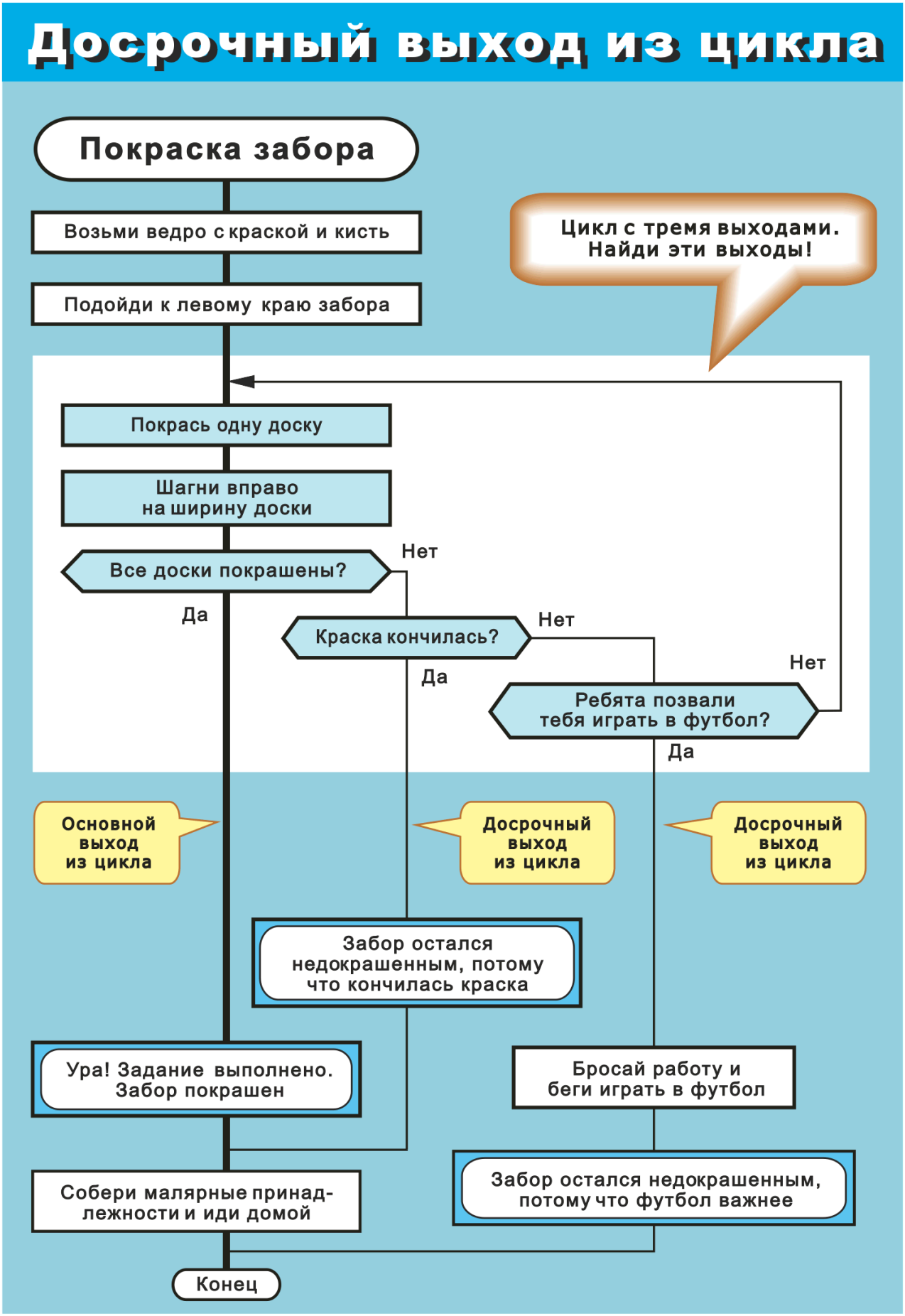


Рис. 52. Два досрочных выхода: (1) потому, что краска кончилась, (2) потому что ребята позвали играть в футбол

Правило одной стрелки

Если две или три стрелки цикла сходятся в одном месте, нужно объединить их и превратить в одну стрелку

ДВА ДОСРОЧНЫХ ВЫХОДА ИЗ ЦИКЛА

Жизнь полна неожиданностей. Кому охота красить забор, если все нормальные люди уже играют в футбол?

Этот полезный для футбола и вредный для забора случай отражен на рис. 52. Данный алгоритм интересен тем, что в нем три выхода из цикла: основной и два досрочных.

В первом случае забор будет покрашен как надо. Во втором дело не ладится из-за нехватки краски. В третьем — из-за любви к футболу.

ВЫВОДЫ

Анализируя циклы со стрелкой на рис. 43—52, можно сделать следующие замечания.

1. Цикл имеет один вход и один или несколько выходов.
2. Если вход в цикл находится на шампуре, то единственный выход также находится на шампуре.
3. Если цикл имеет более одного выхода, основной выход размещается на шампуре. Досрочные выходы — справа от него.
4. Тело цикла ДО находится на шампуре
5. Тело цикла ПОКА находится справа от шампура.
6. Петля цикла находится правее шампура и закручена против часовой стрелки.
7. Закручивать петлю цикла в обратную сторону запрещено.
8. В иконе Вопрос записана логическая переменная величина, принимающая два значения: Да и Нет.
9. Логическая переменная величина задает условие цикла, которое распадается на две части:
    - условие продолжения цикла,
    - условие окончания цикла.

Глава 9

ПРЕОБРАЗОВАНИЕ ЦИКЛА СО СТРЕЛКОЙ В ВЕТОЧНЫЙ ЦИКЛ

ОСОБЕННОСТИ ВЕТОЧНОГО ЦИКЛА

Цикл Стрелка, описанный выше, может использоваться как в примитиве, так и в силуэте. Веточный цикл встречается только в силуэте.

Мы познакомились с веточным циклом в главе 2. В этой главе продолжим изучение и ответим на два вопроса:

- Как превратить цикл со стрелкой в веточный цикл?
- Как превратить примитив в силуэт?

Оказывается, оба вопроса тесно связаны.

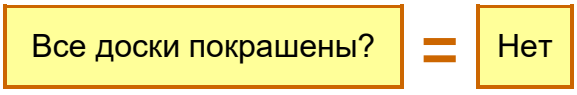
На рис. 47 изображен алгоритм примитив «Покраска забора». Можно ли нарисовать его в виде силуэта? Да, можно. Результат показан на рис. 53.

Силуэт (в отличие от примитива) имеет внутреннюю структуру — он аккуратно разбит на три ветки:

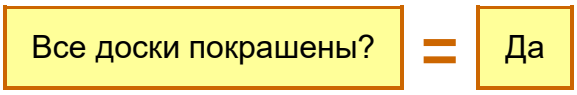
- Подготовка к работе
- Покраска
- Завершение

Нас, разумеется, интересует вторая ветка, где слово «Покраска» встречается дважды: вверху и внизу. Это значит, что перед нами *веточный* цикл. Бегунок, доехав до Адреса «Покраска», тут же вернется к началу ветки и будет «утюжить» ее вновь и вновь.

Циклическое движение по ветке «Покраска» будет продолжаться до тех пор, пока выполняется условие продолжения цикла:



Когда Том Сойер кончит красить забор, появится условие окончания цикла:



После этого бегунок, проходя через икону Вопрос, повернет направо и через Адрес «Завершение» попадет в ветку «Завершение». На этом алгоритм закончит работу (рис. 53).

Можно показать, что алгоритмы на рисунках 47 и 53 эквивалентны — они делают одно и то же (выполняют одни и те же действия).

СИЛУЭТ С ВЕТОЧНЫМ ЦИКЛОМ

Взглянем на рис. 46, где показан алгоритм примитив «Карлсон нашел кошелек». Преобразуем его в силуэт на рис. 54.

Как и раньше, силуэт демонстрирует четкую структуру — он разделен по смыслу на три ветки:

- Находка
- Покупка плюшек
- Завершение

Во второй ветке в иконе Адрес читаем «Покупка плюшек», такая же надпись повторяется в иконе Имя ветки. Подобное совпадение не случайно — перед нами веточный цикл. Мы снова убеждаемся, что цикл Стрелка на рис. 46 благополучно превратился в веточный цикл на рис. 54.

Движение бегунка по ветке «Покупка плюшек» будет повторяться, пока выполняется условие продолжения веточного цикла:

Когда Карлсон истратит все деньги, появится условие окончания цикла:

Бегунок, следуя через икону Вопрос, выйдет направо через «Нет». Через икону Адрес «Завершение» он попадет в ветку «Завершение» и сразу окажется в иконе Конец (рис. 54).

ВЕТОЧНЫЙ ЦИКЛ С ДОСРОЧНЫМ ВЫХОДОМ

Как и любой цикл, веточный цикл может иметь основной и досрочный выходы. Освежим в памяти алгоритм про Карлсона на рис. 48. Действуя по накатанной колее, превратим примитив в силуэт, а цикл Стрелка — в веточный цикл. Результат представлен на рис. 55.

Оба алгоритма имеют досрочный выход из цикла. На рис. 48 мы имеем досрочный выход из цикла со стрелкой, а на рис. 55 он превратился в досрочный выход из веточного цикла.

Черные треугольники показывают, что веточный цикл находится во второй ветке, которая называется «Покупка плюшек». В той же ветке расположены две иконы Вопрос, организующие выход из цикла (рис. 55).

Для основного выхода служит верхний Вопрос «В кошельке есть монеты?» и ответ «Нет».

Нижний Вопрос «Карлсон уже наелся?» и ответ «Да» реализует досрочный выход.

ВЕТОЧНЫЙ ЦИКЛ С ДВУМЯ ДОСРОЧНЫМИ ВЫХОДАМИ

Усложним задачу и попробуем придумать веточный цикл не с одним, а с двумя досрочными выходами.

В качестве прообраза возьмем уже знакомый нам алгоритм «Покраска забора» для случая, когда краска кончилась не только в ведре, но и в сарае. Эта ситуация описана с помощью цикла Стрелка на рис. 50.

Повторим уже испытанный прием, т. е. превратим примитив в силуэт, а цикл Стрелка — в веточный цикл. Результат показан на рис. 56.

Что нового на этот раз? Что изменилось?

Первое отличие (по сравнению с рис. 53) — силуэт теперь содержит не три, а четыре ветки:

- Подготовка к покраске
- Покраска
- Доставка краски
- Уборка

Кроме того, силуэт на рис. 56 имеет два досрочных выхода из цикла:

- Краска в ведре кончилась? Да.
- В сарае есть краска? Нет.

Появилась новая ветка «Доставка краски», которая обеспечивает второй досрочный выход.

Еще одно отличие — внизу мы замечаем две иконы Адрес, содержащие черные треугольники (рис. 56).

Легко проверить, что силуэт с двумя досрочными выводами работает правильно.

ВЕТОЧНЫЙ ЦИКЛ С ЦИКЛОМ «ДО»

Забудем про досрочные выходы и перейдем к новому сюжету.

Веточный цикл можно использовать в сочетании с циклом ДО. При этом образуется конструкция «цикл в цикле».

Предположим, забор состоит из 300 или 500 досок, которые нужно покрасить. Силуэт на рис. 53 успешно решает эту работу при любом числе досок.

Усложним задачу. Будем считать, что для покраски одной доски надо провести по ней кистью несколько раз. Например, 10 или 20 раз. Наш силуэт пока еще не умеет этого делать. Как быть?

Надо придумать новый алгоритм, изображенный на рис. 57.

Внутри веточного цикла «Покраска» появился новый элемент — цикл ДО, спсособный наносить на доску нужное число мазков. Для этой цели цикл ДО содержит иконы:

- Обмакни кисть в краску
- Сделай мазок кистью по доске
- Покраска доски закончена?

Как только данная доска будет полностью покрашена с помощью цикла ДО, выполняется команда:

- Шагни вправо на ширину доски

Это значит, что в дело вступает веточный цикл, который осуществляет переход к следующей доске.

Подведем итоги. При покраске забора веточный цикл переходит от доски к доске, а цикл ДО — от мазка к мазку. Их совместная работа называется «цикл в цикле», потому что цикл ДО находится внутри веточного цикла.

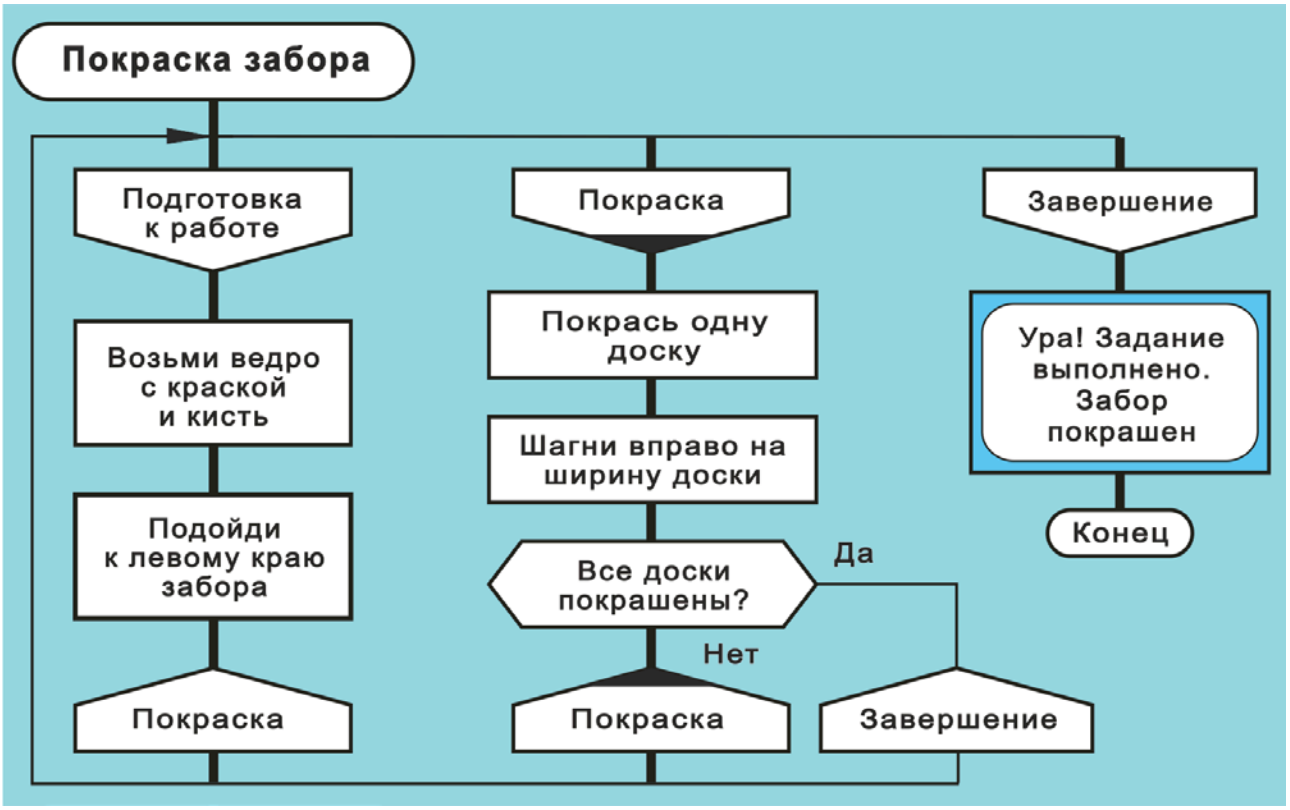


Рис. 53. Алгоритм силуэт «Покраска забора» с веточным циклом. Сравни с рис. 47

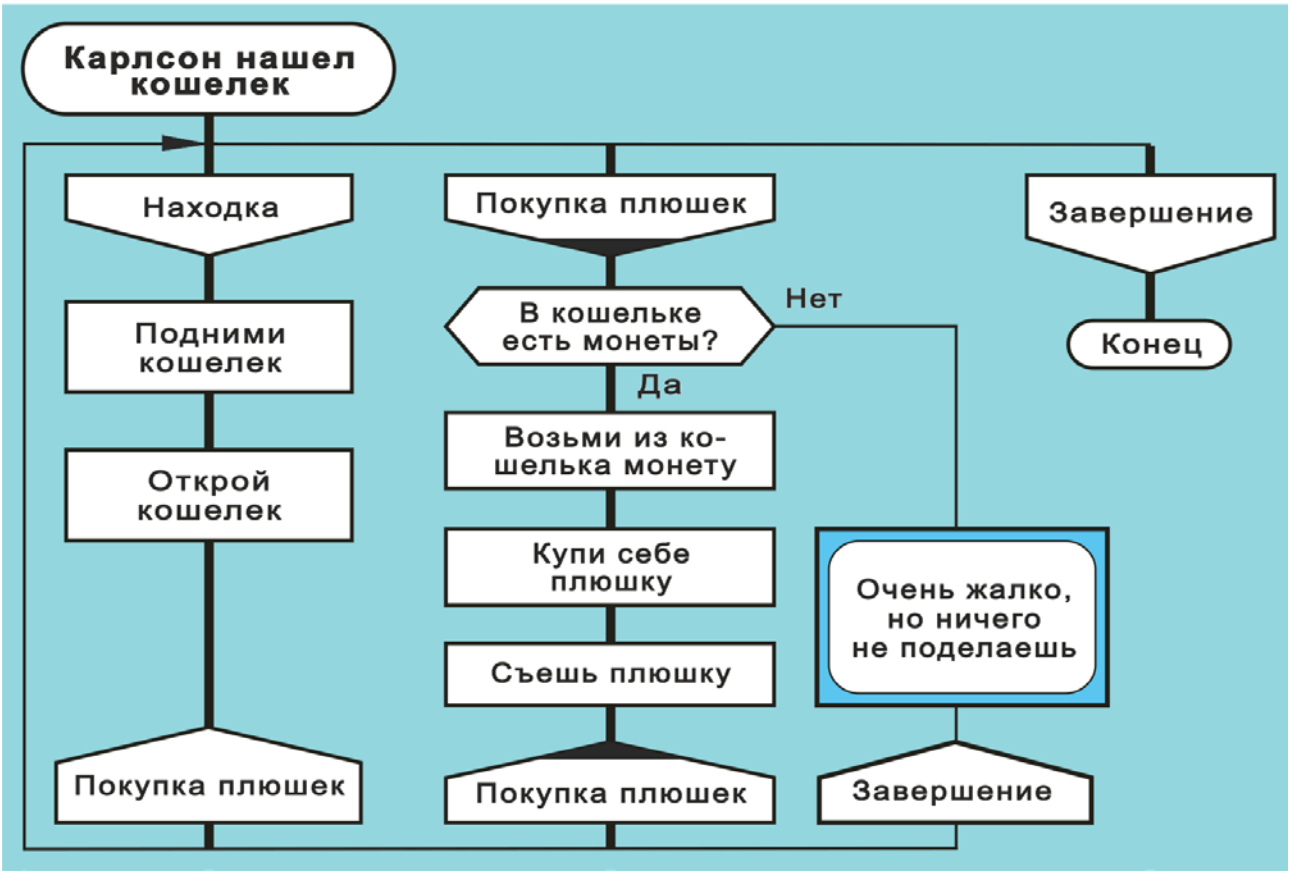


Рис. 54. Алгоритм силуэт «Карлсон нашел кошелек» с веточным циклом. Сравни с рис. 46

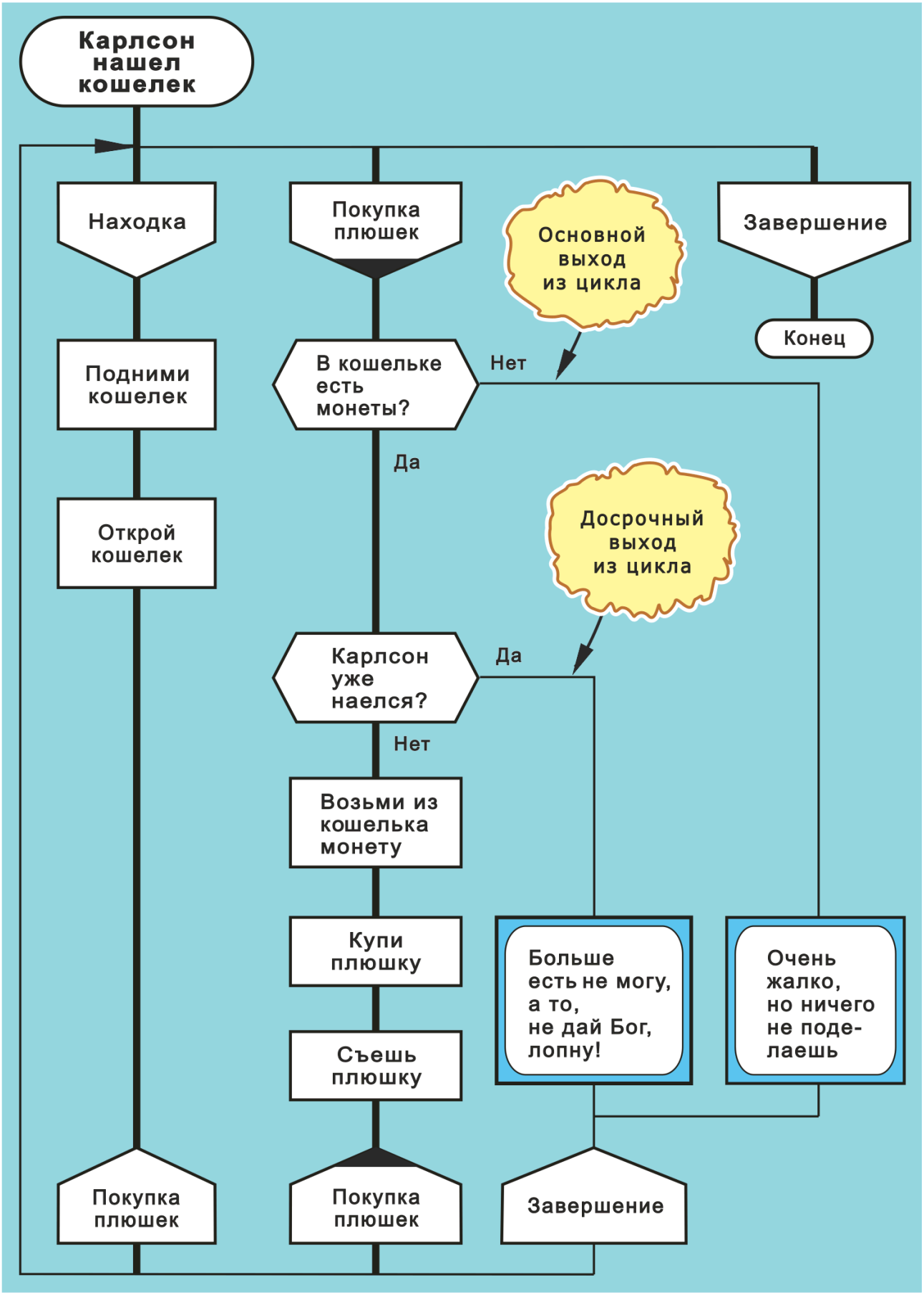


Рис. 55. Алгоритм силуэт «Карлсон нашел кошелек» с досрочным выходом. Найдите веточный цикл. Найдите досрочный выход. Сравните с рис. 48

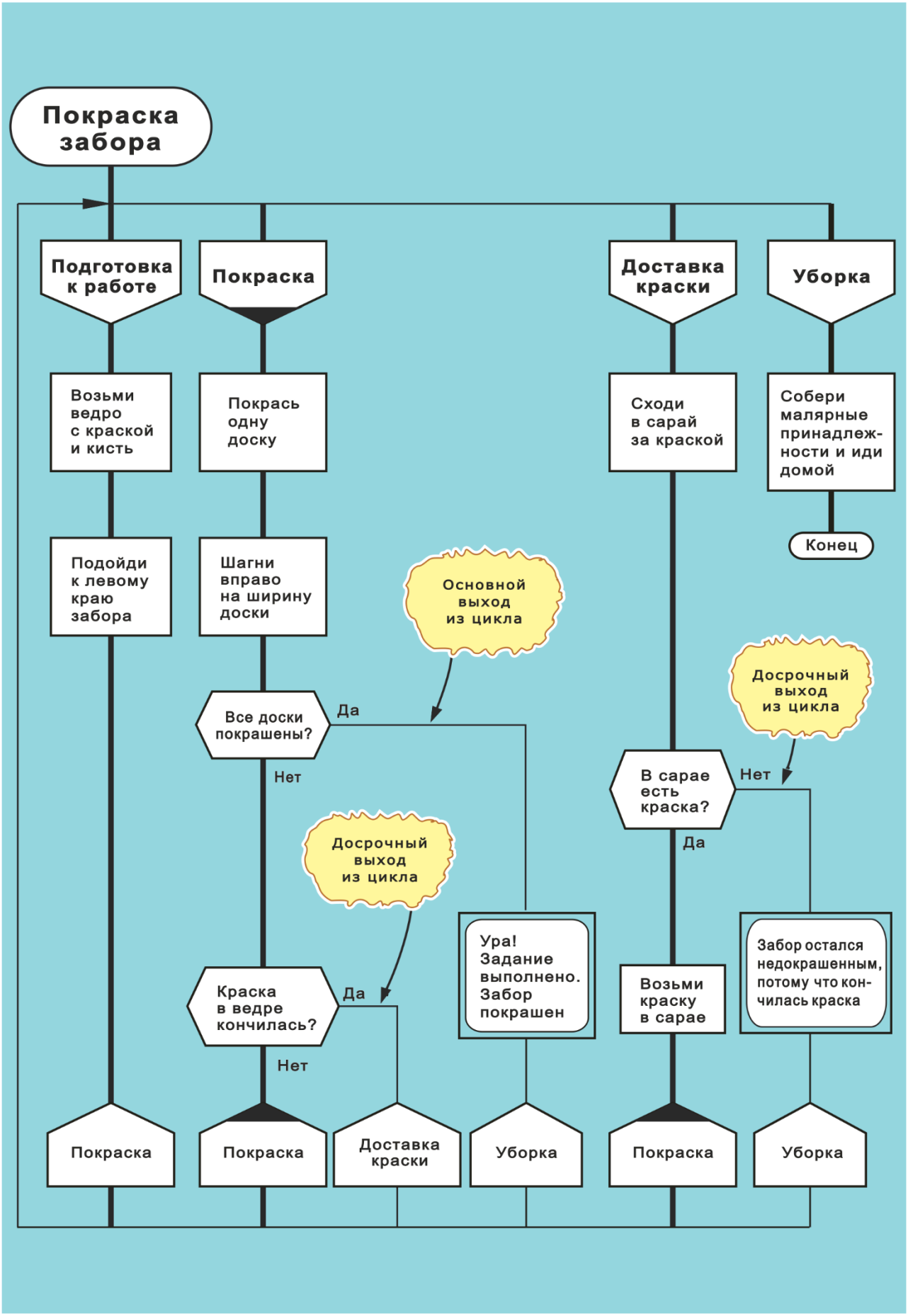


Рис. 56. Алгоритм силуэт «Покраска забора» с двумя досрочными выходами. Найдите веточный цикл. Найдите досрочные выходы. Сравните с рис. 50

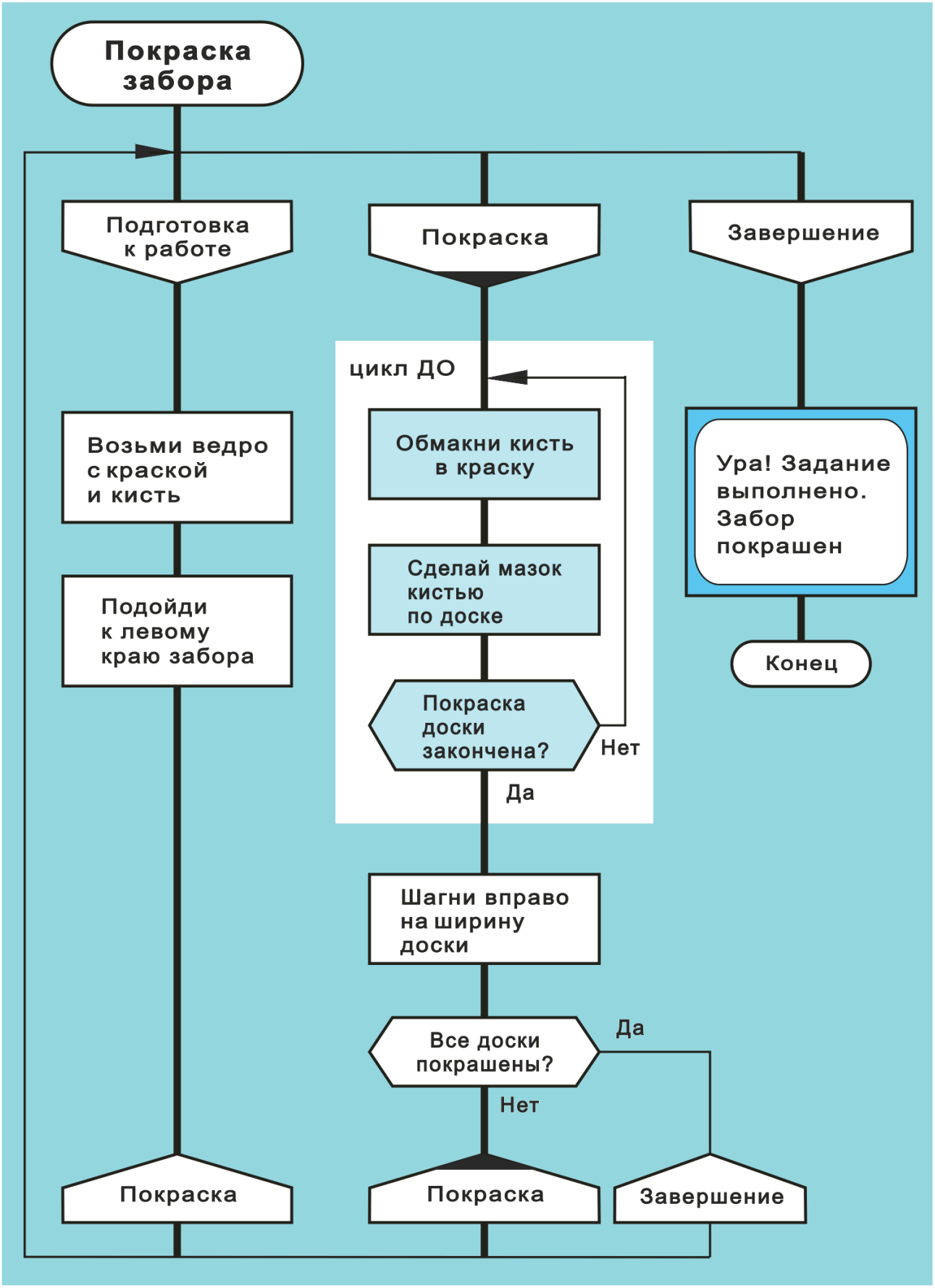


Рис. 57. Цикл ДО внутри веточного цикла.
Найдите цикл ДО и веточный цикл. Объясните, как они работают

СМЫСЛ ГОВОРИТ О МНОГОМ

Силуэт делится на ветки не случайно, а по смыслу. В иконах Имя ветки следует писать не первое попавшееся название, а название, точно и полно отражающее смысл.

Смысловые названия веток очень важны. Они облегчают чтение алгоритма и содействуют глубокому пониманию.

КОГДА В ИКОНЕ «ИМЯ ВЕТКИ» ПИШУТ СЛОВО «ЗАВЕРШЕНИЕ»

Рассмотрим случай, когда в последней ветке силуэта находятся всего две иконы: «Имя ветки» и «Конец». В этом случае в иконе Имя ветки пишут слово Завершение (рис. 54, 55).

Если в последней ветке (кроме икон Имя ветки и Конец) нарисованы одна или несколько икон Комментарий, в иконе «Имя ветки» также пишут Завершение (рис. 53, 57).

Правило

При описанных условиях запрещается заменять слово «Завершение» каким-либо другим словом

Если же в последней ветке (кроме икон Имя ветки и Конец) находится икона, отличная от Комментария, вместо слова Завершение пишут слово, обозначающее смысл, т. е. содержание ветки.

Например, на рис. 56 использовано смысловое слово Уборка.

ВЫВОДЫ

1. *Веточный цикл* — это повторное исполнение одной и той же ветки.
2. Чтобы построить веточный цикл, нужно написать в иконе Адрес название данной ветки (или более левой ветки).
3. Веточный цикл используется только в силуэте.
4. Как и любой цикл, веточный цикл может иметь основной и досрочный выходы.
5. В веточном цикле в иконах Имя ветки и Адрес используются черные треугольники (маркеры).
6. Маркеры привлекают внимание и позволяют легко заметить веточный цикл даже при беглом взгляде.
7. Если в последней ветке силуэта находятся только иконы: Имя ветки, Конец и, возможно, Комментарий, в иконе Имя ветки пишут слово «Завершение».

Глава 10

ЦИКЛ СО СЧЕТЧИКОМ

ПРОСТАЯ МАТЕМАТИЧЕСКАЯ ЗАДАЧА: ВЫЧИСЛЕНИЕ ФАКТОРИАЛА

Факториал натурального числа *n* определяется как произведение всех натуральных чисел от 1 до *n* включительно. Обозначается *n*!, произносится эн факториал.

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$

Пример

$$6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720$$

Итак, мы хотим вычислить факториал. Прежде, чем приступить к работе, следует познакомиться с командой «присвоить».

КОМАНДА «ПРИСВОИТЬ»

Рассмотрим выражение X = X + 1. Что означает знак = в этом выражении?

Предположим, что знак = означает равно. Мы сразу придем к противоречию. Действительно, если вычесть X из обеих частей уравнения, получим

$$0 = 1$$

что неверно.

Дело в том, что в алгоритмах знак = имеет совершенно другой смысл. Он означает не «равно», а «присвоить».

Команда «присвоить» X = X + 1 означает, что переменная X в левой части выражения увеличивает свое значение на 1.

Во избежание путаницы для команды «присвоить» вместо знака = используют знак «:=» (двоеточие и равно). Говорят, что команда X := A присваивает переменной X значение A.

Предположим, что переменная X имеет значение 3, а нам нужно изменить это значение и сделать его равным 157.

Что для этого нужно? В алгоритме нужно написать команду X := 157. Команда означает, что старое значение 3 стирается и переменной X будет присвоено новое значение 157.

ВЫЧИСЛЕНИЕ ФАКТОРИАЛА С ПОМОЩЬЮ ЦИКЛА «ДО»

Решение задачи с помощью цикла ДО представлено на рис. 58. В иконе Комментарий описана задача с использованием математических обозначений, где знак = обозначает «равно».

Ниже записан алгоритм, в котором используется операция присвоить :=

Предположим, что n = 10 и нужно вычислить факториал 10!

В двух верхних иконах Действие указаны команды

- X := 1
- k := 1,

которые присваивают переменным X и k значение 1 (рис. 58).

Далее бегунок спускается в икону Действие, где написано

- X := kX

(это первая команда цикла ДО).

Что делает команда? Она вычисляет kX, получает 1 и присваивает переменной X значение 1. Значит, X = 1.

Следующая команда

- k := k + 1

присваивает переменной k значение 2.

Зачем нужна переменная k? Что она делает? Она считает число проходов бегунка через цикл ДО. Когда переменная k примет значение 10, мы вычислим факториал 10! = 3628800.

В иконе Вопрос записана формула

- k <= n

Поскольку мы предположили, что n = 10, в конце первого прохода по циклу данная формула принимает вид:

- 2 <= 10

Так как 2 меньше 10, на Вопрос отвечаем «Да» и бегунок по стрелке возвращается к началу цикла.

В конце 2-го прохода по циклу формула k <= n принимает вид:

- 3 <= 10

Остальные данные представлены в таблице. Из таблицы видно, что в конце 10-го прохода по циклу мы получим искомый результат X = 10! = 3628800. При этом формула k <= n принимает вид:

- 11 <= 10

Поскольку 11 больше 10, на Вопрос отвечаем «Нет», бегунок выходит вниз и цикл ДО заканчивается.

Какой проход по циклу	Перемен-ная цикла k	X = kX	k = k + 1	n	X = n!	k <= n n = 10	Выход иконы Вопрос
1	1	1	2	1	1! = 1	2 <= 10	Да
2	2	2	3	2	2! = 2	3 <= 10	Да
3	3	6	4	3	3! = 6	4 <= 10	Да
4	4	24	5	4	4! = 24	5 <= 10	Да
5	5	120	6	5	5! = 120	6 <= 10	Да
6	6	720	7	6	6! = 720	7 <= 10	Да
7	7	5040	8	7	7! = 5040	8 <= 10	Да
8	8	40320	9	8	8! = 40320	9 <= 10	Да
9	9	362880	10	9	9! = 362880	10 <= 10	Да
10	10	3628800	11	10	10! = 3628800	11 <= 10	Нет

ВЫЧИСЛЕНИЕ ФАКТОРИАЛА С ПОМОЩЬЮ ЦИКЛА «ДЛЯ» (for loop)

На рис. 58 и 59 показаны два варианта вычисления факториала: слева используется цикл ДО, справа — цикл ДЛЯ.

Варианты имеют много общего. В обоих случаях:

- факториал обозначается через X, а переменная цикла через k;
- до начала работы цикла задается начальное значение X := 1;
- в теле цикла пишут команду X := kX.

А в чем отличие? Во-первых, вместо стрелки на рис. 59 появились две новые иконы:

- начало цикла ДЛЯ,
- конец цикла ДЛЯ (рис. 19, пункт 12 и 13).

Во-вторых, работа с переменной k задается по-другому. Бесследно исчезают три команды:

- k := 1
- k := k + 1
- k <= n

Вместо них в иконе «Начало цикла ДЛЯ» размещаются:

- переменная цикла k,
- ее начальное и конечное значения. Например, от k = 1 до n,
- шаг цикла.

ЦИКЛ СО СЧЕТЧИКОМ (count-controlled loop)

Цикл ДЛЯ — это цикл со счетчиком. При чем здесь счетчик?

Иногда заранее известно, сколько раз должен выполняться цикл. Для задач такого типа служит цикл ДЛЯ. Если известно, что цикл должен выполняться 200 раз, в иконе «Начало цикла ДЛЯ» пишут

- k
- от k = 1 до 200 (рис. 59).

Переменная k пробегает значения от 1 до 200. Это значит, что цикл будет выполняться ровно 200 раз. Переменную k часто называют счетчиком (loop counter).

Счетчик и переменная цикла — одно и то же.

Счетчик цикла считает проходы по циклу. Проходы — это шаги. Если k = 1, это первый шаг, или первый проход по циклу. Если k = 200, это двухсотый шаг, или двухсотый проход по циклу. В нашем случае 200 есть граничное значение счетчика цикла, после которого происходит выход из цикла.

Иногда говорят не шаг, а шаг изменения (переменной цикла). «Если шаг изменения равен единице, то он не указывается» [6]. На рис. 59 в иконе «Начало цикла ДЛЯ» шаг не указан.

Предположим, шаг равен 2. Это значит, что переменная цикла k будет пробегать значения: 1, 3, 5, 7, 9, …

Икона «Конец цикла ДЛЯ» служит графической границей цикла. Она отделяет цикл от остальной дракон-схемы. В ней нужно писать «Конец цикла k». Запрещается оставлять икону пустой.

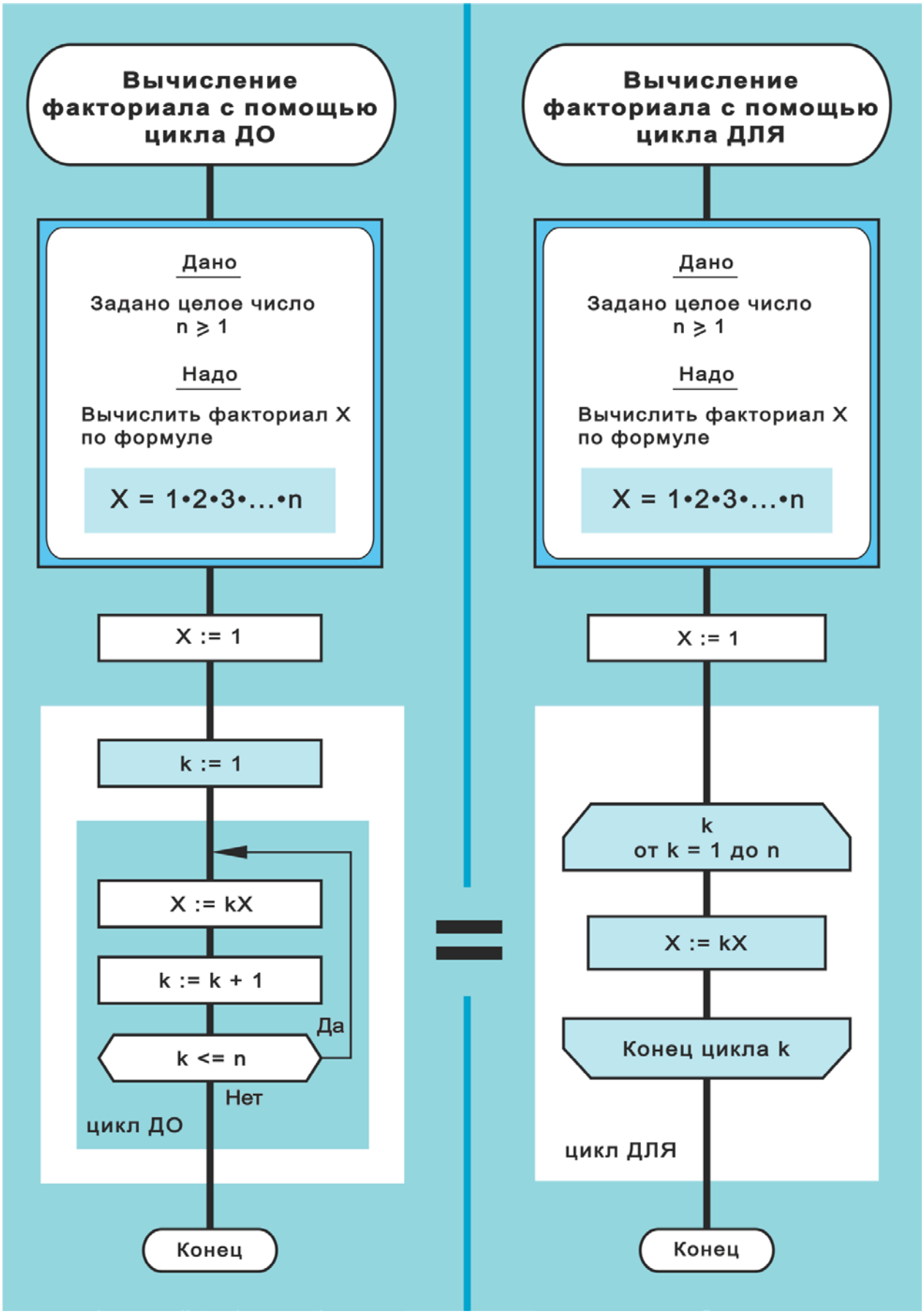


Рис. 58. Как вычислить факториал X = n! с помощью цикла ДО

Рис. 59. Как вычислить факториал X = n! с помощью цикла ДЛЯ

Внутри иконы «Начало цикла ДЛЯ» указываются переменная цикла, ее начальное и конечное значения и шаг (если шаг отличен от 1). Правила записи этих величин определяется выбранным вариантом текстового синтаксиса. На рис. 59 изображен вариант, по умолчанию принимающий, что шаг равен 1.

ИКОНА «ПОЛКА» И ОПИСАНИЕ ДАННЫХ

Мы продолжаем рассматривать примеры, связанные с циклом ДЛЯ. Перед этим следует рассказать об иконе Полка, которая нужна для описания данных.

На верхнем этаже Полки пишут ключевое слово, например, «Данные» (рис. 60). На нижнем этаже указывают описание данных. В нашем примере оно занимает три строки.

В верхней строке описан одномерный массив с именем «Вес.кролика», содержащий 100 элементов. Каждый элемент массива является вещественным числом (ВЕЩ).

```
МАССИВ ВЕЩ Вес.кролика [100]
```

Средняя строка (слева) говорит, что имя «Общий.вес.всех.кроликов» также имеет тип данных ВЕЩ.

```
ВЕЩ Общий.вес.всех.кроликов
```

Последняя строка указывает, что имя k является целым числом ЦЕЛ.

```
ЦЕЛ k
```

Рис. 60. Икона Полка может использоваться для описания данных

ВЫЧИСЛЯЕМ ВЕС ВСЕХ КРОЛИКОВ С ПОМОЩЬЮ ЦИКЛА «ДЛЯ»

Рассмотрим задачу. В крольчатнике сто клеток, причем некоторые клетки пустые, а в остальных по одному кролику. Каков суммарный вес всех кроликов?

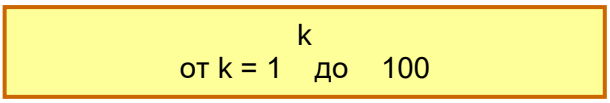
Алгоритм расчета показан на рис. 61.

Вверху в иконе Комментарий дано условие задачи. Чуть ниже в иконе Полка описаны данные.

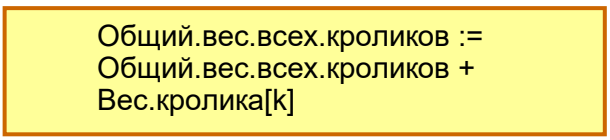
Затем в иконе Действие обнуляем основную переменную:

```
Общий.вес.всех.кроликов := 0
```

Ниже размещаем цикл ДЛЯ. В иконе «Начало цикла ДЛЯ» в верхней строке пишем переменную цикла k, а во второй строке — диапазон ее изменения.



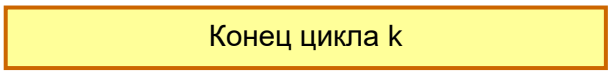
В теле цикла помещаем команду присвоить со сложением:



При первом проходе цикла команда прибавляет к нулю вес кролика в первой клетке, а если она пустая, прибавляет 0. При следующих проходах прибавляет по очереди вес кроликов во всех остальных клетках.

На сотом проходе суммирование будет закончено и получен искомый результат — общий вес всех кроликов.

В иконе «Конец цикла ДЛЯ» размещена последняя надпись:



ВЫЧИСЛЯЕМ МАКСИМАЛЬНЫЙ ВЕС ОДНОГО КРОЛИКА

На рис. 62 представлен еще один алгоритм с циклом ДЛЯ. Условия задачи те же самые. Разница лишь в том, что необходимо узнать максимальный вес кролика в крольчатнике.

Выделяем в памяти компьютера ячейку с именем «Максимальный.вес.кролика». До начала цикла в эту ячейку поместим вес кролика из первой клетки. А затем в каждом проходе цикла будем сравнивать этот вес (назовем его «помеченным») с весом кролика в очередной клетке.

Если вес следующего кролика больше помеченного, выходим из иконы Вопрос вниз через «Да» и заносим его в «максимальную» ячейку. Если меньше, выходим вправо через «Нет», возвращаемся к началу цикла и выполняем следующий проход.

Легко видеть, что «помеченный» вес, находящийся в ячейке с именем «Максимальный.вес.кролика», будет неуклонно расти. И после окончания алгоритма он действительно превратится в максимальный вес кролика в крольчатнике.

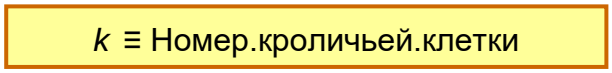
ФИЗИЧЕСКИЙ СМЫСЛ ПЕРЕМЕННОЙ ЦИКЛА

Что означает переменная цикла *k* на рис. 61 и 62? Каков ее физический смысл в данном алгоритме?

Многие программисты забывают (или не считают нужным) ответить на этот вопрос. В результате смысл буквы *k* остается неясным, а сама программа нередко становится непонятной.

Чтобы этого не случилось, можно применить эргономический прием.

В иконе «Начало цикла ДЛЯ» в нижней строке желательно написать формализованный комментарий, например:



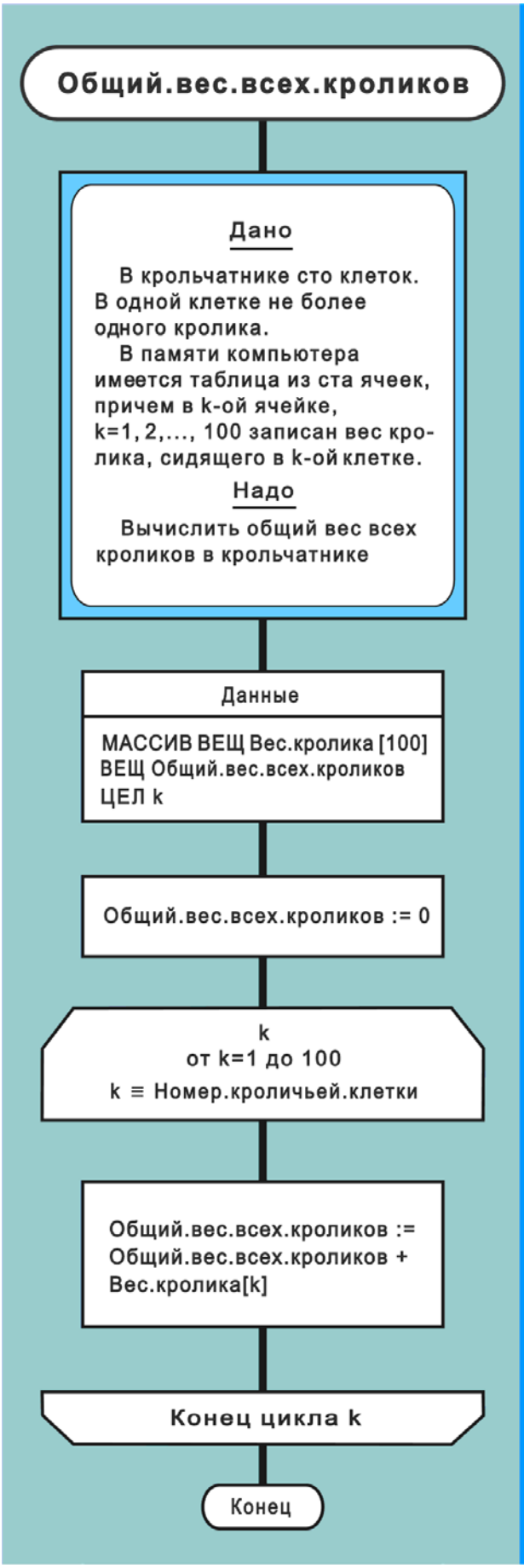


Рис. 61. Программа, вычисляющая общий вес всех кроликов в крольчатнике

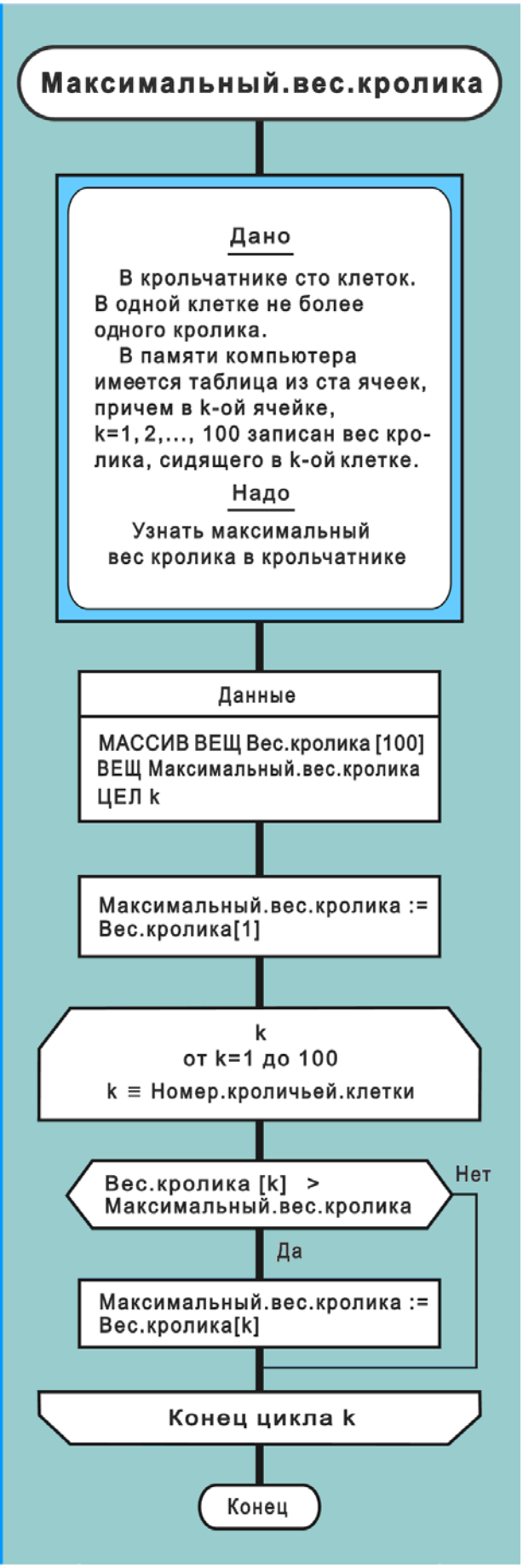


Рис. 62. Программа, определяющая максимальный вес одного кролика в крольчатнике

Знак тождественного равенства ≡ показывает, что после него следует имя-комментарий, то есть комментарий, который пишется по правилам записи идентификаторов.

Эргономическое преимущество формализованного комментария состоит в следующем.

- Во-первых, он позволяет устранить традиционную «забывчивость» программистов и доходчиво объяснить читателю смысл абстрактного идентификатора. Дескать, *k* — это номер кроличьей клетки.
- Во-вторых, что немаловажно, объяснение размещается на чертеже именно там, где нужно (в иконе «Начало цикла ДЛЯ»). Это значит, что читатель получает ответ моментально — в ту самую секунду, когда он впервые увидел букву *k* и в его голове забрезжил вопрос: а что такое *k*?

Таким образом, формализованный комментарий «Номер.кроличьей.клетки» превращает математический символ *k* в текст на естественном языке. И за счет этого делает символ *k* значительно более понятным.

ВЫВОДЫ

1. Во избежание путаницы для операций «равно» и «присвоить» желательно использовать разные знаки.
2. В команде «присвоить» вместо знака = можно использовать знак «:=» (двоеточие и равно).
3. Команда X := A присваивает переменной X значение A.
4. Цикл ДЛЯ состоит из трех элементов:
   - икона «Начало цикла ДЛЯ
   - икона «Конец цикла ДЛЯ
   - между ними тело цикла ДЛЯ.
5. В иконе «Начало цикла ДЛЯ» размещаются:
   - переменная цикла,
   - ее начальное и конечное значения,
   - шаг цикла (если он не равен 1).
6. Переменная цикла считает проходы по циклу, или шаги.
7. Переменную цикла часто называют счетчиком.
8. Цикл ДЛЯ — это цикл со счетчиком.
9. Икона «Конец цикла ДЛЯ» служит графической границей цикла. Она отделяет цикл от остальной дракон-схемы.
10. В ней нужно писать «Конец цикла <переменная цикла>». Запрещается оставлять икону пустой.

Глава 11

ЦИКЛ ВНУТРИ ДРУГОГО ЦИКЛА

ВВЕДЕНИЕ

Изучим ситуацию, когда цикл Стрелка размещен внутри другого цикла Стрелка. В данной главе будут представлены четыре варианта:

- цикл ДО в цикле ДО;
- цикл ДО в цикле ПОКА;
- цикл ПОКА в цикле ДО;
- цикл ПОКА в цикле ПОКА.

ЦИКЛ «ДО» ВНУТРИ ЦИКЛА «ДО»

Рассмотрим графическую конструкцию «цикл в цикле». Пусть это будет цикл ДО в цикле ДО.

На рис. 47 есть команда «Покрась одну доску». Взглянем на нее «под микроскопом». Чтобы покрасить доску, надо сделать несколько операций: макнуть кисть в краску, сделать мазок, потом еще и еще — до тех пор, пока вся доска не станет окрашенной. Эти действия можно изобразить в виде цикла.

Полученный цикл мысленно поместим на рис. 47 вместо команды «Покрась одну доску». Результат показан на рис. 63. Мы получили алгоритм, содержащий конструкцию «цикл в цикле».

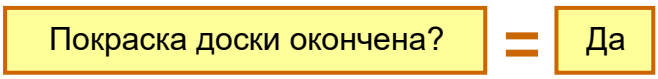
Алгоритм работает так. Предположим, забор красит Том Сойер. Сначала Том выполняет две команды (рис. 63):

- Возьми ведро с краской и кисть
- Подойди к левому краю забора

Дальше нужно покрасить самую первую доску. Для этого Том исполняет команды, содержащиеся в цикле ДО (2):

- Обмакни кисть в краску
- Сделай мазок кистью по доске
- Покраска доски окончена?

Если не окончена, Том продолжает красить до тех пор, пока не будет выполнено условие окончания цикла:



Рассмотрим числовой пример. Предположим, чтобы покрасить одну доску, нужно семь раз макнуть кисть в краску и сделать семь мазков. Значит цикл ДО (2) будет выполнен ровно семь раз.

После этого Том выполняет команды цикла ДО (1):

- Шагни вправо на ширину доски
- Все доски покрашены?

Если нет (не все доски покрашены), Том примется за следующую доску.

Давайте проследим маршрут (рис. 63). Из иконы «Все доски покрашены?» выходим через «Нет» направо. По стрелке попадаем на вход цикла ДО (1). Затем Том начинает красить вторую доску. Он семь раз макнет кисть и сделает семь мазков. Разделавшись со второй доской, Том шагнет вправо и возьмется за третью доску. И так далее — пока весь забор не будет покрашен.

ЦИКЛ «ДО» ВНУТРИ ЦИКЛА «ПОКА»

В алгоритме на рис. 46 заменим икону «Съешь плюшку» на цикл ДО, состоящий из икон «Откуси кусок плюшки» и «Плюшка съедена?» (рис. 64). В итоге получим цикл в цикле. Цикл ПОКА, построенный с помощью иконы «В кошельке есть монеты?», является внешним. В нем прячется внутренний цикл ДО (рис. 64).

При выполнении цикла ДО бегунок кружит по внутренней петле. За это время Карлсон откусит один кусок плюшки, потом другой и так далее. Когда он покончит с первой плюшкой, будет выполнено условие окончания цикла ДО.

Плюшка съедена? = Да

После этого маршрут бегунка меняется. Если в кошельке по-прежнему есть деньги, бегунок побежит по внешней петле.

Предположим, в кошельке 50 монет, а Карлсон съедает плюшку за три приема. Это значит, что каждая команда цикла ПОКА будет выполнена 50 раз, а каждая команда цикла ДО — 150 раз. (Чтобы съесть 50 плюшек, откусив каждую 3 раза, нужно сделать 50 x 3 = 150 «откусываний»).

ЦИКЛ «ПОКА» ВНУТРИ ЦИКЛА «ДО»

Рассмотрим алгоритм на рис. 65. Предположим, старый пасечник Микола решил собрать мед во всех ульях своей пасеки. Сначала Микола выполняет две команды (рис. 65):

- Надень защитную маску против пчел
- Зайди на пасеку

Дальше нужно взять мед в самом первом улье. Для этого Микола забирает мед из первого улья, делая это поочередно, порцию за порцией. При этом он исполняет команды, содержащиеся в цикле ПОКА:

- Мед из данного улья собран полностью?
- Собери следующую порцию меда из данного улья

При выполнении цикла ПОКА (2) бегунок кружит по внутренней петле. За это время Микола берет одну порцию меда, потом другую и т. д. Когда он покончит с первым ульем, будет выполнено условие окончания цикла ПОКА (2).

Мед из данного улья собран полностью? = Да

После этого Микола выполняет команды:

- Переходи к следующему улью
- Мед из всех ульев собран полностью?

Если нет (если еще остались полные ульи) Микола продолжает обходить пасеку, улей за ульем. Когда он покончит с последним ульем будет выполнено условие окончания цикла ДО (1):

Мед из всех ульев собран полностью? = Да

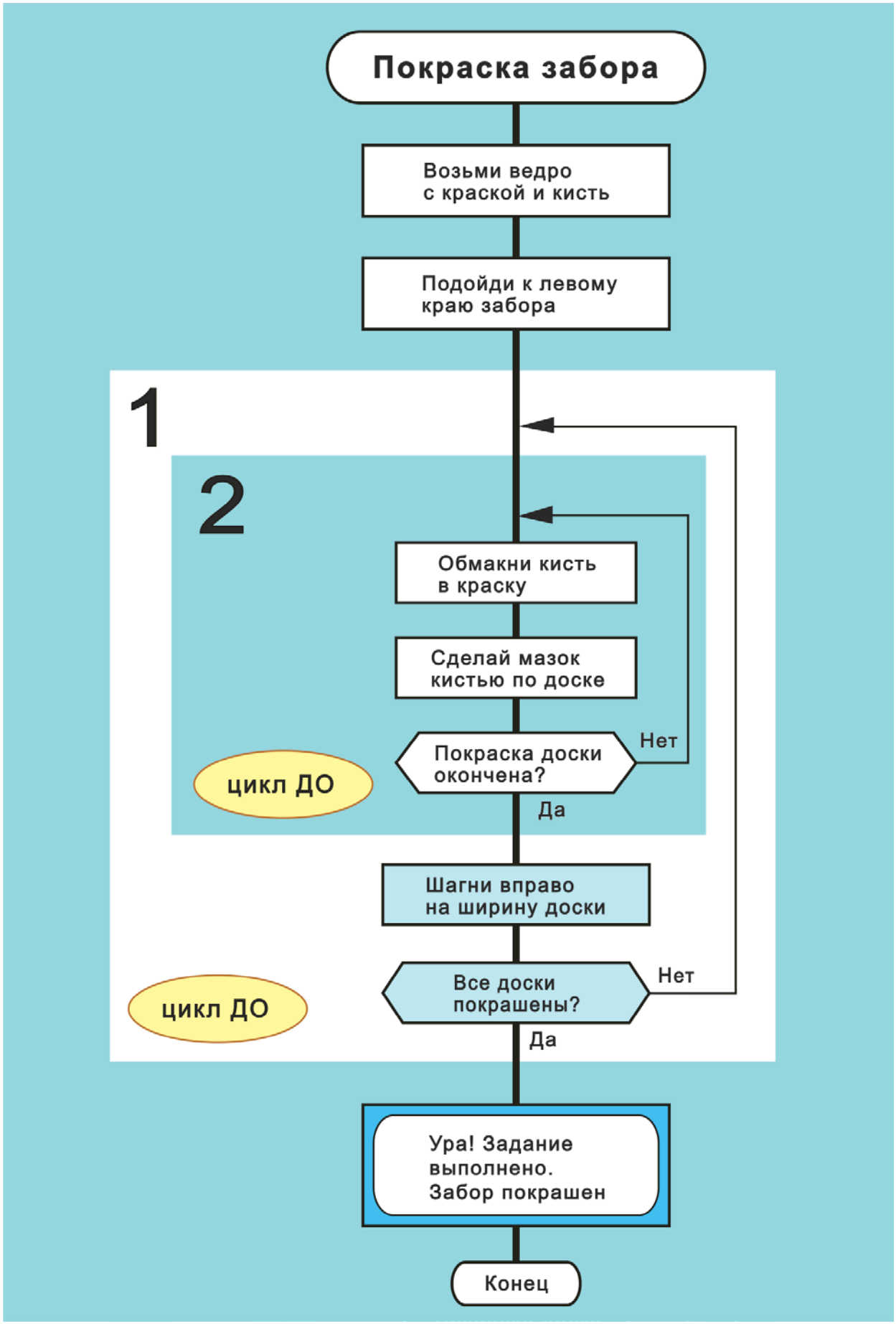


Рис. 63. Алгоритм «Покраска забора». Внутри цикла ДО (1) находится цикл ДО (2). Сравни с рис. 47

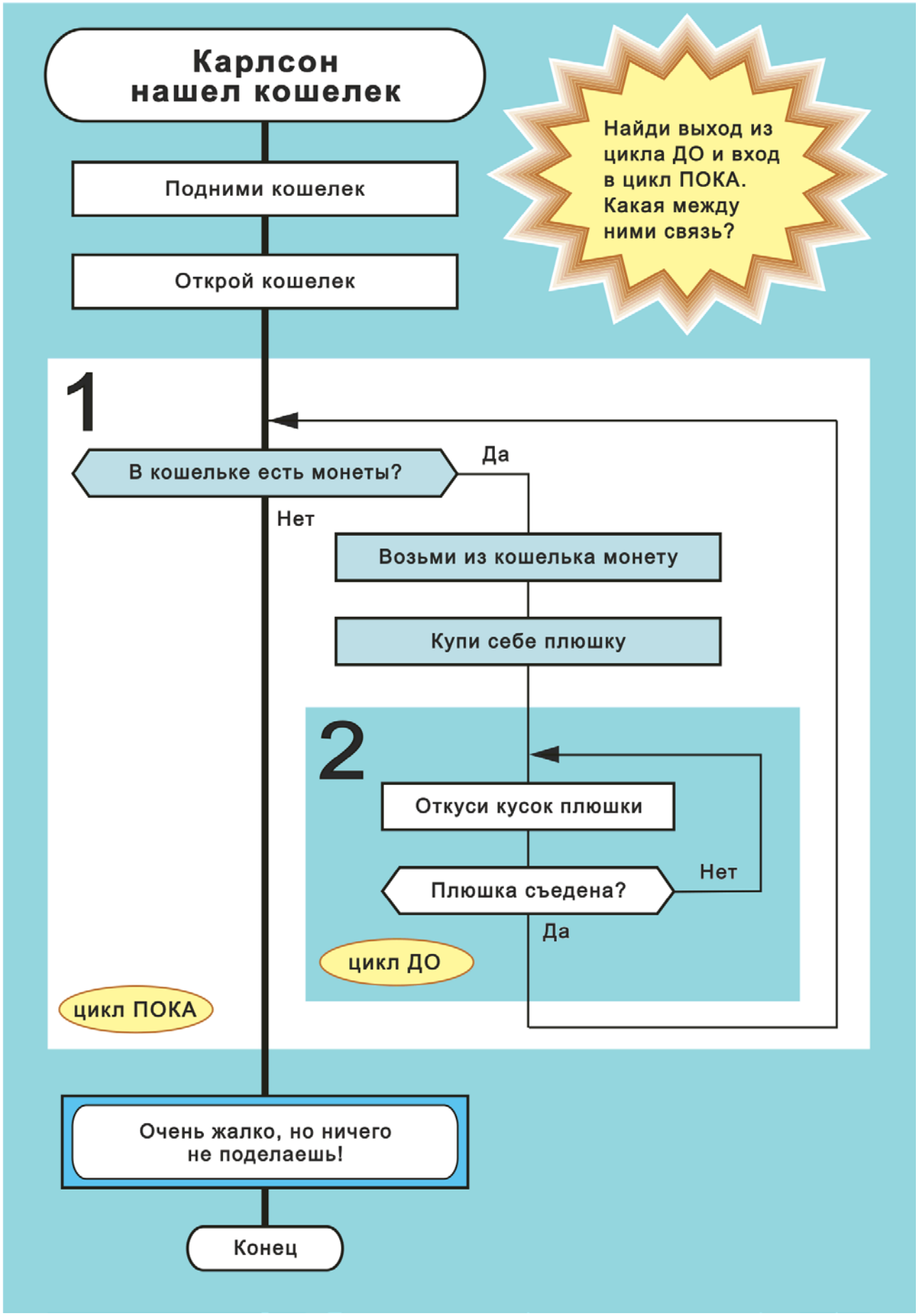


Рис. 64. Алгоритм «Карлсон нашел кошелек».
Внутри цикла ПОКА находится цикл ДО. Сравни с рис. 46

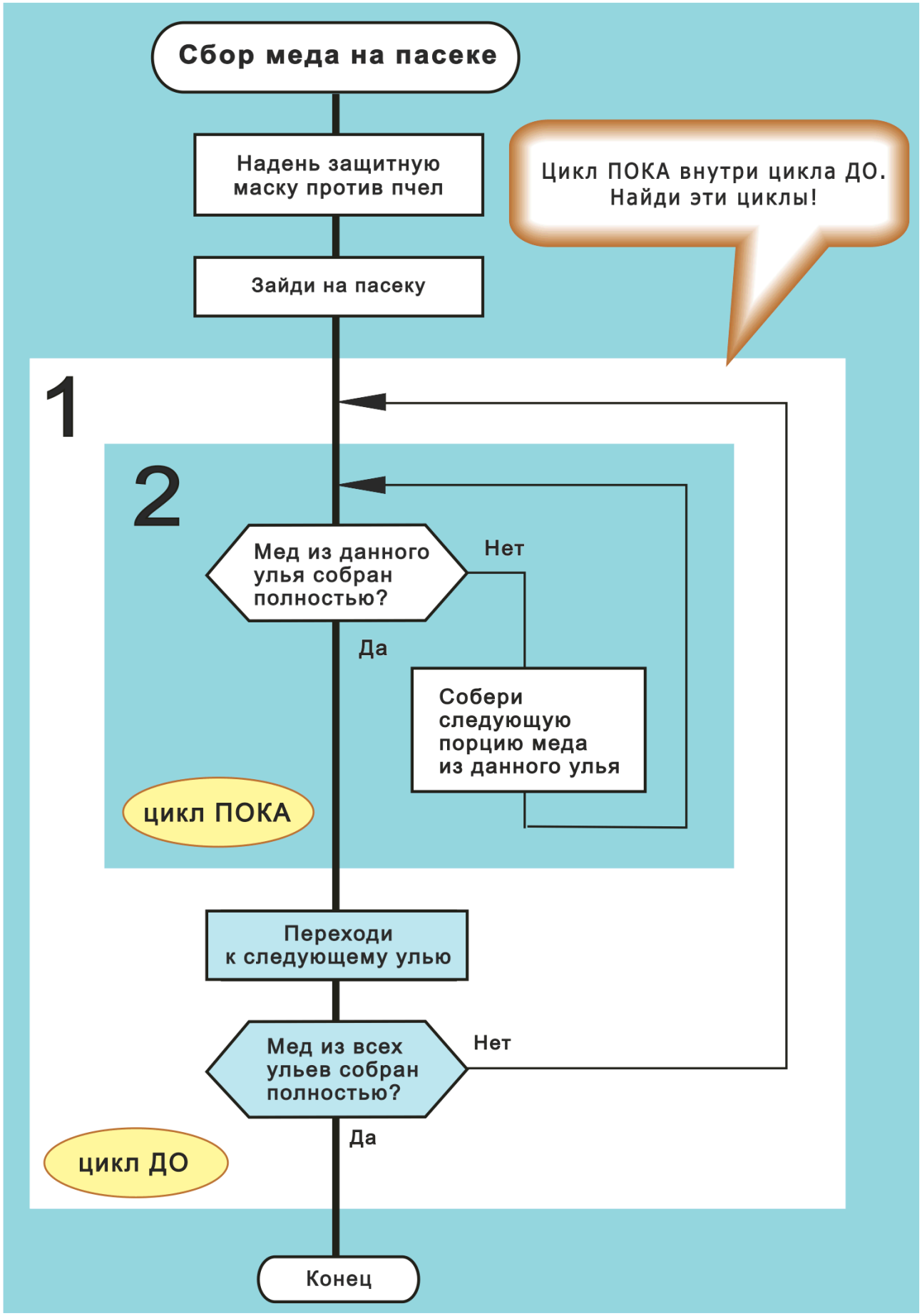


Рис. 65. Алгоритм «Сбор меда на пасеке».
Внутри цикла ДО находится цикл ПОКА

Предположим, на пасеке 30 ульев, причем из каждого улья Микола достает 6 порций меда. Это значит, что каждая команда цикла ДО будет выполнена 6 раз, а каждая команда цикла ПОКА — 180 раз. (Чтобы обойти 30 ульев, взяв из каждого 6 порций, нужно собрать 30 × 6 = 180 порций меда).

ЦИКЛ «ПОКА» ВНУТРИ ЦИКЛА «ПОКА»

Рассмотрим алгоритм на рис. 66. Это рассказ о том, как Иван получал деньги и тратил их на еду. Сначала Иван выполняет две команды (рис. 66):

- Зайди в банк
- Проверь свой счет

Прежде всего, Иван интересуется: «На счете есть деньги?». Если нет, то история заканчивается. Потому как на нет и суда нет.

Если же деньги есть, выполняется условие продолжения цикла ПОКА (1):

На счете есть деньги? = Да

В этом случае Иван выполняет команды:

- Сними со счета часть денег
- Купи себе еду на неделю

Затем начинает работу цикл ПОКА (2). Если еда есть, выполняется условие продолжения цикла:

На сегодня еда есть? = Да

Поэтому Иван выполняет команды:

- Позавтракай
- Пообедай
- Поужинай

Когда наступит следующий день, на вопрос «На сегодня еда есть?» следует ответ «Да».

При выполнении цикла ПОКА (2) бегунок кружит по внутренней петле семь раз (то есть в течение недели). Когда неделя кончится, еда тоже кончится (поскольку еда была куплена ровно на семь дней). Бегунок выходит из иконы Вопрос через «Нет». И попадает на вход внешнего цикла ПОКА.

Перед Иваном вновь возникает вопрос «На счете есть деньги?». И так далее.

СТРУКТУРА «ЦИКЛ В ЦИКЛЕ»

На рис. 67 представлены четыре варианта структуры «цикл в цикле».

- цикл ДО (2) внутри цикла ДО (1);
- цикл ДО (2) внутри цикла ПОКА (1);
- цикл ПОКА (2) внутри цикла ДО (1);
- цикл ПОКА (2) внутри цикла ПОКА (1).

Внешний цикл обозначен цифрой 1, внутренний — цифрой 2.

1-й вариант показан на рис. 49 (Покраска забора).

2-й вариант — на рис. 50 (Карлсон нашел кошелек).

3-й вариант — на рис. 51 (Сбор меда на пасеке).

4-й вариант — на рис. 52 (Деньги и еда).

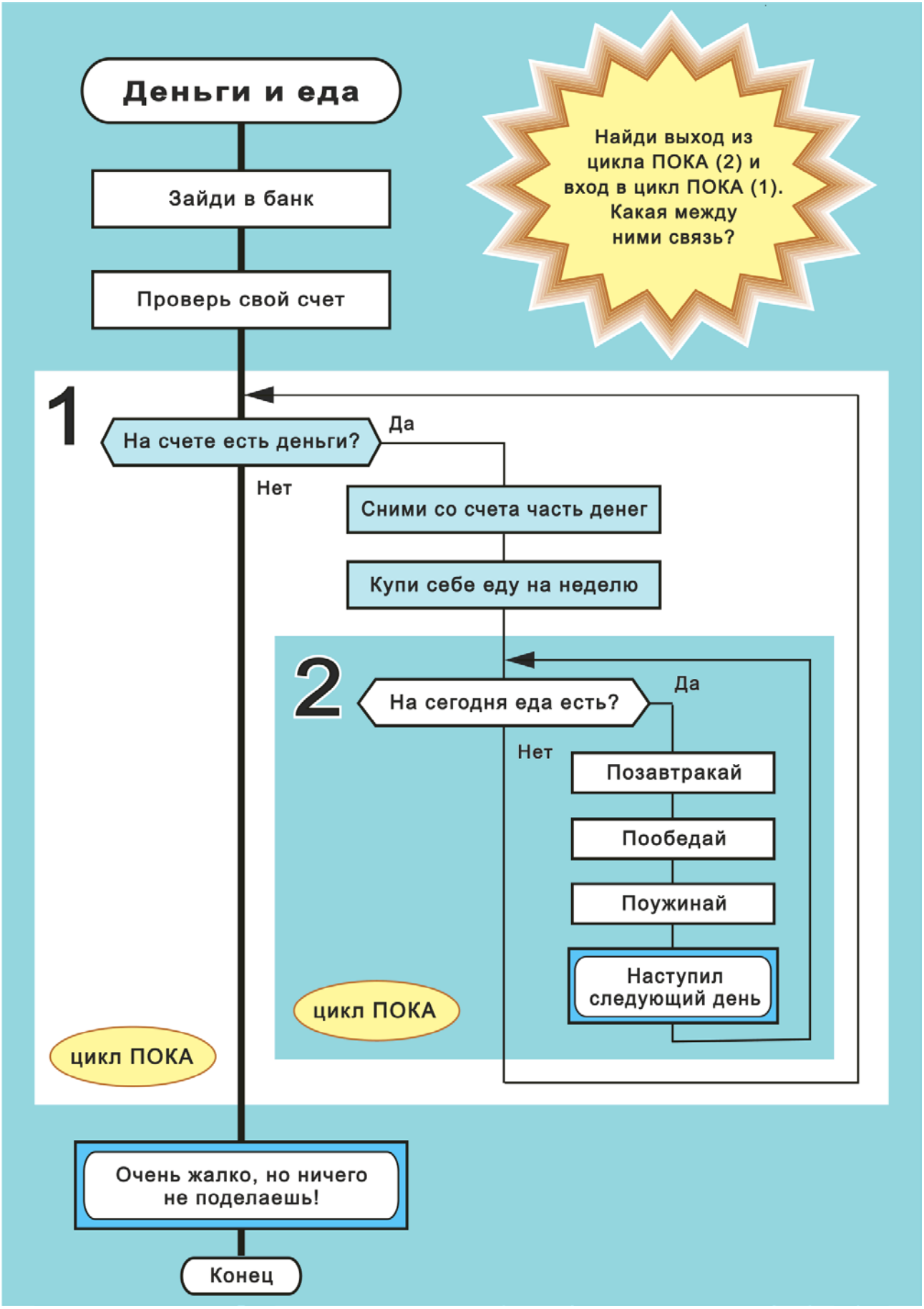


Рис. 66. Алгоритм «Деньги и еда». Внутри цикла ПОКА (1) находится цикл ПОКА (2)

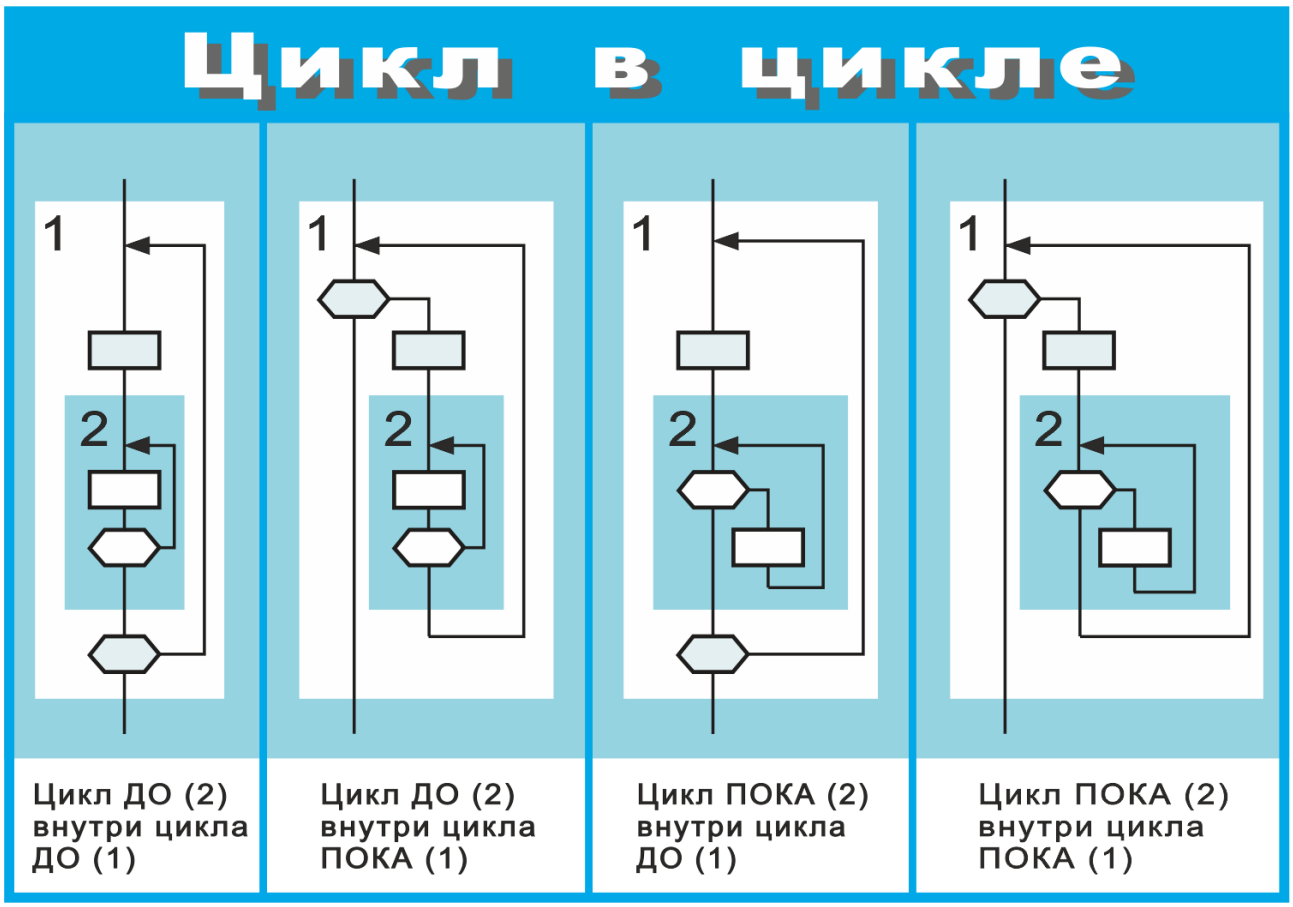


Рис. 67. Четыре варианта структуры «цикл в цикле»

Рисунок 67 задает правила графического синтаксиса, используемые при создании конструкции «цикл в цикле».

Правила не надо учить и запоминать. Почему? Потому что правила помнит и выполняет компьютерная программа, которая называется «ДРАКОН-конструктор».

Памятка

В любых дракон-схемах (в том числе, в циклах) запрещается использовать пересечение линий. Этот запрет выполняется автоматически — с помощью ДРАКОН-конструктора. Рисунки в книге подтверждают это правило. Пересечения линий нигде и никогда не используются.

ВЫВОДЫ

1. В главе рассмотрены четыре варианта структуры «цикл Стрелка в цикле Стрелка».
2. В языке ДРАКОН строго заданы правила графического синтаксиса, используемые при создании конструкции «цикл в цикле».
3. Указанные правила реализует программа «ДРАКОН-конструктор».

Часть 3

АЛГОРИТМИЧЕСКАЯ ЛОГИКА

Математическая логика в алгоритмах. Визуальная алгебра логики

Глава 12

ЛОГИЧЕСКИЕ ОПЕРАЦИИ И, ИЛИ, НЕ

ЛОГИЧЕСКАЯ ОПЕРАЦИЯ «И»

— Где можно купить щенка?

— В нашем городке они продаются на рынке, но сегодня рынок закрыт. К тому же щенков продают не каждый день. Щенки довольно дорогие и какие-то невзрачные — не знаю, понравятся ли они вам.

Упростим рассказ. Будем считать, что покупка щенка возможна в том и только в том случае, когда выполняются три условия (рис. 68):

- у покупателя есть деньги (P);
- щенки есть в продаже (Q);
- щенок понравился (R).

В итоге получаем логическую функцию

$$Z = P \,\&\, Q \,\&\, R$$

где *Z* означает «Можно купить щенка». Или просто «Купи щенка» (рис. 68).

& означает логическую операцию «И».

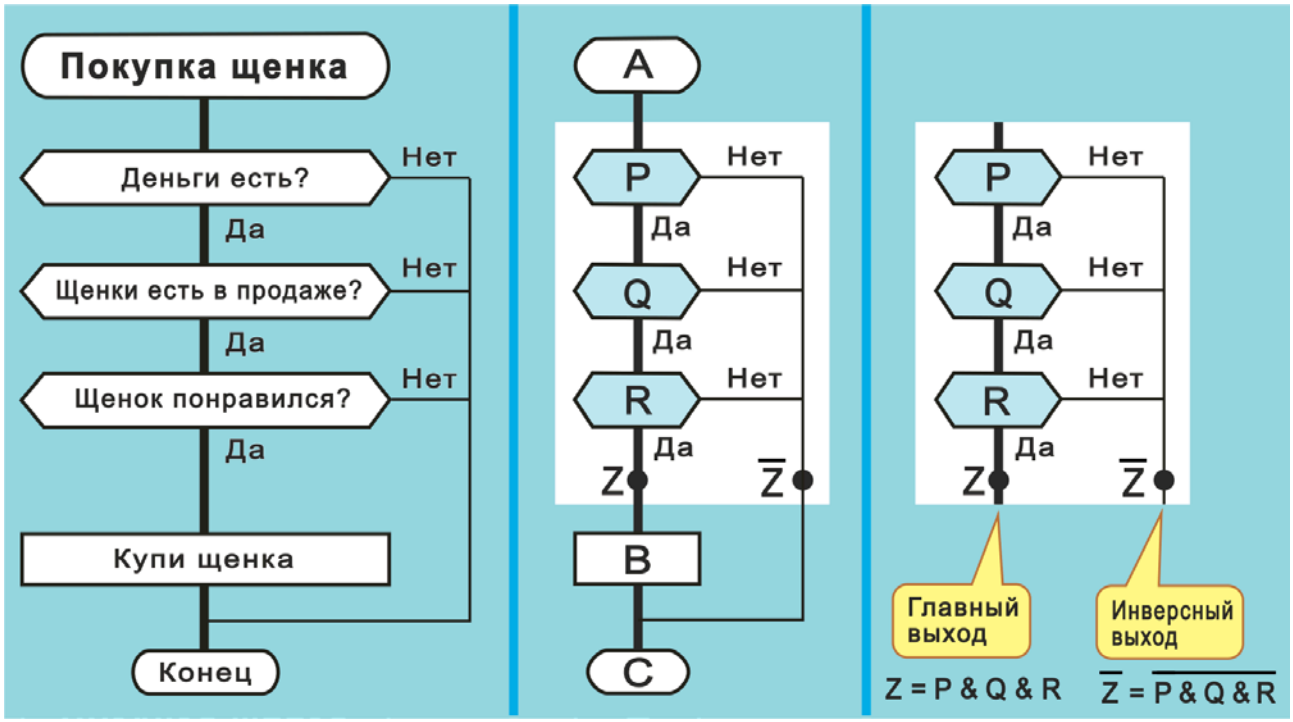


Рис. 68. Алгоритм «Покупка щенка» с операцией «И»

Проследим работу алгоритма на рис. 68. Маршрут, идущий по шампуру, последовательно получает ответ «Да» на три Вопроса:

- Деньги есть? Да.
- Щенки есть в продаже? Да.
- Щенок понравился? Да.

После этого следует команда «Купи щенка» и алгоритм завершается.

Если же хоть одно из условий не выполнено, бегунок сворачивает направо через «Нет», и покупка становится невозможной. При этом выполняется принцип «Чем правее, тем хуже»: покупка — это хорошо, неудача — плохо.

АЛГОРИТМЫ, ИСПОЛЬЗУЮЩИЕ ОПЕРАЦИЮ «И»

На рис. 68 приведены два примера алгоритмов. Слева описан рассказ о покупке щенка. В центре тот же самый алгоритм представлен в абстрактной математической форме.

Упражнение. Попытайтесь в этих алгоритмах найти и выделить логическую функцию «И». Сравните свой результат с рис. 68, справа.

ДВА СПОСОБА ЗАПИСИ ОПЕРАЦИИ «И»

В языке ДРАКОН предусмотрены два способа записи операции «И»: текстовый и визуальный (графический) — см. рис. 69.

В первом случае используют одну икону Вопрос, внутри которой пишут логическое выражение, состоящее из логических переменных, соединенных значком &. Этот значок (&) называется амперсанд и, как мы уже знаем, обозначает операцию «И» (рис. 69, слева).

В другом случае на одной вертикали рисуют несколько икон Вопрос, причем в каждой иконе записывают одну логическую переменную (рис. 69, справа).

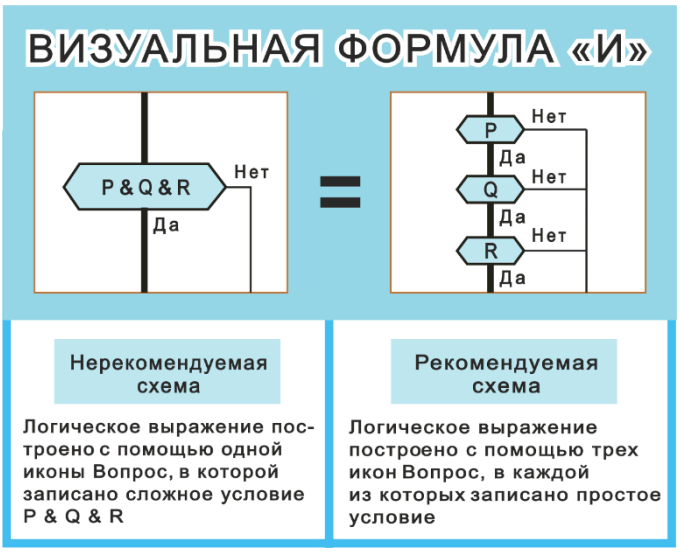


Рис. 69. Рисуйте дракон-схему «И», как показано справа. Избегайте нерекомендуемых схем

КАКОЙ СПОСОБ ЛУЧШЕ: ТЕКСТОВЫЙ ИЛИ ВИЗУАЛЬНЫЙ?

Визуальная формула на рис. 69 показывает, что оба способа эквивалентны.

Для практического использования рекомендуется визуальный способ (справа), так как он более нагляден и позволяет быстрее найти ошибку в сложном алгоритме. Следует подчеркнуть, что текстовый способ (слева) не является запрещенным. Но пользоваться им следует с осторожностью и лишь в тех случаях, когда пользователь убежден в своих способностях гарантировать отсутствие ошибок.

Опыт показывает, что большинство людей выбирает визуальный способ как более легкий. Однако подготовленные специалисты, знакомые с математической логикой, иногда предпочитают текстовый метод. Таким людям можно посоветовать освоить оба метода.

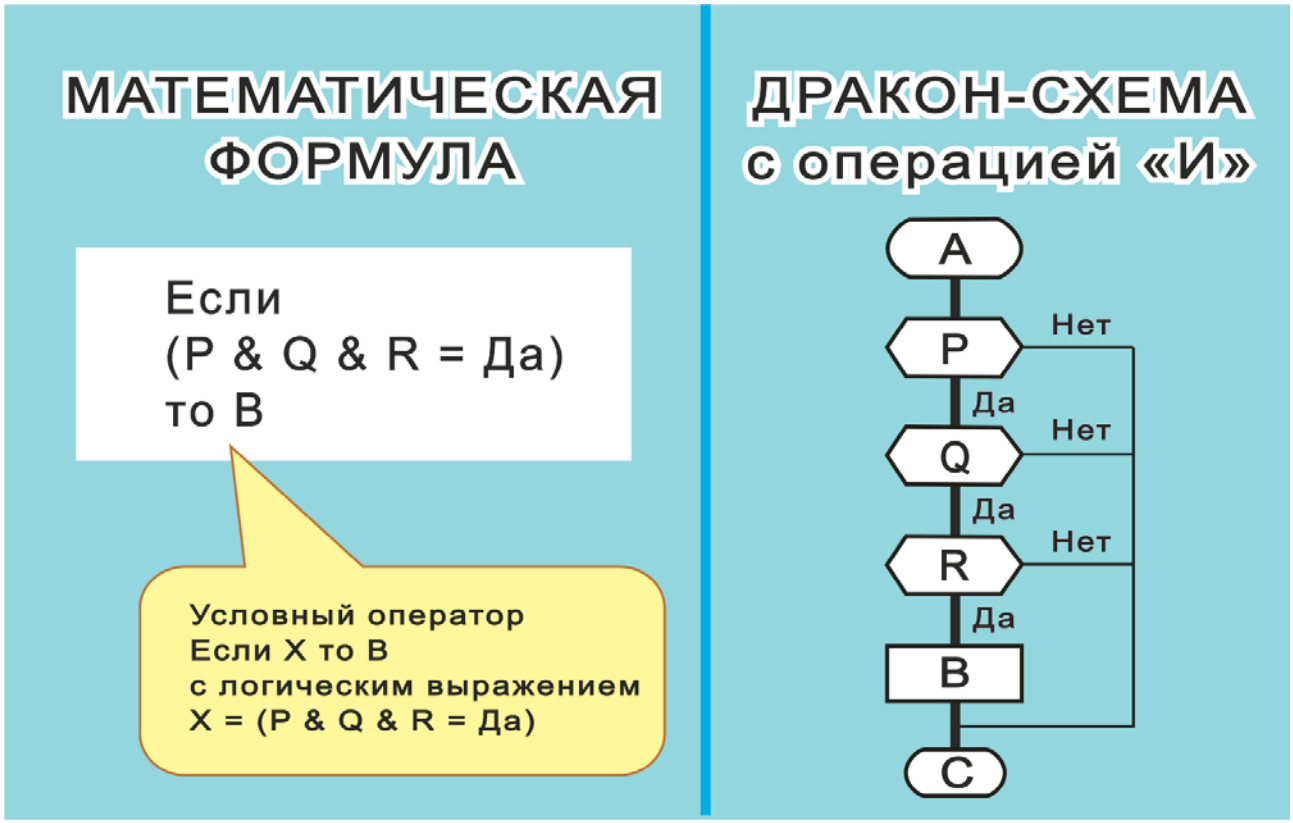


Рис. 70. Математическая формула и дракон-схема с логической операцией «И»

СРАВНЕНИЕ МАТЕМАТИЧЕСКОЙ ФОРМУЛЫ И ДРАКОН-СХЕМЫ

На рис. 70 приведена математическая формула (условный оператор) и равносильная ей дракон-схема. Какую из них следует предпочесть? Какая является более понятной? Более эргономичной?

Слева представлена традиционная формула, понятная далеко не всем.

Формула справа, написанная на языке ДРАКОН, намного легче для понимания. Она становится еще более наглядной, если заменить абстрактные буквы *P, Q, R, B* на конкретные производственные понятия. Например (рис. 71):

- *P* = Норма подачи топлива;
- *Q* = Норма зажигания;
- *R* = Норма электропитания;
- *B* = Включить двигатель.

Два объекта на рис. 70 (текстовый и графический) математически эквивалентны. Это значит, что дракон-схема является *математической формулой.*

Отсюда вытекает, что математические формулы бывают не только текстовые, но и графические.

И последнее. Анализируя формулу на рис. 70 слева, обычно приходится вникать в сложные подробности, например:

X = [P & Q & R = Да] = [(P = Да) & (Q = Да) & (R = Да)]

Дракон-схема (рис. 70, справа) хороша тем, что полностью избавляет читателя от подобной ненужной работы.

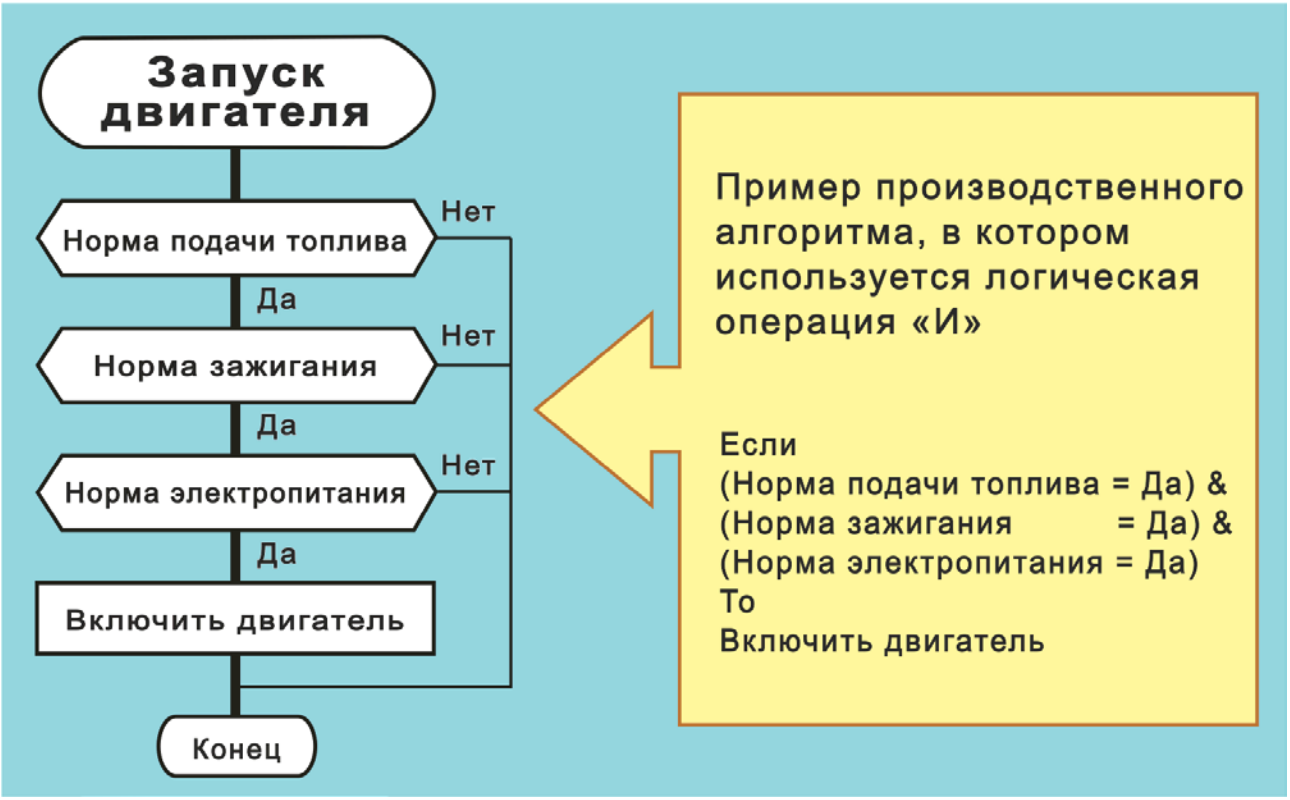


Рис. 71. Пример производственного алгоритма с операцией «И»

На рис. 68 – 71 вместо знака «И» использован знак «&» (амперсанд). Оба знака имеют одинаковый смысл и обозначают одно и то же. Далее мы будем использовать оба знака.

ЛОГИЧЕСКАЯ ОПЕРАЦИЯ «ИЛИ»

Пете не повезло — он заболел. Что с ним случилось?

На этот счет может быть много ответов. Но медициной мы заниматься не будем. Ограничимся логикой.

Для простоты будем считать, что Петя болен, если выполняется хотя бы одно из трех условий (рис. 72):

- У Пети корь (*L*);
- У Пети болит зуб (*M*);
- У Пети ушиб (*N*).

В итоге получаем логическую функцию

Z = L или M или N

где Z означает «Петя заболел». Или просто «Лечи Петю» (рис. 72).

Проследим работу алгоритма на рис. 72. Бегунок, идущий по шампуру, встречает развилку на три направления, где предлагаются три Вопроса. Предположим, что хотя бы на один вопрос получен ответ «Да»:

- У Пети корь? Да.
- У Пети болит зуб? Да.
- У Пети ушиб? Да.

В таком случае выполняется процедура «Лечи Петю», после чего алгоритм заканчивается.

Если же на все три Вопроса получен ответ «Нет», бегунок уходит на боковой маршрут и процедура не выполняется.

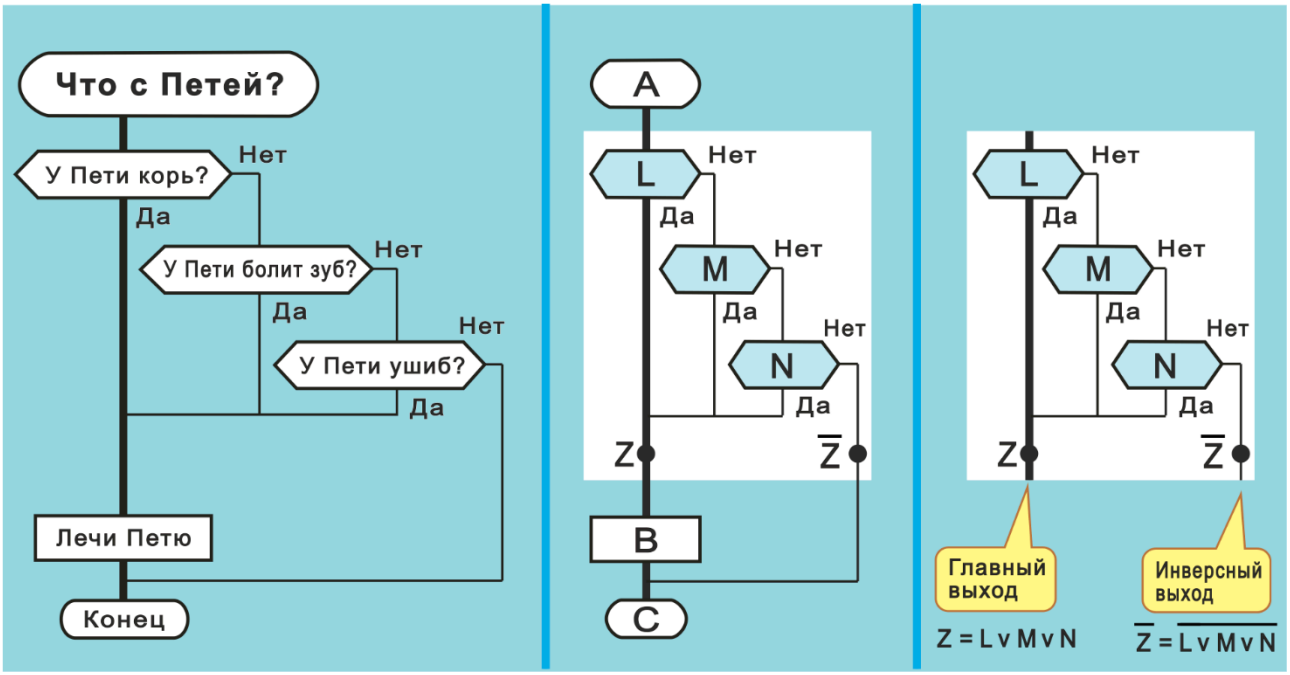


Рис. 72. Алгоритм «Что с Петей» с операцией «ИЛИ»

АЛГОРИТМЫ, ИСПОЛЬЗУЮЩИЕ ОПЕРАЦИЮ «ИЛИ»

На рис. 72 приведены два примера алгоритмов. Слева рассказ о заболевшем Пете. В центре тот же самый алгоритм представлен в абстрактной математической форме.

Попытайтесь в этих алгоритмах найти и выделить логическую функцию «ИЛИ». Сравните свой результат с рис. 72, справа.

ДВА СПОСОБА ЗАПИСИ ОПЕРАЦИИ «ИЛИ»

В языке ДРАКОН есть два способа записи операции ИЛИ: текстовый и визуальный.

В первом случае используют одну икону Вопрос, внутри которой пишут логическое выражение, состоящее из логических переменных, соединенных значком V. Этот значок (V) обозначает операцию «ИЛИ» (рис. 73, слева).

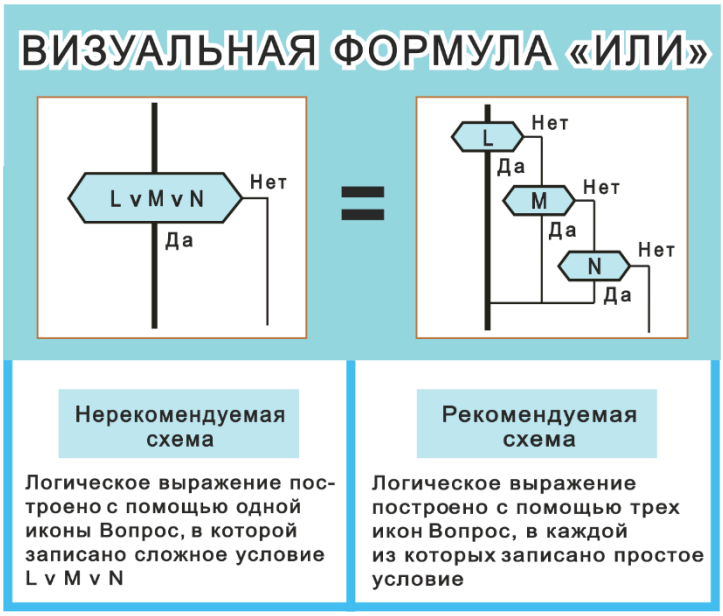


Рис. 73. Рисуйте дракон-схему «И», как показано справа. Избегайте нерекомендуемых схем

В другом случае на одной вертикали рисуют несколько икон Вопрос, причем в каждой иконе записывают одну логическую переменную (рис. 73, справа). Оба способа дают верный результат.

***Кто придумал знак* V** (по-русски читается В). Придумал английский математик Бертран Рассел в 1908 году — это первая буква латинского слова vel что означает ИЛИ.

Увы, латынь мало кто знает. Гораздо легче запомнить этот символ как жестовый знак победы (Victory).

«Виктория» в исполнении Уинстона Черчилля

КАКОЙ СПОСОБ ЛУЧШЕ: ТЕКСТОВЫЙ ИЛИ ВИЗУАЛЬНЫЙ?

Визуальная формула на рис. 73 говорит, что оба способа эквивалентны.

Для практического использования рекомендуется визуальный способ (справа), так как он более нагляден и позволяет быстрее найти ошибку в сложном алгоритме.

Текстовый способ (слева) следует использовать с осторожностью и лишь в тех случаях, когда пользователь уверен в своих способностях гарантировать отсутствие ошибок.

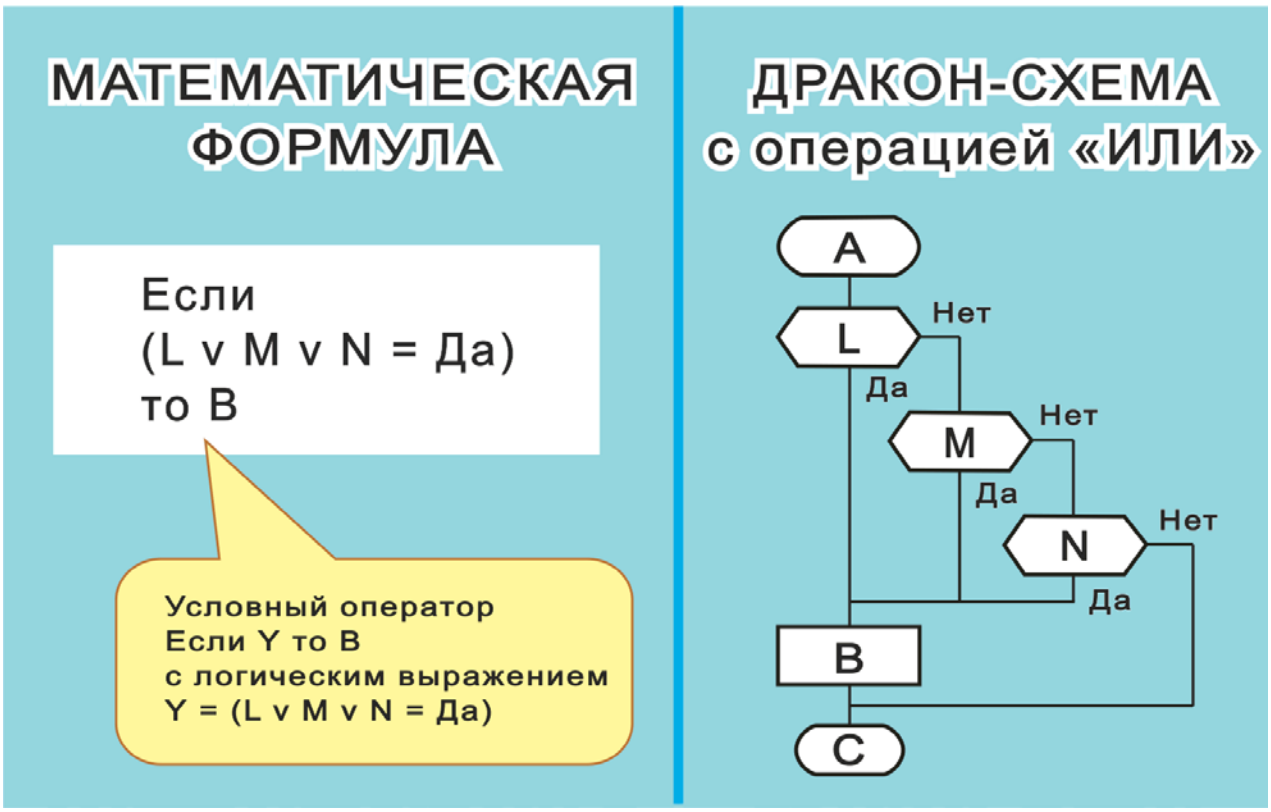


Рис. 74. Математическая формула и дракон-схема с операцией ИЛИ

СРАВНЕНИЕ МАТЕМАТИЧЕСКОЙ ФОРМУЛЫ И ДРАКОН-СХЕМЫ

На рис. 74 приведена математическая формула и эквивалентная ей дракон-схема.

Слева — традиционная текстовая формула, понятная сравнительно узкому кругу специалистов.

Справа — «демократическая» графическая формула, записанная на языке ДРАКОН. Она доступна более широкому кругу работников. Как показывает практика, правая формула понятна даже тем людям, которые испытывают непреодолимые затруднения при работе со сложной левой формулой.

Изучая формулу слева на рис. 74, иногда приходится анализировать довольно сложные детали:

$$X = [L \vee M \vee N = Да] = [(L = Да) \vee (M = Да) \vee (N = Да)]$$

Дракон-схема (рис. 74, справа) хороша тем, что избавляет пользователя от лишней работы. Единственное, что требуется от читателя — уметь отвечать на простые да-нетные вопросы.

На рис. 72 – 74 вместо знака «ИЛИ» использован знак «V». Оба знака имеют одинаковый смысл и обозначают одно и то же. Далее мы будем использовать оба знака.

ЛОГИЧЕСКАЯ ОПЕРАЦИЯ «НЕ»

На рис. 75 слева в иконе Вопрос читаем:

- $\overline{\text{Молоко скисло}}$?

Верхняя линия — знак «Не». Это значит, что верхнюю линию можно удалить и заменить ее на частицу «не»:

- $\overline{\text{Молоко скисло}}$? = Молоко не скисло? (1)

Маршрут, идущий по шампуру, проходит через «Да»:

- $\overline{\text{Молоко скисло}}$? Да.

Учитывая формулу (1), получим:

- Молоко не скисло? Да.
- Пей молоко.

На этом алгоритм кончает работу.

ДВОЙНОЕ ОТРИЦАНИЕ

Иногда бывает, что в одном предложении появляются два отрицания. Такие случаи особенно неприятны, так как их трудно понять. Рассмотрим случай, когда на рис. 75 (слева) бегунок сворачивает вправо через «Нет»:

- $\overline{\text{Молоко скисло}}$? Нет.

Как понимать эту фразу? Для начала избавимся от верхней черты согласно формуле (1). Мы сразу получаем два отрицания:

- Молоко не скисло? Нет.

Первое отрицание «не», второе «Нет». Правило гласит, что два отрицания взаимно уничтожаются. Вычеркиваем «не» и «Нет», получаем, что *молоко скисло.* Уяснив это обстоятельство, можно сообразить, что запись в алгоритме сделана правильно:

- $\overline{\text{Молоко скисло}}$? Нет. *(значит, скисло).*
- Вылей молоко.

Мы рассмотрели пример с логическим отрицанием «Не». Бросается в глаза, что знак отрицания при чтении вызывает трудности и порою сбивает с толку. Желательно от него избавиться, как показано на рис. 76, справа.

Чем различаются две схемы на рис. 76? В схеме справа удалена верхняя черта, а слова «Да» и «Нет» поменялись местами.

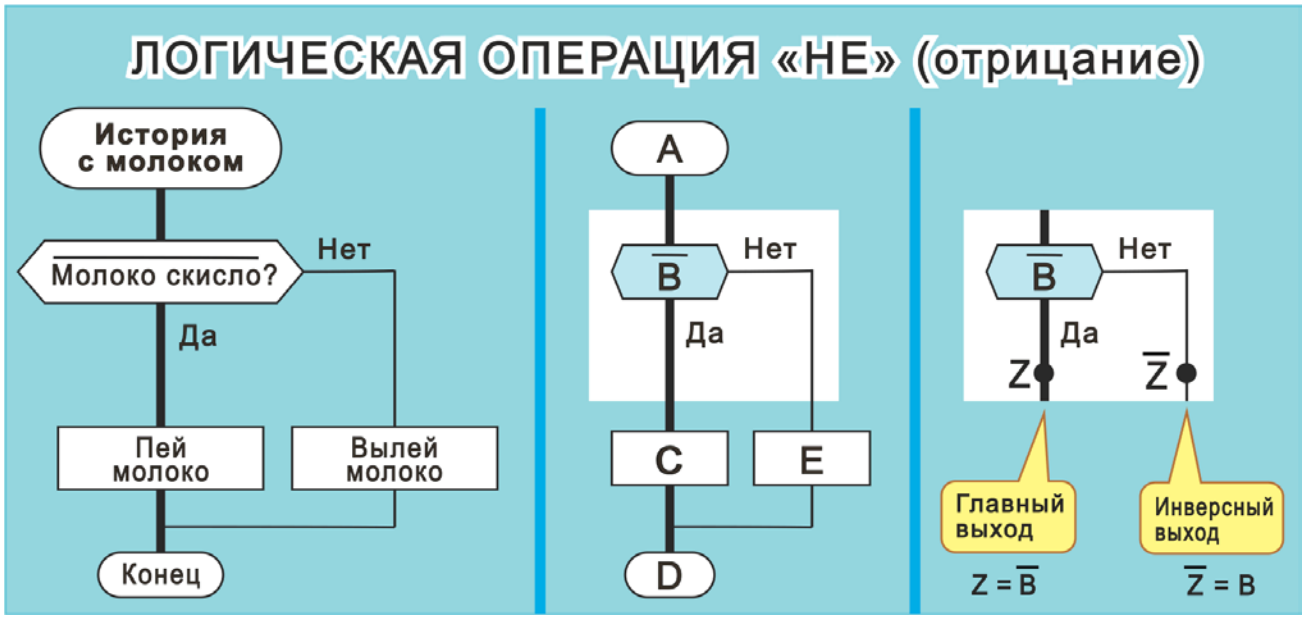


Рис. 75. Алгоритм «История с молоком» с логической операцией «НЕ»

КАК ИЗБАВИТЬСЯ ОТ ЗНАКА ОТРИЦАНИЯ

Чтобы сделать текст алгоритма более понятным, нужно исключить любые отрицания из иконы Вопрос.

Для этого необходимо:

- убрать из иконы Вопрос частицу «НЕ»,
- удалить логический знак отрицания (верхнюю черту или ее заменители),
- убедиться в отсутствии двойного отрицания.

Перечисленные условия почти всегда можно выполнить. Исключением являются случаи, когда в автоматизированных системах управления используются сигналы «Нет нормы» и им подобные.

В последнем случае сочетание

- Нет нормы? Нет.

означает наличие неустранимого двойного отрицания.

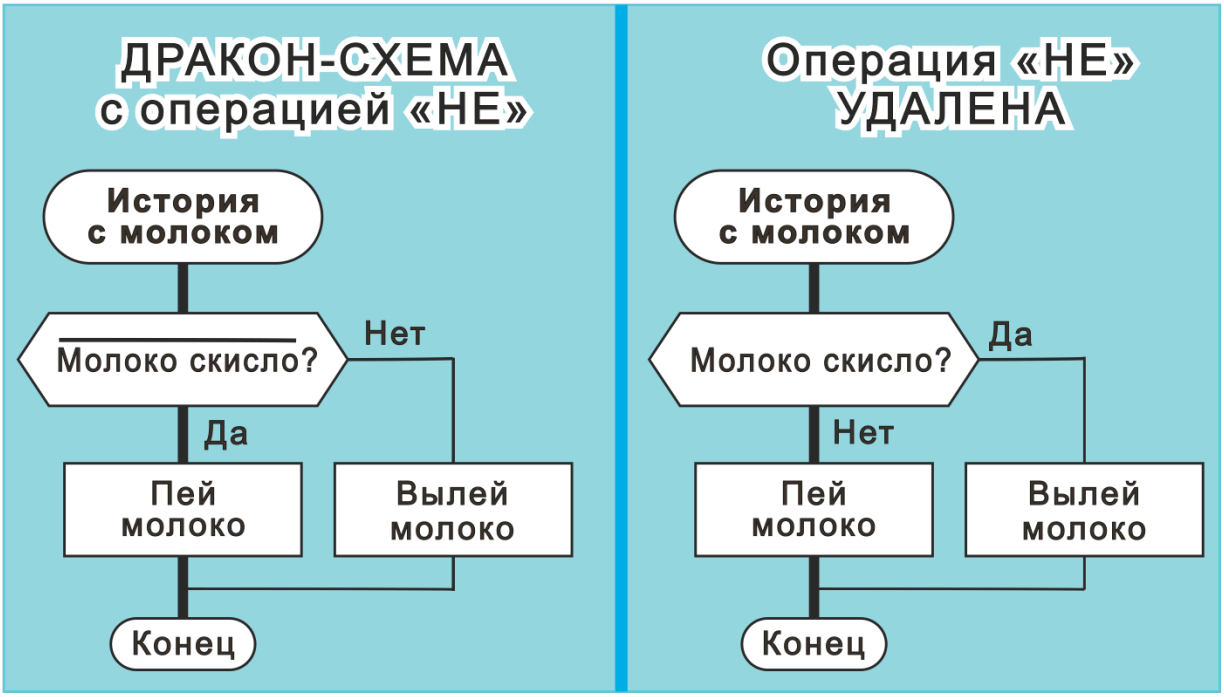


Рис. 76. Алгоритм «История с молоком». Удаление операции «НЕ»

ВЫВОДЫ

1. В алгоритмах логические операции «И», «ИЛИ», «НЕ» играют важную роль.
2. Во многих случаях логические операции вызывают трудности при понимании, что приводит к ошибкам в алгоритмах.
3. Актуальной задачей является создание новой нотации для логических операций, которая облегчает понимание и сокращает число алгоритмических ошибок.
4. Язык ДРАКОН использует новую нотацию для алгоритмической логики с целью сокращения ошибок.

В следующих главах будет дано более подробное описание логических функций «И», «ИЛИ», «НЕ», а также их применение в алгоритмах

Глава 13

ЛОГИЧЕСКАЯ ФУНКЦИЯ «И»

ЛОГИЧЕСКАЯ СХЕМА «И» С ДВУМЯ УСЛОВИЯМИ

Жена попросила повесить картину. Придется забить в стену гвоздь. Что для этого нужно? Молоток и гвоздь. Если есть гвоздь, а молотка нет, ничего не выйдет. И наоборот, если есть молоток без гвоздя, толку не будет. Непременно нужно то и другое. Говорят, что нужно совпадение двух условий: гвоздя и молотка.

Обозначим молоток через M, а гвоздь через G. Тогда логическая функция «И» (конъюнкция) примет вид:

$$Z = M \,\&\, G \qquad (1)$$

где *Z* — логическая функция «И»;

& — логическая операция И.

ТАБЛИЦА ИСТИННОСТИ

Функцию Z (конъюнкцию) можно задать таблицей истинности:

M	G	Z = M & G
0	0	0
0	1	0
1	0	0
1	1	1

О чем говорит 1-я строка таблицы? Если нет молотка (M = 0) и нет гвоздя (G = 0), то логическая функция Z (функция «И») также равна 0.

2-я строка. Если нет молотка (M = 0), а гвоздь есть (G = 1), то Z по-прежнему равна 0.

3-я строка. Если есть молоток (M = 1), а гвоздя нет (G = 0), то Z и в этом случае 0.

4-я строка. Тут все меняется. Если есть оба предмета: и молоток (M = 1), и гвоздь (G = 1), то функция Z (наконец-то!) приобретает значение 1.

ЖЕЛАТЕЛЬНО ИЗБЕГАТЬ СЛОЖНЫХ ВЫРАЖЕНИЙ

Обычно значениями логических переменных считается пара (ИСТИНА, ЛОЖЬ). С эргономической точки зрения, такой подход нельзя признать удачным.

В самом деле, ИСТИНА и ЛОЖЬ — трудные понятия, особенно для начинающих. Использование таких сложных слов в примере о гвоздях (как и в любом другом

конкретном примере) является неоправданным. Оно не содействует быстрому пониманию, а наоборот, затемняет картину и порождает ошибки.

Ненужную сложность следует решительно устранить. Заменим ИСТИНУ и ЛОЖЬ на простые слова: «Да» и «Нет». Смысл этих слов понятен каждому. Даже ребенок знает, что такое Да и Нет.

По этой причине в языке ДРАКОН в логических выражениях используются «Да» и «Нет». А логические функции и переменные рассматриваются как да-нетные вопросы и утверждения.

Логическая переменная величина	Это переменная величина, которая принимает два значения: «Да» и «Нет»

Напомним слова математического классика: «… если мы выберем любой частный вопрос…, то наша процедура… даст нам на него определенный ответ «Да» или «Нет» [7].

Таблицу истинности можно слегка подправить. Заменив 1 и 0 на Да и Нет, получим:

M	G	Z = M & G
Нет	Нет	Нет
Нет	Да	Нет
Да	Нет	Нет
Да	Да	Да

КАК ПРЕДСТАВИТЬ СХЕМУ «И» НА ЯЗЫКЕ ДРАКОН

Выше мы описали конъюнкцию с помощью формулы Z = M & G. На рис. 77 эта формула изображена графически с помощью двух икон Вопрос:

- Есть молоток? Да.
- Есть гвоздь? Да.

Проведя взглядом по шампуру сверху вниз, мы попадаем в точку Z, которая означает, что оба условия выполнены — у нас есть молоток и гвоздь. В точке Z можно поместить команду «Забей гвоздь молотком».

Точка Z, в которую попадает бегунок, пробежавший по шампуру через два выхода «Да», есть графическое изображение функции «И» (конъюнкции).

ТЕКСТОВАЯ И ГРАФИЧЕСКАЯ ЗАПИСЬ ФУНКЦИИ «И»

На рис. 77 показана неполная (недоделанная) дракон-схема. Почему недоделанная? Потому, что у нее отсутствуют выходы «Нет». Недочет исправлен на рис. 78, где показан не только главный выход Z, но и инверсный выход $\overline{Z}$ (читается не зэт), обозначающий логическое отрицание Z.

Инверсный выход очень важен, но мы его объясним позже. А сейчас сосредоточим внимание на главном выходе (рис. 77).

В белом квадрате (на рис. 77) функция «И» изображена в виде графики. А внизу — в виде текста: Z = M & G

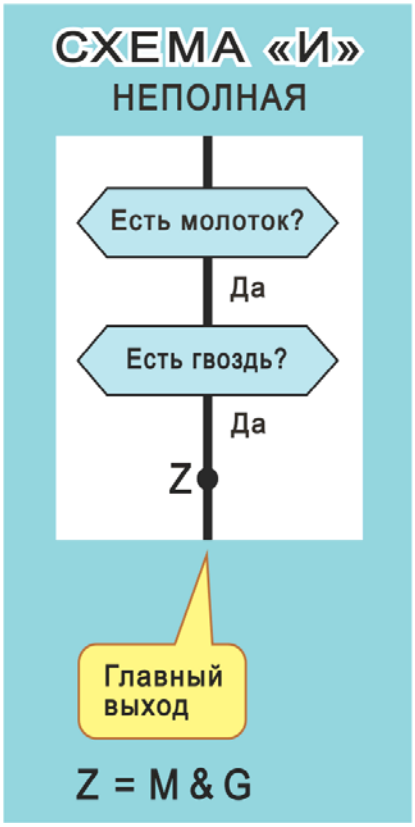


Рис. 77. Главный выход Z находится на шампуре. Z обозначает логическую функцию «И»

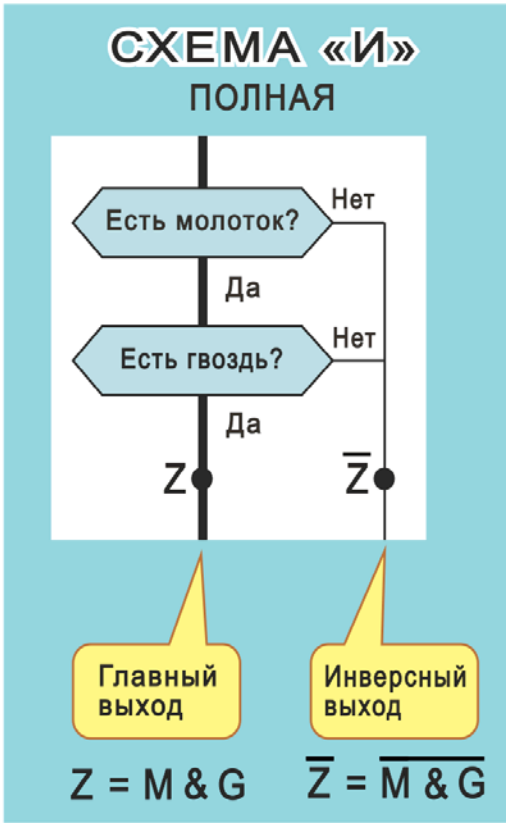


Рис. 78. Справа от шампура находится инверсный выход $\overline{Z}$, который является логическим отрицанием главного выхода Z

ЛОГИЧЕСКАЯ СХЕМА «И» С ТРЕМЯ УСЛОВИЯМИ

Как изменится схема «И» на рис. 77 при увеличении числа условий?

Чтобы повесить картину, нужны не только молоток и гвоздь, но и стремянка. Давайте добавим в схему еще одно условие — икону Вопрос со стремянкой. Обозначим стремянку буквой S. Решение представлено на рис. 79.

Высота дракон-схемы «И» увеличилась на один «этаж». На чертеже изображены уже не две, а три иконы Вопрос (рис. 79).

Нетрудно сообразить, что при добавлении четвертого и последующих условий шампур будет соответственно расти в высоту.

ЛОГИЧЕСКАЯ ФУНКЦИЯ «И»

Мы уже видели примеры логической функции «И»:

- для двух переменных: Z = M & G.
- для трех переменных: Z = M & G & S.

Мы знаем также, что логическая функция «И» имеет два образа, две нотации:

- текстовое представление,
- графическое представление.

Между ними есть существенное отличие. В отличие от текста, где функция «И» имеет только один ответ, ***графическое представление функции «И» имеет два ответа (выхода): главный и инверсный*** (см. рис. 68, 78, 80).

Логическая функция «И»

- Это функция, которая принимает значение «ДА», если все логические переменные имеют значение «Да».
- В остальных случаях функция принимает значение «Нет».

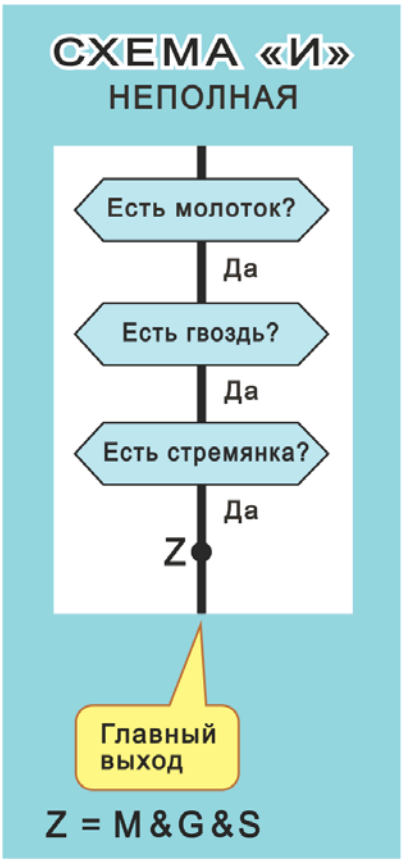


Рис. 79. Добавление еще одного условия в логическую схему «И» увеличивает высоту шампура на один этаж

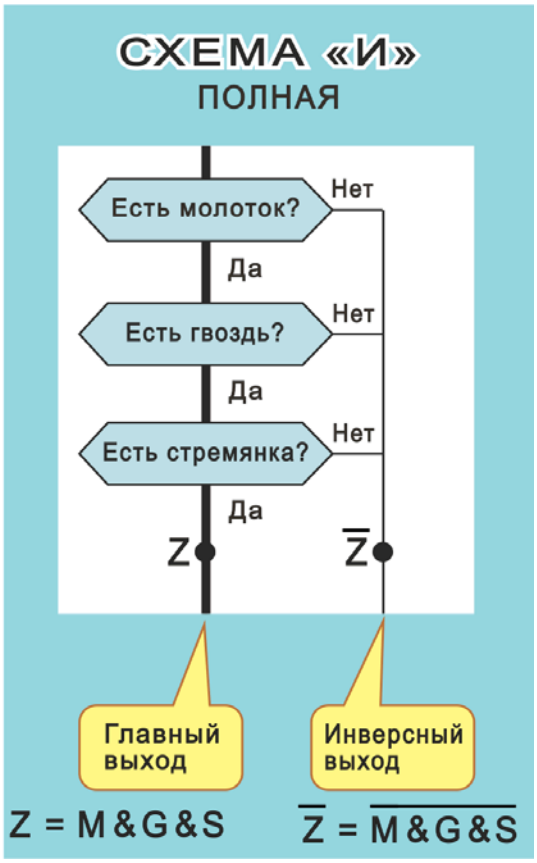


Рис. 80. Инверсный выход $\overline{Z}$ является логическим отрицанием главного выхода Z при любом числе условий

КАК ЧИТАТЬ АБСТРАКТНЫЕ ДРАКОН-СХЕМЫ

Чтение абстрактных схем имеет особенности. Чтобы облегчить чтение, можно включить воображение и использовать полезные подсказки.

Обратимся снова к рис. 77 и 78. Заменим содержательные надписи в иконах Вопрос на мнемонические буквы. (Молоток на букву M, гвоздь на букву G). Результат показан на рис. 81.

Как читать этот рисунок? Очень просто. Когда в иконе Вопрос видим букву M, читаем:

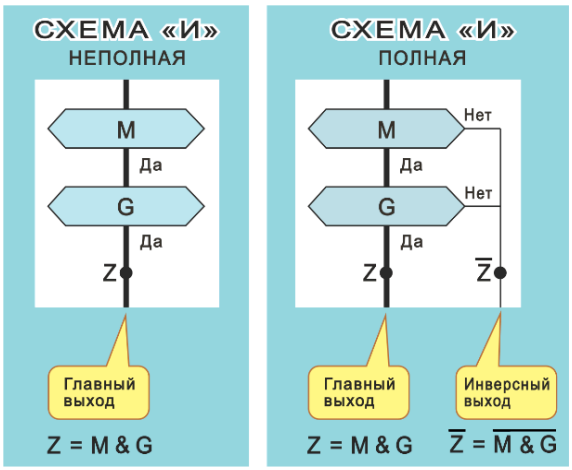


Рис. 81. Логическая схема «И» с двумя условиями

«Есть ли у нас условие M?» При этом подразумеваем: Есть ли у нас молоток? Если есть, выходим из иконы Вопрос вниз через «Да».

Когда видим букву G, говорим: «Есть ли условие G?» Подразумеваем: Есть ли гвоздь? Если есть, опять выходим через «Да».

Есть и другой способ. Забудем о том, что M и G обозначают гвоздь и молоток. Будем считать, что это две абстрактные буквы, которые ничего не обозначают.

В таком случае процесс чтения тоже можно облегчить. Давайте предположим, что M и G — это два предмета, которые можно положить на стол, а можно убрать со стола.

Когда в иконе Вопрос мы видим букву M, читаем: «Есть ли у нас предмет M?» При этом подразумеваем: Есть ли на нашем воображаемом столе предмет M? Если есть, выходим из иконы Вопрос вниз через «Да». Если же нет (стол пустой), выходим через выход «Нет».

АБСТРАКТНАЯ СХЕМА С ТРЕМЯ УСЛОВИЯМИ

В предыдущем параграфе мы рассмотрели правила чтения схемы «И» с двумя логическими переменными. Повторяя рассуждения, можно распространить данный способ на схему «И» с тремя переменными (рис. 82).

Общий принцип остался без изменения. Мы мысленно предполагаем, что M, G и S — это три предмета (например, три книги), которые можно выложить на стол, а можно убрать со стола.

Если предмет находится на столе, бегунок выходит через выход «Да». Если же предмет отсутствует (на столе ничего нет), бегунок выходит через «Нет».

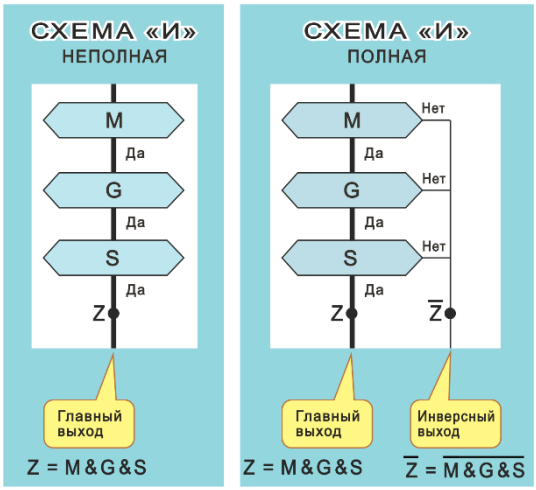


Рис. 82 Логическая схема «И» с тремя условиями

ЗНАК ВОПРОСА МОЖНО УБРАТЬ

В данной книге приняты следующие соглашения относительно иконы Вопрос.

- После содержательных предложений («Есть молоток?» или «Есть гвоздь?») обязательно ставится вопросительный знак (рис. 77 – 80).
- Если же в иконе Вопрос записана абстрактная буква (M, G, S и т. д.), знак вопроса опускается (рис. 81, 82), но, конечно, всегда подразумевается.

ЛОГИЧЕСКАЯ СВЯЗКА «И»

Знак «И» имеет еще одно название — *логическая связка И*.

Почему связка? Потому что знак «И» (или «&») связывает между собой логические переменные.

Например, на рис. 82 (внизу) описана логическая функция Z = M & G & S. В этой формуле логическая связка & повторяется два раза. Две связки & связывают между собой три логические переменные M, G, S.

ИНВЕРСНЫЙ ВЫХОД ИЗ СХЕМЫ «И»

Инверсный значит обратный, противоположный.

Логическая функция «И» (в графическом представлении) имеет два выхода (рис. 83):

- главный Z,
- инверсный $\overline{Z}$

Как правильно читать такую схему? Прежде всего, следует сконцентрировать внимание на главном выходе и забыть про инверсный. Это можно сделать, закрыв инверсный выход рукой или листом бумаги. В результате получим левую часть рисунка 83.

Перед нами простая и понятная схема — мы видим совпадение трех условий, нанизанных на шампур.

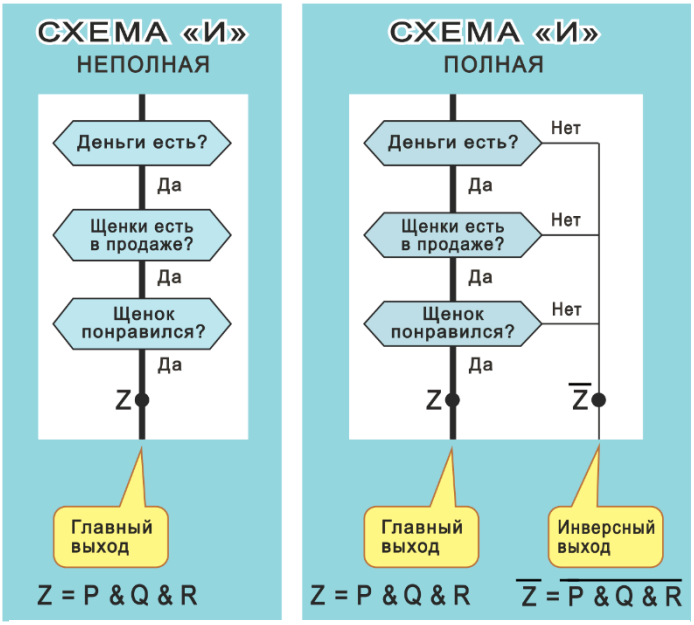


Рис. 83. Логическая схема «И» с тремя условиями

После этого можно перейти к изучению инверсного выхода. Начнем с понятия «фрагмент».

ЛОГИЧЕСКИЙ ФРАГМЕНТ ДРАКОН-СХЕМЫ

Фрагмент дракон-схемы называется логическим, если он имеет один вход, два выхода и содержит только иконы Вопрос, соединенные между собой.

Примеры логических фрагментов показаны на рис. 69, 73, 77 – 83.

Как удостовериться, что некоторая схема является логическим фрагментом? Проверьте, что схема имеет:

- только один вертикальный вход сверху,
- только два вертикальных выхода внизу,
- в схеме используются только иконы Вопрос и соединительные линии (и больше ничего).

СТАНДАРТНАЯ И НЕСТАНДАРТНАЯ ЛОГИЧЕСКАЯ СХЕМА «И»

Стандартная логическая схема «И» для *N* логических переменных — это схема, содержащая *N* икон Вопрос, которые:

- расположены на одной вертикали;
- в каждой иконе Вопрос содержится одна переменная.

Стандартная схема «И» для трех логических переменных представлена на рис. 84, слева.

Нестандартная логическая схема «И» для *N* логических переменных — схема, полученная с помощью рокировки стандартной схемы «И». На рис. 84 (справа) дан пример нестандартной схемы для трех переменных.

Стандартная и нестандартная схемы «И» равносильны. Они описывают один и тот же алгоритм.

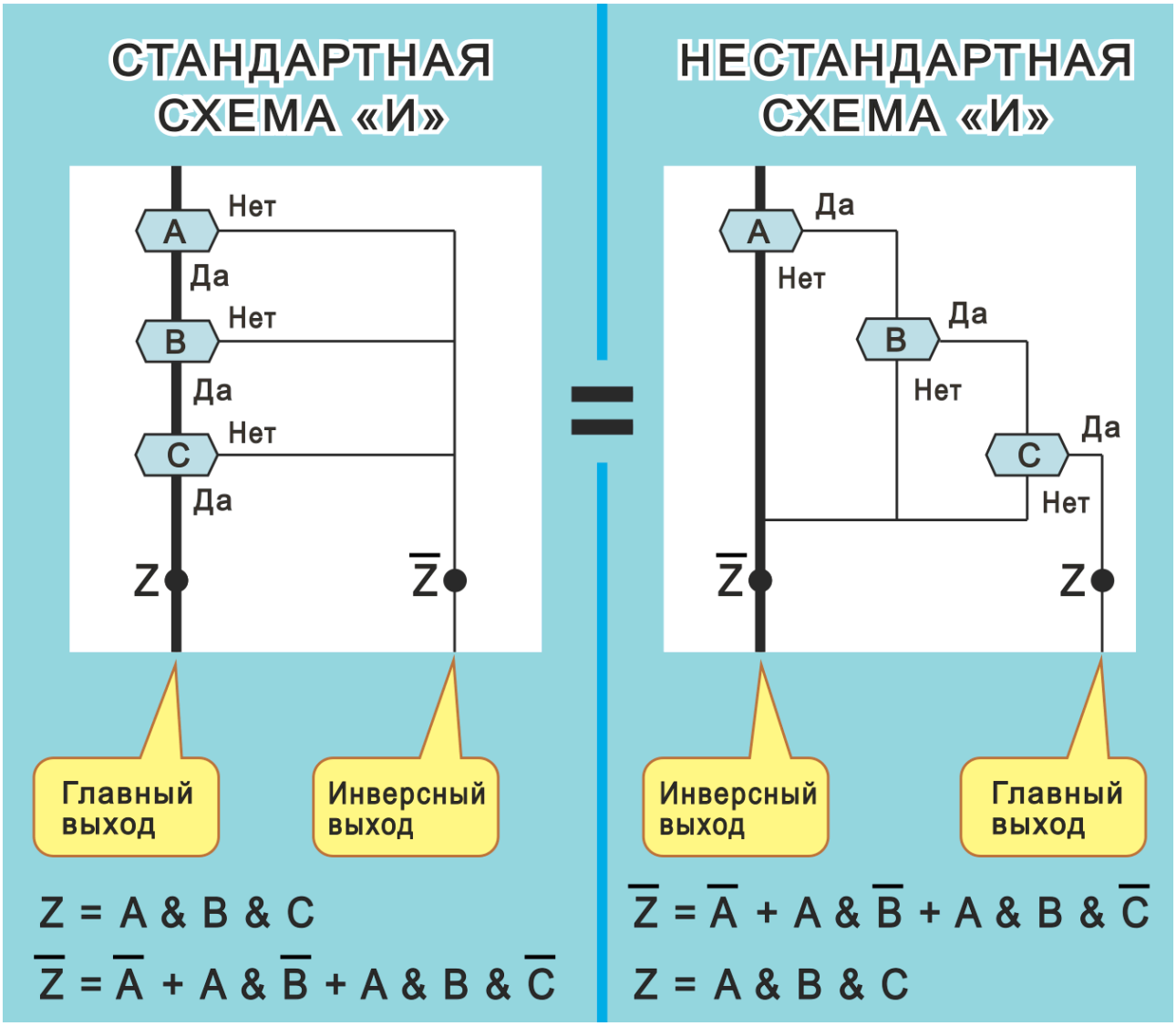


Рис. 84. Стандартная (слева) и нестандартная (справа) схема «И»

ЧЕМ РАЗЛИЧАЮТСЯ СТАНДАРТНАЯ И НЕСТАНДАРТНАЯ СХЕМЫ «И»

Отличия показаны на рис. 84 и состоят из трех пунктов.

1. Как размещены иконы Вопрос?
   - В стандартной схеме они лежат на шампуре (рис. 84, слева).
   - В нестандартной — расположены лесенкой (рис. 84, справа).
2. Где находится главный выход?
   - В стандартной схеме он лежит на шампуре.
   - В нестандартной — расположен справа.
3. Где находится инверсный выход?
   Он зеркально меняется местами с главным выходом.
   - В стандартной схеме он находится справа.
   - В нестандартной — лежит на шампуре

ТЕОРЕМА ФРАГМЕНТА

Теорема. Инверсный выход $\bar{Z}$ фрагмента является логическим отрицанием главного выхода Z.

Доказательство теоремы предоставляем читателям.

ВЫВОД ФОРМУЛЫ ДЛЯ ИНВЕРСНОГО ВЫХОДА

Не читайте этот параграф! Он предназначен только для любителей математики.

На рис. 84 (внизу слева) представлены две формулы, которые требуют пояснения. В них мы заменили классический знак V на более удобный знак + (знак логического сложения). Оба знака (V и +) обозначают операцию «ИЛИ», которая называется также логическим сложением.

Поместим интересующие нас формулы в рамку.

$$Z = A \,\&\, B \,\&\, C \tag{1}$$

$$\bar{Z} = \bar{A} + A \,\&\, \bar{B} + A \,\&\, B \,\&\, \bar{C} \tag{2}$$

Цель состоит в следующем. Опираясь на теорему фрагмента, необходимо доказать, что, взяв отрицание от формулы (1), мы получим формулу (2), то есть:

$$\overline{A \,\&\, B \,\&\, C} = \bar{A} + A \,\&\, \bar{B} + A \,\&\, B \,\&\, \bar{C} \tag{3}$$

Таким образом, мы должны доказать равенство (3). Прежде всего, учтем

- Закон де Моргана $\overline{x \,\&\, y} = \bar{x} + \bar{y}$ (4)

С его помощью упростим левую часть уравнения (3)

$$\overline{A \,\&\, B \,\&\, C} = \bar{A} + \bar{B} + \bar{C} \tag{5}$$

И получим главную формулу, которую нужно доказать

$$\bar{A} + \bar{B} + \bar{C} = \bar{A} + A \,\&\, \bar{B} + A \,\&\, B \,\&\, \bar{C} \tag{6}$$

Далее будем опираться на хорошо известные соотношения алгебры логики [8].

- Закон дистрибутивности $x + y \,\&\, w = (x + y) \,\&\, (x + w)$ (7)
- Закон исключенного третьего $x + \bar{x} = 1$ (8)
- Тождество $x \,\&\, 1 = x$ (9)

Упростим первые два члена правой части уравнения (6). Учитывая соотношения (7 – 9), выполним преобразования

$$\bar{A} + A \,\&\, \bar{B} = (\bar{A} + A) \,\&\, (\bar{A} + \bar{B}) = 1 \,\&\, (\bar{A} + \bar{B}) = \bar{A} + \bar{B}$$

Отсюда следует, что

$$\bar{A} + A \,\&\, \bar{B} = \bar{A} + \bar{B} \tag{10}$$

Учитывая соотношение (10), уравнение (6) принимает вид

$$\bar{A} + \bar{B} + \bar{C} = \bar{A} + \bar{B} + A \,\&\, B \,\&\, \bar{C} \tag{11}$$

Поскольку уравнения (3), (6) и (11) равносильны, задача сводится к необходимости доказать равенство (11). Принимая во внимание (4), (7), (8), (9), запишем

$$\bar{A} + \bar{B} + A \,\&\, B \,\&\, \bar{C} = \overline{A \,\&\, B} + A \,\&\, B \,\&\, \bar{C} =$$
$$= (\overline{A \,\&\, B} + A \,\&\, B) \,\&\, (\overline{A \,\&\, B} + \bar{C}) =$$
$$= 1 \,\&\, (\overline{A \,\&\, B} + \bar{C}) = \overline{A \,\&\, B} + \bar{C} = \bar{A} + \bar{B} + \bar{C}$$

Отсюда вытекает, что равенство (11) доказано и, следовательно, наша задача полностью решена. Таким образом, обе формулы на рис. 84 (внизу слева), описывающие главный и инверсный выход логической схемы «И», являются доказанными и правильными.

РАССТАНОВКА «ДА» И «НЕТ» НА ВЫХОДАХ ИКОНЫ ВОПРОС

Глядя на рис. 84 (слева), можно заметить, что нижние выходы у трех икон Вопрос помечены словом «Да». Является ли такой подход обязательным? Можно ли изменить расстановку «Да» и «Нет» и поменять их местами?

Да, можно. Чтобы убедиться в этом, на рис. 85 (слева) показан другой вариант — возле иконы B мы изменили расстановку «Да» и «Нет» на противоположную.

На что повлияло такое изменение?

Во-первых, на нестандартную схему «И», где возле иконы B также изменилась расстановка «Да» и «Нет» (рис. 85, справа). Во-вторых, изменение затронуло формулу главного и инверсного выхода (над буквой B либо появился, либо исчез знак отрицания).

Подведем итоги. Пользователь имеет право расставлять слова «Да» и «Нет» по своему усмотрению. Принцип построения логической схемы «И» (как стандартной, так и не стандартной) при этом остается неизменным. Разумеется, нестандартная схема всегда должна выводиться из стандартной с помощью рокировки.

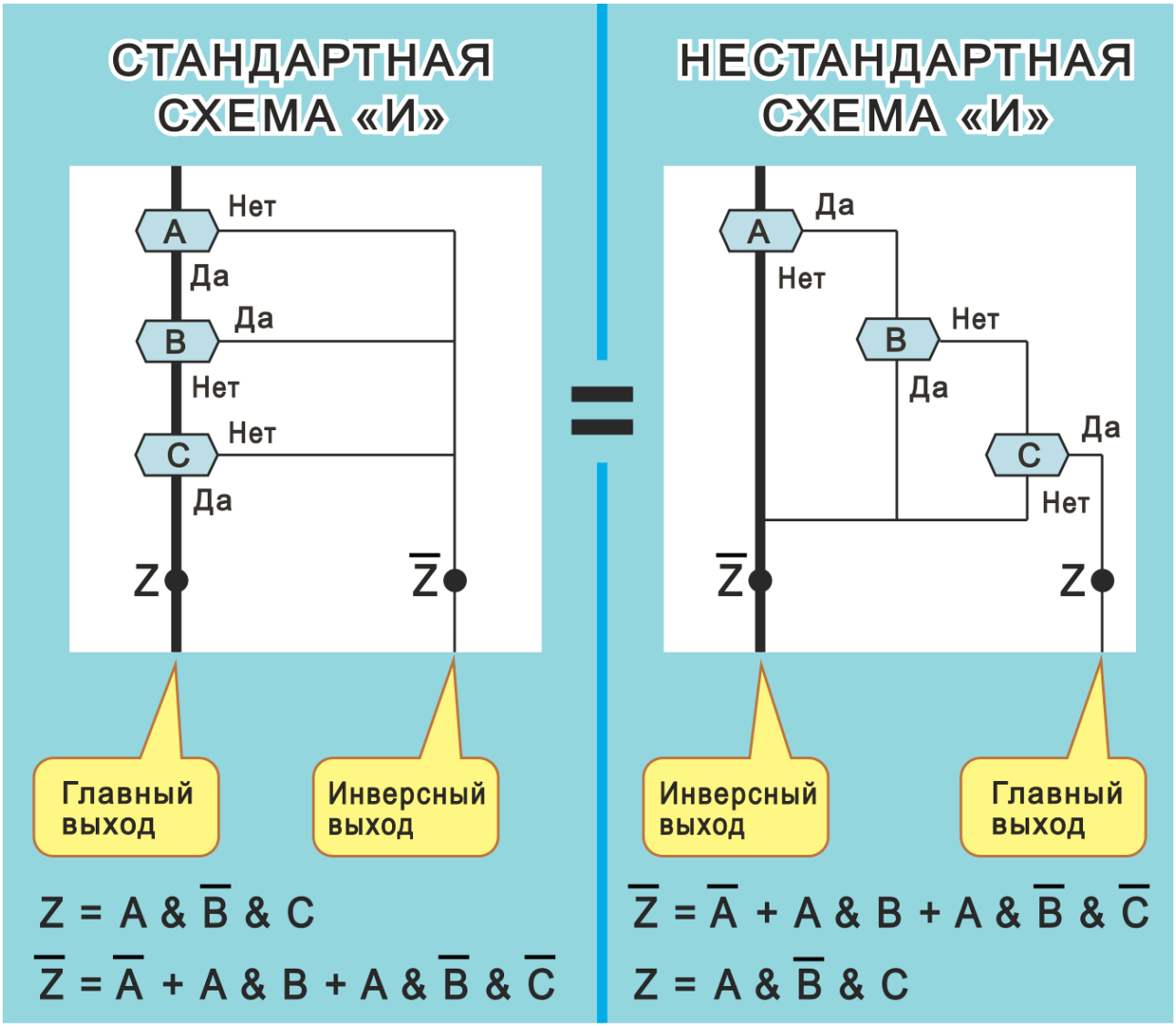


Рис. 85. Стандартная (слева) и нестандартная (справа) схема «И». Произвольное расположение «Да» и «Нет» на выходе икон Вопрос

ВЫВОДЫ

1. В языке ДРАКОН логическая переменная величина принимает два значения: «Да» и «Нет» (1 и 0).
2. Логическая функция «И» принимает значение «Да», если все переменные имеют значение «Да». В остальных случаях она принимает значение «Нет».
3. Логическая функция «И» имеет две нотации:
    - текстовое представление,
    - графическое представление.
4. Графическое представление функции «И» имеет два выхода: главный Z и инверсный $\overline{Z}$.
5. Фрагмент дракон-схемы называется логическим, если он имеет один вход, два выхода и содержит только иконы Вопрос.
6. Графическая схема «И» бывает стандартная и нестандартная.
7. Стандартная схема «И» содержит несколько икон Вопрос, которые:
    - расположены на одной вертикали;
    - в каждой иконе Вопрос содержится одна переменная.
8. Нестандартная схема «И» образуется из стандартной с помощью рокировки.
9. В стандартной схеме «И» иконы Вопрос лежат на шампуре. В нестандартной они расположены лесенкой.
10. В стандартной схеме главный выход лежит на шампуре. В нестандартной он расположен справа.
11. Расстановка слов «Да» и «Нет» на выходах икон Вопрос может быть произвольной.

Глава 14

ЛОГИЧЕСКАЯ ФУНКЦИЯ «ИЛИ»

ЛОГИЧЕСКАЯ СХЕМА «ИЛИ» С ДВУМЯ УСЛОВИЯМИ

Внезапно погас свет, все испугались, засуетились и в суматохе задели китайскую вазу. Ваза упала и разбилась.

Кто разбил вазу? Или Федя, или Коля, или оба вместе. Обозначим Федю через F, а Колю через K. Тогда логическая функция «ИЛИ» (дизъюнкция) примет вид:

$$Z = F \vee K \qquad (1)$$

где *Z* — логическая функция «ИЛИ»;
v — логическая операция «ИЛИ».

ТАБЛИЦА ИСТИННОСТИ

Функцию Z (дизъюнкцию) можно задать таблицей истинности:

F	K	Z = F v K
0	0	0
0	1	1
1	0	1
1	1	1

О чем говорит 1-я строка таблицы? Если Федя и Коля не трогали вазу (F= 0, K = 0), то ваза цела и логическая функция Z (функция «ИЛИ») равна 0.

2-я строка. Если Федя не виноват (F = 0), а вазу уронил Коля (K = 1), то функция Z равна 1.

3-я строка. Если вазу толкнул Федя (F = 1), а Коля ни при чем (K = 0), то Z и в этом случае равна 1.

4-я строка. Если же Федя и Коля вдвоем разбили вазу, то функция Z по-прежнему имеет значение 1.

Таблицу истинности можно слегка подправить. Заменив 1 и 0 на Да и Нет, получим:

F	K	Z = F v K
Нет	Нет	Нет
Нет	Да	Да
Да	Нет	Да
Да	Да	Да

КАК ПРЕДСТАВИТЬ СХЕМУ «ИЛИ» НА ЯЗЫКЕ ДРАКОН

Выше мы описали дизъюнкцию с помощью формулы Z = F v K. На рис. 86 эта формула изображена графически с помощью двух икон Вопрос:

- Федя разбил вазу? Да. А если Нет, то
- Коля разбил вазу? Да.

Верхняя икона Вопрос организует развилку на «Да» и «Нет».

Обратите внимание: у нижней иконы Вопрос отсутствует выход «Нет» (так как он никак не влияет на главный выход).

Точка Z, в которую попадает бегунок, пробежавший по любому из двух маршрутов, есть графическое изображение функции «ИЛИ» (для случая, когда ваза разбилась).

ТЕКСТОВАЯ И ГРАФИЧЕСКАЯ ЗАПИСЬ ФУНКЦИИ «ИЛИ»

На рис. 86 показана неполная дракон-схема, у которой удален нижний выход «Нет». Изъян исправлен на рис. 87, где показан не только главный, но и инверсный выход.

Рис. 86 позволяет упростить схему и облегчить понимание работы главного выхода.

В белом квадрате (на рис. 87) функция «ИЛИ» изображена в виде графики. А внизу — в виде текста.

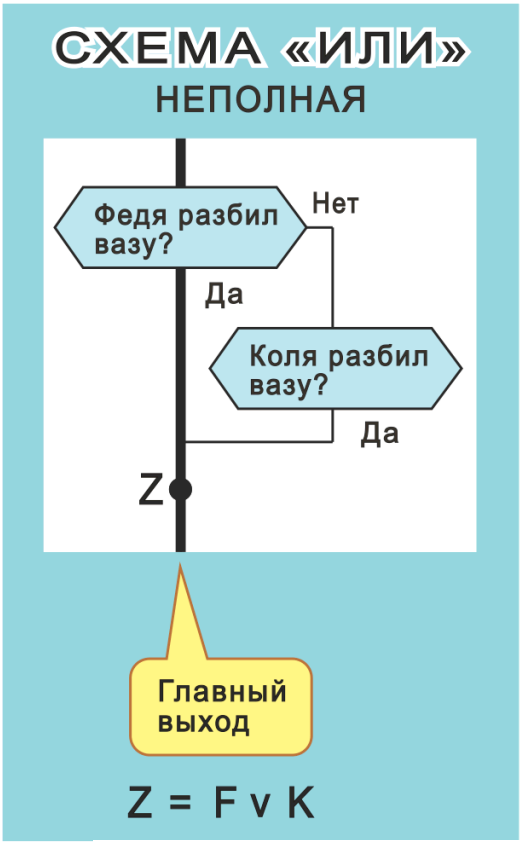


Рис. 86. Главный выход Z находится на шампуре. Z обозначает логическую функцию «ИЛИ»

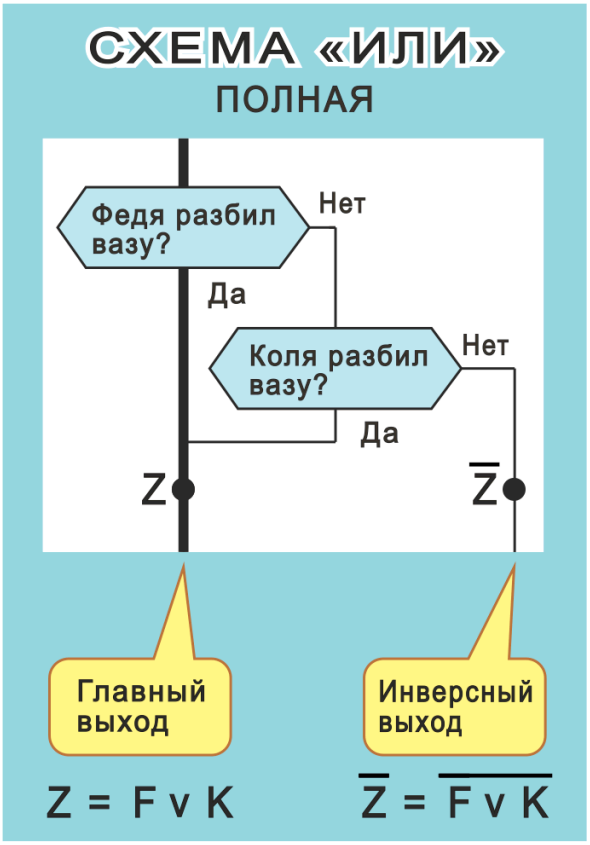


Рис. 87. Справа от шампура $\overline{\ \ }$ находится инверсный выход $\overline{Z}$, который является логическим отрицанием главного выхода Z

ЛОГИЧЕСКАЯ СХЕМА «ИЛИ» С ТРЕМЯ УСЛОВИЯМИ

Предположим, что вазу разбили не два, а три человека: Федя, Коля и Таня. Обозначим Таню буквой Т. Схема «ИЛИ» с тремя условиями показана на рис. 88 и 89.

Схема «ИЛИ» по-прежнему изображена лесенкой, но ее ширина увеличилась на одну «ступеньку». На чертеже представлены три иконы Вопрос.

Нетрудно сообразить, что при добавлении четвертого и последующих условий схема будет неуклонно расти в ширину.

ЛОГИЧЕСКАЯ ФУНКЦИЯ «ИЛИ»

Мы рассмотрели примеры логической функции «ИЛИ»:

- для двух переменных: Z = F v K.
- для трех переменных: Z = F v K v T.

Мы знаем также, что функция «ИЛИ» имеет две нотации: текстовую и графическую. В отличие от текста, где функция «ИЛИ» имеет только один ответ, ***графическое представление функции «ИЛИ» имеет два ответа (выхода): главный и инверсный*** (см. рис. 68, 78, 80).

Логическая функция «ИЛИ»

- Это функция, которая принимает значение «Да», если хотя бы одна логическая переменная имеет значение «Да».
- Если же все переменные принимают значение «Нет», функция также обращается в «Нет».

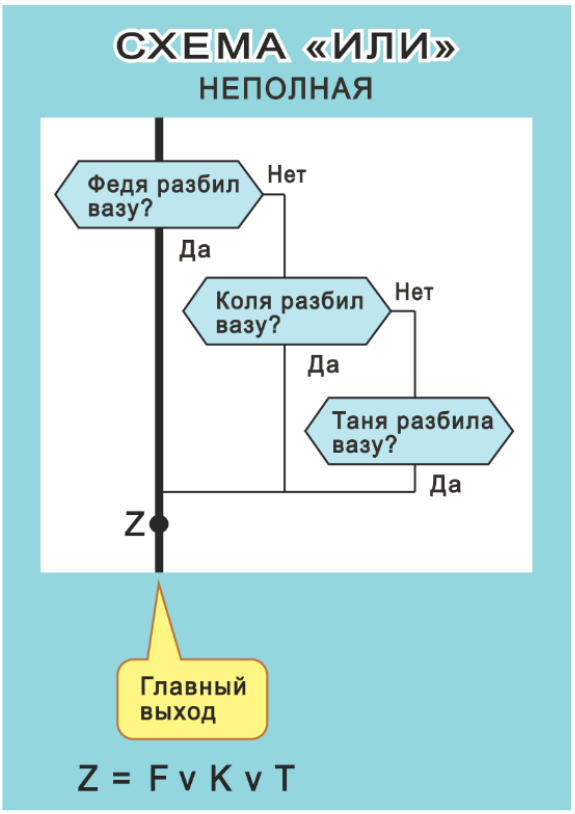


Рис. 88. Добавление еще одного условия в логическую схему «ИЛИ» увеличивает ширину «лестницы»

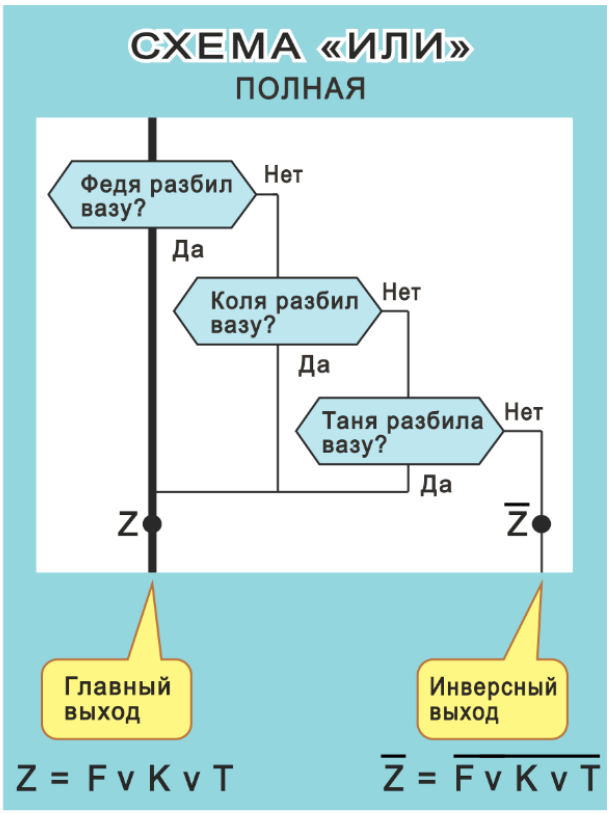


Рис. 89. Инверсный выход $\overline{Z}$ является логическим отрицанием главного выхода Z при любом числе условий

ЛОГИЧЕСКАЯ СВЯЗКА «ИЛИ»

Знак «ИЛИ» имеет еще одно название — *логическая связка ИЛИ*.

Почему связка? Потому что знак «ИЛИ» (или «V») связывает между собой логические переменные.

Например, на рис. 89 (внизу) описана логическая функция Z = F v K v T. В этой формуле логическая связка V повторяется два раза. Она связывает между собой три логические переменные F, K, T.

СТАНДАРТНАЯ И НЕСТАНДАРТНАЯ ЛОГИЧЕСКАЯ СХЕМА «ИЛИ»

Стандартная логическая схема «ИЛИ» для *N* логических переменных — это схема, содержащая *N* икон Вопрос, которые:

- расположены лесенкой;
- в каждой иконе Вопрос содержится одна переменная.

Стандартная схема «ИЛИ» для трех логических переменных представлена на рис. 90, слева.

Нестандартная логическая схема «ИЛИ» для *N* логических переменных — схема, полученная с помощью рокировки стандартной схемы «ИЛИ». На рис. 84 (справа) показана нестандартная схемы для случая трех переменных.

Стандартная и нестандартная схемы «И» равносильны. Они описывают один и тот же алгоритм.

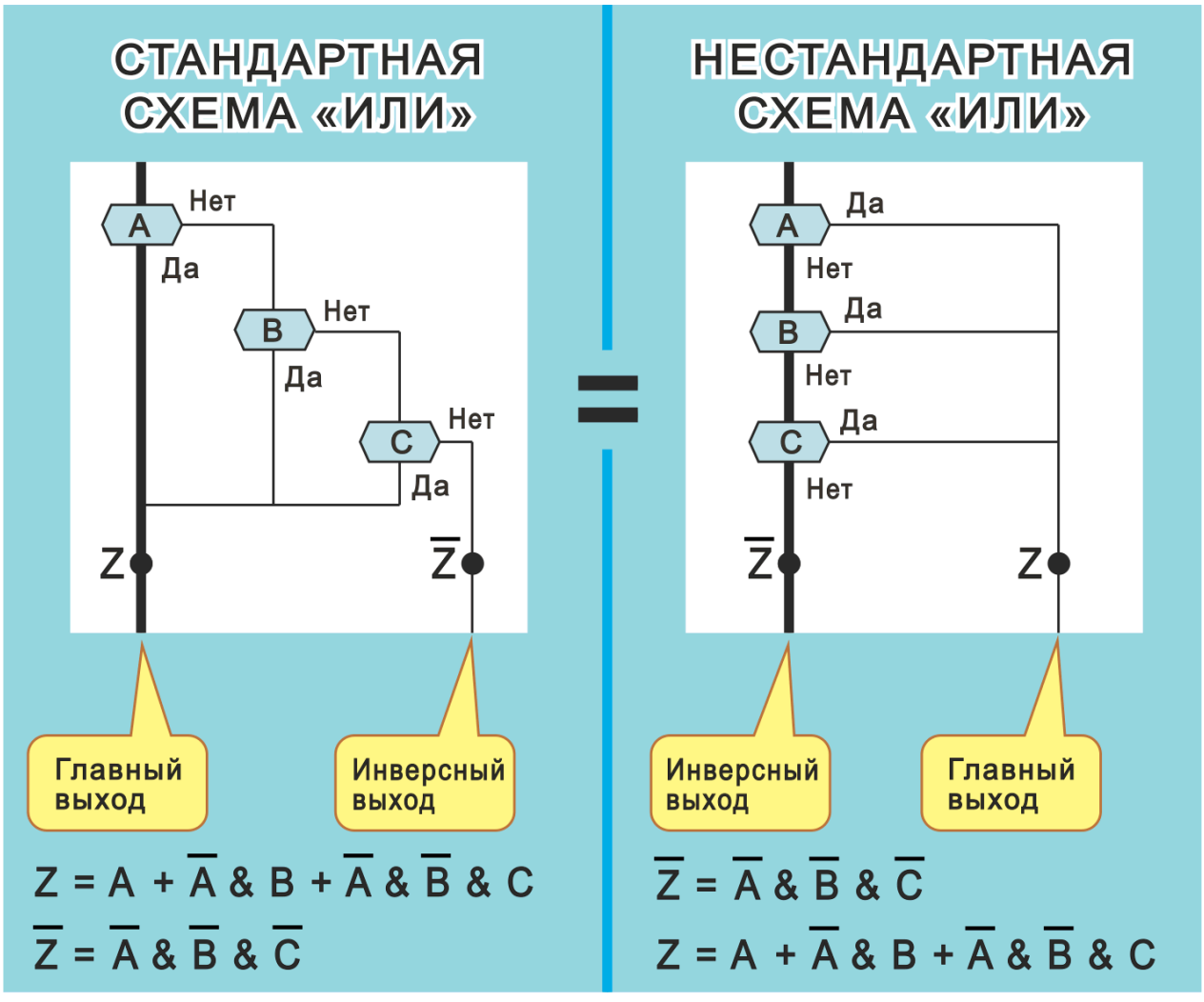


Рис. 90. Стандартная (слева) и нестандартная (справа) схема «ИЛИ»

ЧЕМ РАЗЛИЧАЮТСЯ СТАНДАРТНАЯ И НЕСТАНДАРТНАЯ СХЕМЫ «ИЛИ»

Отличия показаны на рис. 90 и сводятся к трем пунктам.

1. Как размещены иконы Вопрос?
   - В стандартной схеме они расположены лесенкой (рис. 90, слева).
   - В нестандартной — лежат на шампуре (рис. 90, справа).
2. Где находится главный выход?
   - В стандартной схеме он лежит на шампуре.
   - В нестандартной — расположен справа.
3. Где находится инверсный выход?
   Он зеркально меняется местами с главным выходом.
   - В стандартной схеме он находится справа.
   - В нестандартной — лежит на шампуре

КАК ПРЕОБРАЗОВАТЬ ФОРМУЛУ МАРШРУТА В КОНЪЮНКТИВНУЮ ФОРМУ

На рис. 91 в качестве примера показан маршрут, идущий по шампуру. Маршрут проходит по трем иконам через выход «Нет», а по иконам C и D — через выход «Да». Формула маршрута имеет вид

$$\text{A нет B нет C да D да E нет} \qquad (1)$$

Наша цель — преобразовать формулу маршрута (1) в конъюнктивную форму. Прежде всего, нужно выполнить два шага.

Шаг 1. Если в формуле маршрута после латинской буквы стоит «нет», замените букву на ее отрицание, а слово «нет» удалите. Пример: “A нет” превратится в $\overline{A}$.

Шаг 2. Если после буквы стоит «да», сохраните букву, а слово «да» удалите. Пример: «C да» превратится в C.

Применив эти действия к примеру (1), получим

$$\overline{A}\ \overline{B}\ C\ D\ \overline{E} \qquad (2)$$

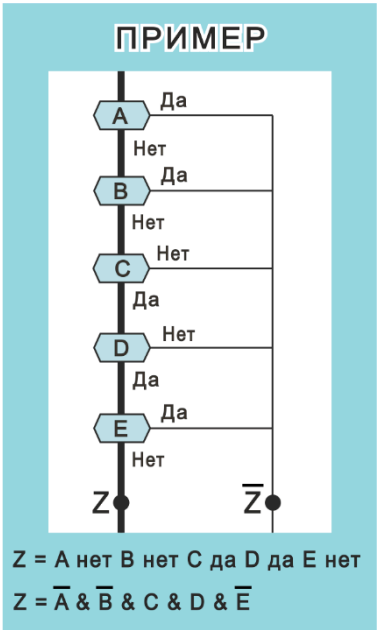


Рис. 91. Маршрут, идущий по шампуру

Определение. Литералом называют логическую переменную или её логическое отрицание.

В нашем примере литералами являются: $\overline{A}, \overline{B}, C, D, \overline{E}$.

Нас интересует маршрут, идущий по шампуру на рис. 91. Чтобы описать маршрут, между литералами в формуле (2) нужно вставить знаки конъюнкции &. Получим

$$\overline{A} \,\&\, \overline{B} \,\&\, C \,\&\, D \,\&\, \overline{E} \qquad (3)$$

Определение. Конъюнкт — это конъюнкция литералов [9].

Нетрудно сообразить, что формула (3) является конъюнктом или (что одно и то же) конъюнктивной формой.

В данном параграфе на частном примере мы показали, что формула маршрута (1) может быть преобразована в конъюнктивную форму (3). Это значит, что маршрут может быть математически описан двумя эквивалентными способами:

- с помощью формулы маршрута (1),
- с помощью конъюнктивной формы (3).

$$Z = A \text{ нет } B \text{ нет } C \text{ да } D \text{ да } E \text{ нет } = \overline{A} \,\&\, \overline{B} \,\&\, C \,\&\, D \,\&\, \overline{E}$$

Этот вывод справедлив и в общем случае (для любого маршрута). Отсюда следует, что формулы маршрута можно описывать с помощью конъюнктов.

ФОРМУЛЫ МАРШРУТОВ В СТАНДАРТНОЙ СХЕМЕ «ИЛИ»

Схема ИЛИ на рис. 92 имеет четыре маршрута. Для каждого из них можно построить формулу маршрута в виде конъюнкта.

Первый маршрут идет по шампуру и имеет формулу “A да”. Слово «да» означает, что буква A берется без отрицания, как показано в первой выноске.

Второй маршрут идет правее с формулой “A нет B да”. Преобразовав в конъюнкт, получим $\overline{A} \,\&\, B$.

Третий маршрут идет еще правее и имеет формулу “A нет B нет C да”. После преобразования в конъюнкт, получим $\overline{A} \,\&\, \overline{B} \,\&\, C$.

Наконец, четвертый маршрут (крайний справа) имеет формулу “A нет B нет C нет” и превращается в конъюнкт: $\overline{A} \,\&\, \overline{B} \,\&\, \overline{C}$.

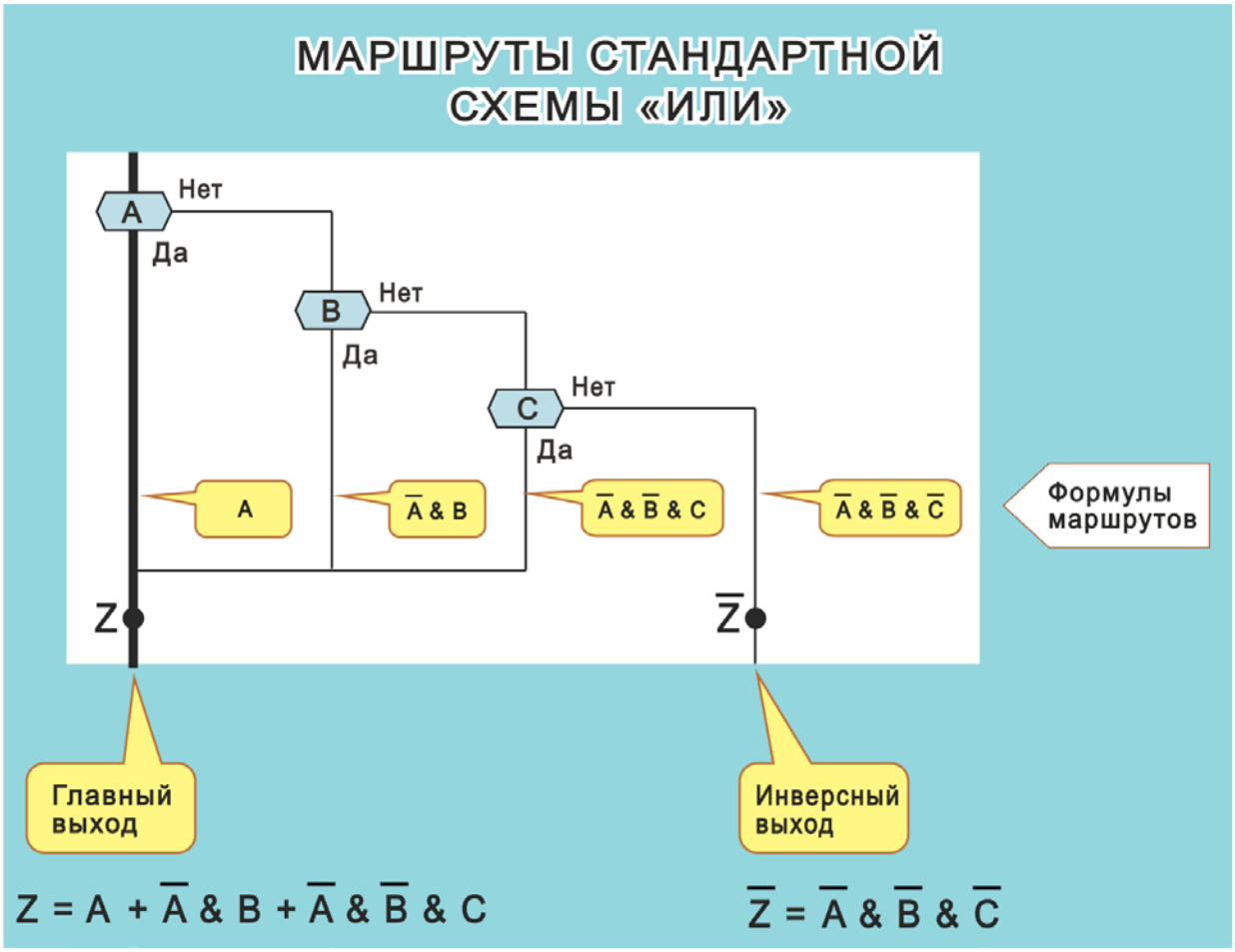


Рис. 92. На схеме «ИЛИ» показаны четыре формулы маршрутов (в четырех выносках). Каждая формула — это конъюнкт.

ФОРМУЛЫ ГЛАВНОГО И ИНВЕРСНОГО ВЫХОДОВ СХЕМЫ «ИЛИ» В ДИЗЪЮНКТИВНОЙ НОРМАЛЬНОЙ ФОРМЕ

Рассмотрим три маршрута на рис. 92, которые проходят через выход «Да» икон A, B, C. Эти три маршрута сливаются в одной точке и играют важную роль — они формируют главный выход Z.

Сказанное означает, что главный выход Z на рис. 92 получен путем объединения указанных маршрутов по схеме «ИЛИ». Математически главный маршрут описывается формулой (4).

$$Z = A + \overline{A}\ \&\ B + \overline{A}\ \&\ \overline{B}\ \&\ C \qquad (4)$$

Правая часть формулы (4) представляет собой три конъюнкта, соединенные знаком логического сложения + (знаком «ИЛИ»). Это значит, что правая часть представляет собой дизъюнктивную нормальную форму (ДНФ).

Инверсный выход $\overline{Z}$ на рис. 92 описывается конъюнктом (5), который также является ДНФ.

$$\overline{Z} = \overline{A}\ \&\ \overline{B}\ \&\ \overline{C} \qquad (5)$$

ВЫВОДЫ

1. Логическая функция «ИЛИ» принимает значение «Да», если хотя бы одна логическая переменная имеет значение «Да». Если же все переменные имеет значение «Нет», функция принимает значение «Нет».
2. Логическая функция «ИЛИ» имеет две нотации: текстовую и графическую.
3. Графическое представление функции «ИЛИ» имеет два выхода: главный и инверсный. Этим она принципиально отличается от текста.
4. Согласно теореме фрагмента инверсный выход $\overline{Z}$ является логическим отрицанием главного выхода Z и наоборот.
5. Графическая схема «ИЛИ» бывает стандартная и нестандартная.
6. Стандартная схема «ИЛИ» содержит несколько икон Вопрос, которые:
   - расположены лесенкой на разных вертикалях;
   - в каждой иконе Вопрос содержится одна переменная.
7. Нестандартная схема «ИЛИ» образуется из стандартной с помощью рокировки.
8. В нестандартной схеме «ИЛИ» все иконы Вопрос лежат на шампуре. В стандартной они расположены лесенкой.
9. В стандартной схеме главный выход схемы «ИЛИ» лежит на шампуре. В нестандартной он расположен справа.
10. Расстановка слов «Да» и «Нет» на выходах икон Вопрос может быть произвольной.
11. Маршруты схемы «ИЛИ» можно описать с помощью конъюнктов.
12. Главный и инверсный выходы схемы «ИЛИ» описывают с помощью дизъюнктивной нормальной формы (ДНФ).

Глава 15

КАК УДАЛИТЬ ЛОГИЧЕСКИЕ СВЯЗКИ ИЗ ЛОГИЧЕСКИХ ВЫРАЖЕНИЙ

ЛОГИЧЕСКИЕ СВЯЗКИ ЖЕЛАТЕЛЬНО УСТРАНИТЬ ИЗ ДРАКОН-СХЕМ

Как известно, логическое отрицание представляет определенную трудность для понимания. В связи с этим Эдвард Йодан советует:

> «Если это возможно, избегайте отрицаний в булевых [логических] выражениях. Представляется, что их понимание представляет трудность для многих программистов» [10].

В похожие ловушки часто попадают не только программисты. Трудности испытывают и многие другие люди.

Опасность ошибочного понимания провоцируют не только знаки отрицания, но и другие связки. Поэтому Йодан расширяет и усиливает свою мысль:

> «Избегайте без нужды использования сложных булевых выражений… Даже если нет неоднозначностей, такие выражения обычно с трудом понимают все, за исключением их автора» [10].

Сходные предостережения делает не только Йодан. Его поддерживает Гленфорд Майерс:

> «Распространенным источником ошибок является использование логических операций И и ИЛИ» [11].

Возникает вопрос: можно ли устранить подобные источники ошибок? Существует ли радикальное средство, позволяющее ликвидировать опасные места, спрятанные в логических выражениях и провоцирующие появление ошибок?

К счастью, от этой неприятности можно избавиться. Ниже будет показано, что логическое отрицание (и другие логические связки) можно безболезненно изъять из графических логических выражений.

РЕКОМЕНДАЦИИ ЭРГОНОМИКИ

Эргономика позволяет сделать алгоритмы (дракон-схемы) более легкими и удобными для понимания. Глядя на эргономичную дракон-схему, человек может сказать: «Посмотрел — и сразу понял!».

Многие люди испытывают трудности, когда видят внутри икон Вопрос сложные логические формулы, содержащие знаки И, ИЛИ, НЕ. Таким людям можно помочь, исключив эти нежелательные знаки.

КАК УБРАТЬ ЛОГИЧЕСКУЮ СВЯЗКУ «НЕ» ИЗ ИКОНЫ ВОПРОС

Логическое отрицание НЕ изображают с помощью верхней черты (рис. 93, вверху слева).

Чтобы удалить логическое отрицание НЕ, нужно выполнить два действия:

- удалить верхнюю черту,
- поменять местами слова «Да» и «Нет» на выходах иконы Вопрос.

Результат показан на рис. 93 вверху справа. Мы видим, что логическая связка НЕ (верхняя черта) бесследно исчезла. При этом логическая схема, как и раньше, выполняет ту же самую функцию — функцию отрицания.

КАК УБРАТЬ ЛОГИЧЕСКУЮ СВЯЗКУ «И» ИЗ ИКОНЫ ВОПРОС

Логическая связка «И» изображается знаком & (амперсанд).

На рис. 93 (в центре слева) показана икона Вопрос, содержащая знак &. Чтобы его удалить, надо нарисовать две иконы Вопрос:

- не содержащие знак &;
- расположенные на шампуре;
- помеченные словом «Да» у нижнего выхода.

Выполнив действия, получим схему на рис. 93 (справа), в которой амперсанд отсутствует. Обе логические схемы (со знаком & и без него) эквивалентны — они реализуют один и тот же алгоритм.

КАК УБРАТЬ ЛОГИЧЕСКУЮ СВЯЗКУ «ИЛИ» ИЗ ИКОНЫ ВОПРОС

Для логической связки «ИЛИ» мы используем два обозначения:

- V (жестовый знак Victoria),
- + (знак логического сложения).

На рис. 93 (внизу слева) в иконе Вопрос показан знак V. Чтобы его убрать, нужно использовать две иконы вопрос:

- не содержащие знак V;
- расположенные лесенкой;
- помеченные словом «Да» у нижнего выхода.

Предлагаемый способ позволяет удалить знак V, что наглядно видно на рис. 93 справа.

Слева на рис. 93 представлены три логические схемы, содержащие логические связки НЕ, И, ИЛИ. Справа показаны схемы, выполняющие те же самые функции, но где связки отсутствуют.

ТЕОРЕМА УДАЛЕНИЯ ЛОГИЧЕСКИХ СВЯЗОК

Изложенный выше метод можно обобщить в виде теоремы.

Теорема . Дракон-схему, содержащую внутри икон Вопрос логические знаки И, ИЛИ, НЕ, всегда можно преобразовать в эквивалентную дракон-схему, не содержащую указанных знаков.

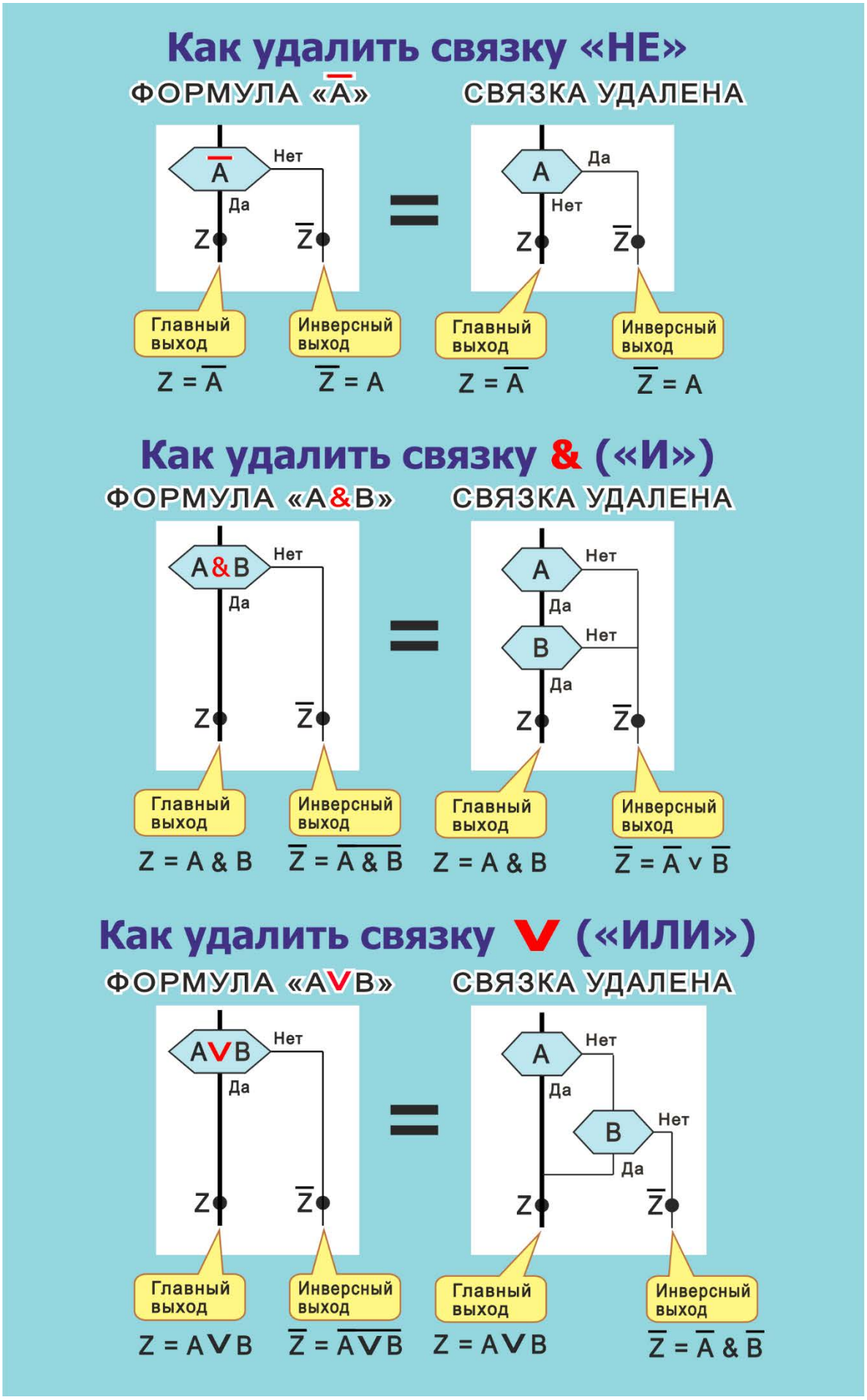


Рис. 93. Удаление логических связок с целью предотвращения ошибок

ПРИМЕРЫ

Практические примеры представлены на рис. 94 и 95, где алгоритмы реализуют логические операции — конъюнкцию (на рис. 94) и дизъюнкцию (на рис. 95). Может создаться (ложное) впечатление, что на рис. 94 и 95 нет никаких логических выражений, так как логические связки отсутствуют.

На самом деле логические выражения не исчезли, а всего лишь изменили форму. Вместо трудной для понимания (и чреватой ошибками) текстовой формы — язык ДРАКОН использует легкую для восприятия графическую форму.

ДРАКОН хорош тем, что позволяет устранить трудные рассуждения о конъюнкциях и дизъюнкциях. Вместо этого надо всего лишь отвечать на простые да-нетные вопросы, что гораздо легче.

Такой подход значительно упрощает объяснение и понимание и снижает требования к квалификации разработчиков. Дополнительную защиту от недоразумений и ошибок обеспечивают комментарии:

- «Замкнуты оба сигнализатора» (рис. 94).
- «Замкнут хотя бы один сигнализатор» (рис. 95).

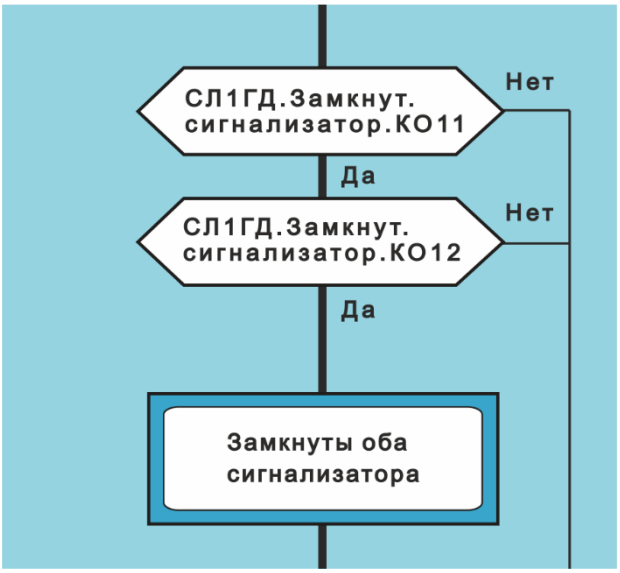


Рис. 94. Выполняется конъюнкция без знака конъюнкции. Комментарий в рамке облегчает понимание и страхует от ошибок

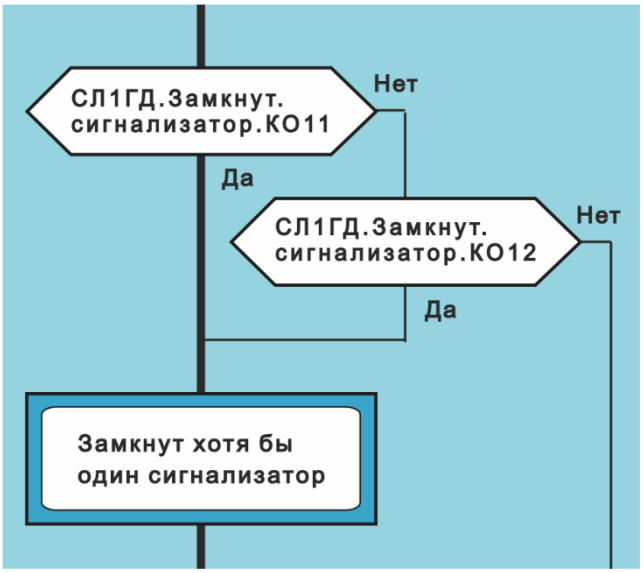


Рис. 95. Выполняется дизъюнкция без знака дизъюнкции. Комментарий облегчает чтение и ускоряет отладку программы

ВЫВОДЫ

1. Многие люди с трудом понимают сложные логические выражения, содержащие логические операции НЕ, И, ИЛИ, что приводит к ошибкам.
2. Чтобы предотвратить ошибки, желательно исключить логические связки НЕ, И, ИЛИ из графических логических выражений.
3. Дракон-схему, содержащую внутри икон Вопрос логические знаки НЕ, И, ИЛИ, можно преобразовать в эквивалентную схему, не содержащую эти знаки.

Глава 16

КАНОНИЧЕСКИЕ ЛОГИЧЕСКИЕ СХЕМЫ

КАНОНИЧЕСКАЯ ДРАКОН-СХЕМА

В главе 15 мы познакомились с понятием «логический фрагмент». Мы помним: фрагмент называется логическим, если он имеет один вход, два выхода и содержит только иконы Вопрос.

Каноническая дракон-схема — это логический фрагмент, который:

- способен описать любую логическую функцию;
- не содержит логических связок И, ИЛИ, НЕ;
- не содержит повторяющихся частей, которые можно удалить с помощью равносильных преобразований.

ЗАДАЧА

Предположим, задана логическая формула. Например, такая

$$Z = (A \,\&\, \overline{B} \,\&\, C) \vee (D \,\&\, E \,\&\, \overline{F})$$

Для этой (или любой иной) формулы можно начертить соответствующий ей логический фрагмент.

Предположим далее, что в полученном фрагменте используются логические связки НЕ (верхняя черта), И (&), ИЛИ (V). Задача состоит в том, чтобы удалить все связки и получить канонический логический фрагмент.

Наша цель — научиться это делать. Процесс решения задачи называется приведением дракон-схемы к каноническому виду.

Чтобы освоить данный метод, мы рассмотрим пять примеров на рис. 96 – 100.

ПРИМЕР 1

Приведение дракон-схемы к каноническому виду есть пошаговый процесс преобразования схемы, причем каждый шаг показан в белом квадрате (рис. 96). Мы видим четыре квадрата, в которых представлены четыре дракон-схемы.

Исходная схема (вверху слева) содержит член $\overline{A \,\&\, B}$. Удалив логическое отрицание (верхнюю черту) и поменяв местами Да и Нет, получаем вторую схему (вверху справа).

Теперь нужно переставить икону С на шампур. Для этого выполняем рокировку и получаем третью схему (внизу слева). Остался последний шаг — убираем амперсанд & и видим желаемый результат (внизу справа). Это и есть каноническая дракон-схема.

Как и планировалось, в канонической дракон-схеме отсутствуют логические связки — они полностью исчезли. Тем не менее, логическая операция И реализуется канонической схемой. Это делается с помощью двух икон Вопрос, содержащих логические переменные A и B.

В данном примере мы использовали три равносильных преобразования:

- удаление логической связки НЕ (верхней черты);
- рокировку;
- удаление логической связки И (&).

Поскольку преобразования равносильны, все четыре дракон-схемы на рис. 96 также являются равносильными.

ЗАЧЕМ НУЖНА ИКОНА С

Икона С обозначает действие; она не входит в состав логического фрагмента.

Мы ввели икону С в наши примеры, исходя из следующих соображений. Логический фрагмент описывает условие. Условие чего? Условие действия. Если условие выполняется — значит, можно реализовать действие.

Отсюда следует, что условие и действие нужно демонстрировать не порознь, а вместе, в одной и той же схеме.

Это первая причина. Но есть и вторая. Как известно, у логического фрагмента два выхода: главный и инверсный. Может оказаться так, что икона действия С будет прицеплена к любому из них. Во избежание недоразумений этот вопрос необходимо стандартизовать. Вводится правило: *в канонической схеме икона действия С должна находиться на шампуре.*

Согласно этому правилу, в наших примерах жирной линией изображается (условный) главный маршрут, который после всех преобразований окажется на шампуре вместе с иконой С.

ГЛАВНЫЙ МАРШРУТ И РОКИРОВКА ДЛЯ АБСТРАКТНЫХ СХЕМ

Понятие «главный маршрут» относится только к смысловым алгоритмам (где можно указать главный маршрут). Оно неприменимо к абстрактным (буквенным) алгоритмам, где понятие главного маршрута теряет силу.

Мы знаем, что преобразование «рокировка» позволяет улучшить эргономичность алгоритмов. Закон рокировки придает нашим эргономическим действиям (позволяющим выпрямить изогнутый главный маршрут) математическую строгость и точность. Отсюда вытекает, что применение рокировки к абстрактным (буквенным) схемам бессмысленно, так как в данном случае рокировка не влияет на эргономичность.

Из этого правила есть исключение. На рис. 96 – 100 мы использовали специальный прием, позволяющий реабилитировать главный маршрут и рокировку. И признать допустимым их использование в абстрактных схемах.

Для этого мы ввели требование: икона С, обозначающая действие, должна находиться на шампуре, а (условный) главный маршрут следует выделить жирной линией. Отсюда немедленно вытекает необходимость переместить икону С на шампур, а для этого как раз и нужна рокировка.

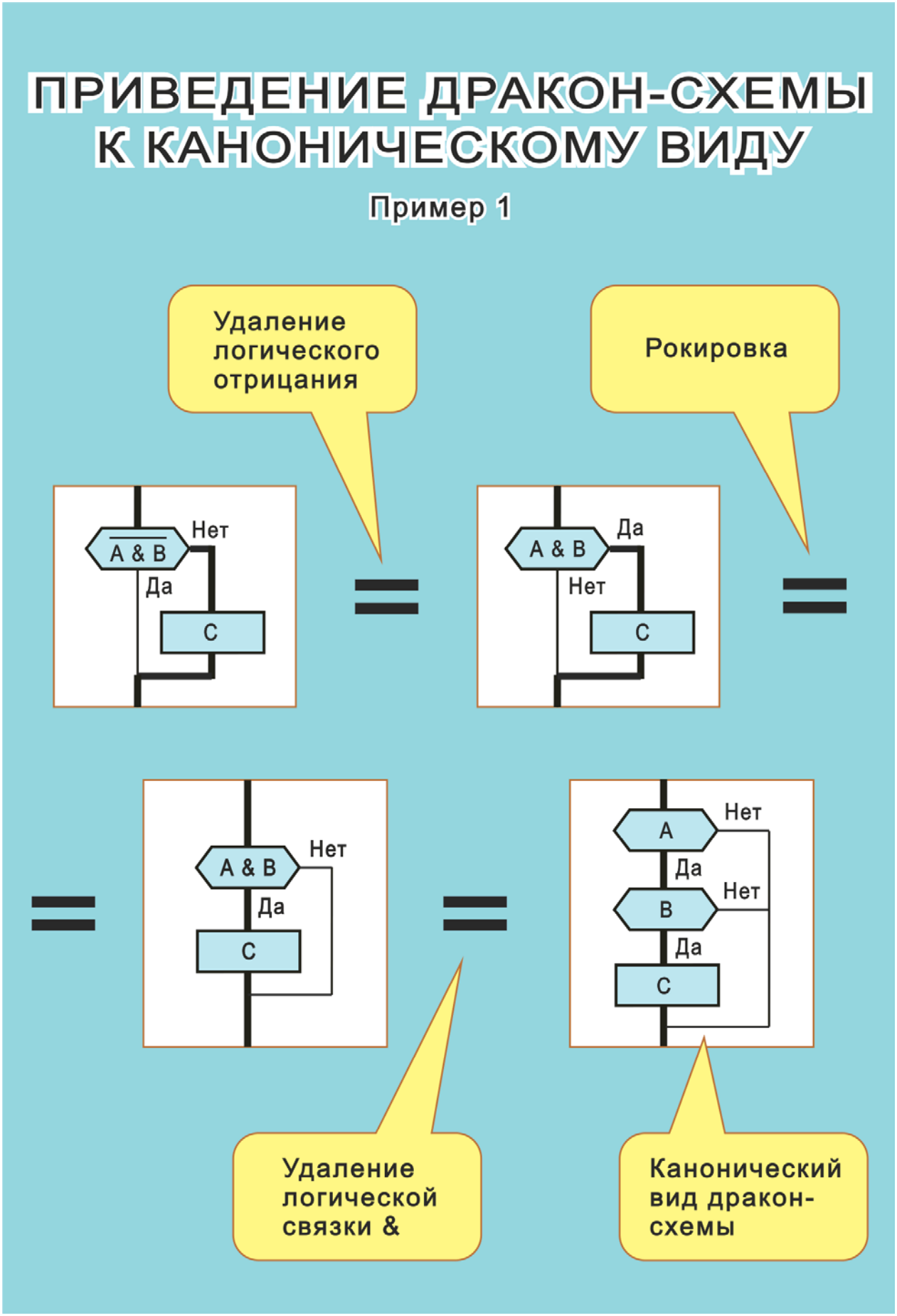


Рис. 96. Цепочка равносильных преобразований позволяет убрать две связки (& и черту) и привести дракон-схему к каноническому виду. (Пример 1)

ПРИМЕР 2

Пример на рис. 97 по своей структуре почти полностью повторяет пример 1 на рис. 96. Разница лишь в том, что вместо схемы И рассматривается схема ИЛИ.

Как и раньше, мы снова видим четыре квадрата, в которых представлены четыре дракон-схемы.

Исходная схема (вверху слева) содержит член $\overline{A \vee B}$. Удалив логическое отрицание (верхнюю черту) и переставив слова Да и Нет, получаем вторую схему (вверху справа).

Далее надо переместить икону С на шампур. Реализуем это с помощью рокировки. Получаем третью схему (внизу слева).

В заключение убираем логическую связку V и получаем окончательный результат (внизу справа). Мы построили каноническую дракон-схему.

Хотя в канонической схеме отсутствуют логические связки, она выполняет те же самые логические функции, что и исходная схема.

В данном примере использованы три равносильных преобразования:

- удаление связки НЕ (верхней черты);
- рокировка;
- удаление связки ИЛИ (V).

Поскольку преобразования равносильны, вывод остается неизменным — все четыре дракон-схемы на рис. 97 являются равносильными.

ПРИМЕР 3

Примеры 1 и 2 демонстрируют, что каноническая схема совпадает со стандартной схемой «И» (а также «ИЛИ») для двух переменных, причем выходы «Да» направлены вниз.

Всегда ли так бывает? Всегда ли у канонической схемы выходы «Да» направлены вниз? Нет, не всегда. Чтобы убедиться в этом, рассмотрим пример на рис. 98.

Здесь мы видим пять белых квадратов, в которых изображены пять дракон-схем.

Исходная схема (вверху слева) содержит член с двумя отрицаниями $\overline{A \vee \overline{B}}$. Удалив первое отрицание (самую верхнюю черту) и переставив слова Да и Нет, получаем вторую схему (вверху в центре).

Затем надо перетащить икону С на шампур. Сделаем это с помощью рокировки. Получим третью схему (вверху справа).

После этого удаляем амперсанд & и реализуем схему И графически в виде двух икон Вопрос (внизу слева).

Напоследок убираем верхнюю черту у члена $\overline{B}$ и получаем окончательный результат — каноническую схему (внизу справа). Обратите внимание: в канонической схеме у иконы Вопрос нижний выход помечен словом «Нет».

В данном примере использованы уже не три, а четыре равносильных преобразования:

- удаление связки НЕ (самой верхней черты);
- рокировка;
- удаление логической связки И (&);
- удаление второй связки НЕ (верхней черты).

Как и раньше, все преобразования равносильны. Поэтому все пять дракон-схем на рис. 97 также являются равносильными.

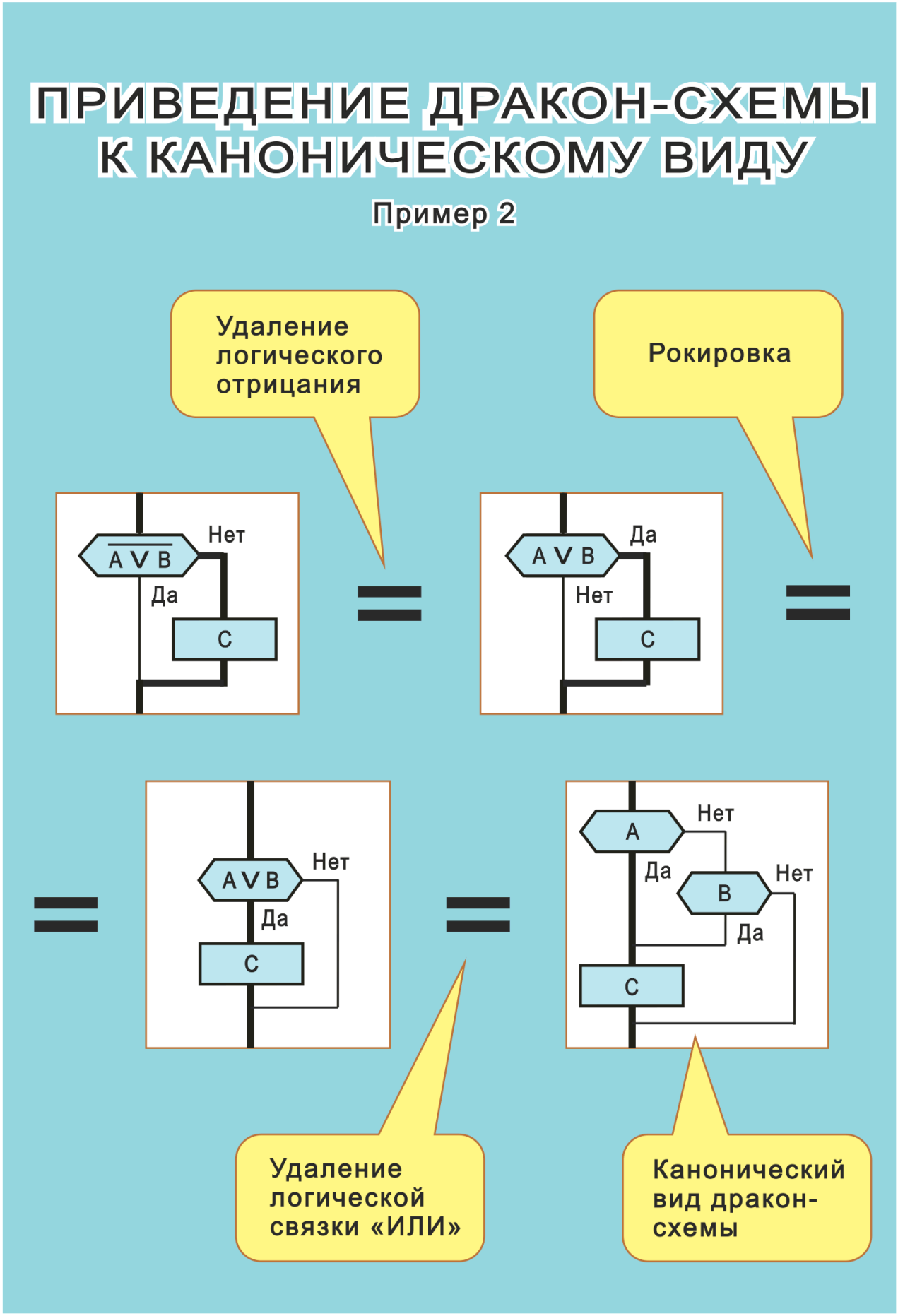


Рис. 97. Цепочка равносильных преобразований позволяет убрать две связки (V и черту) и привести дракон-схему к каноническому виду. (Пример 2)

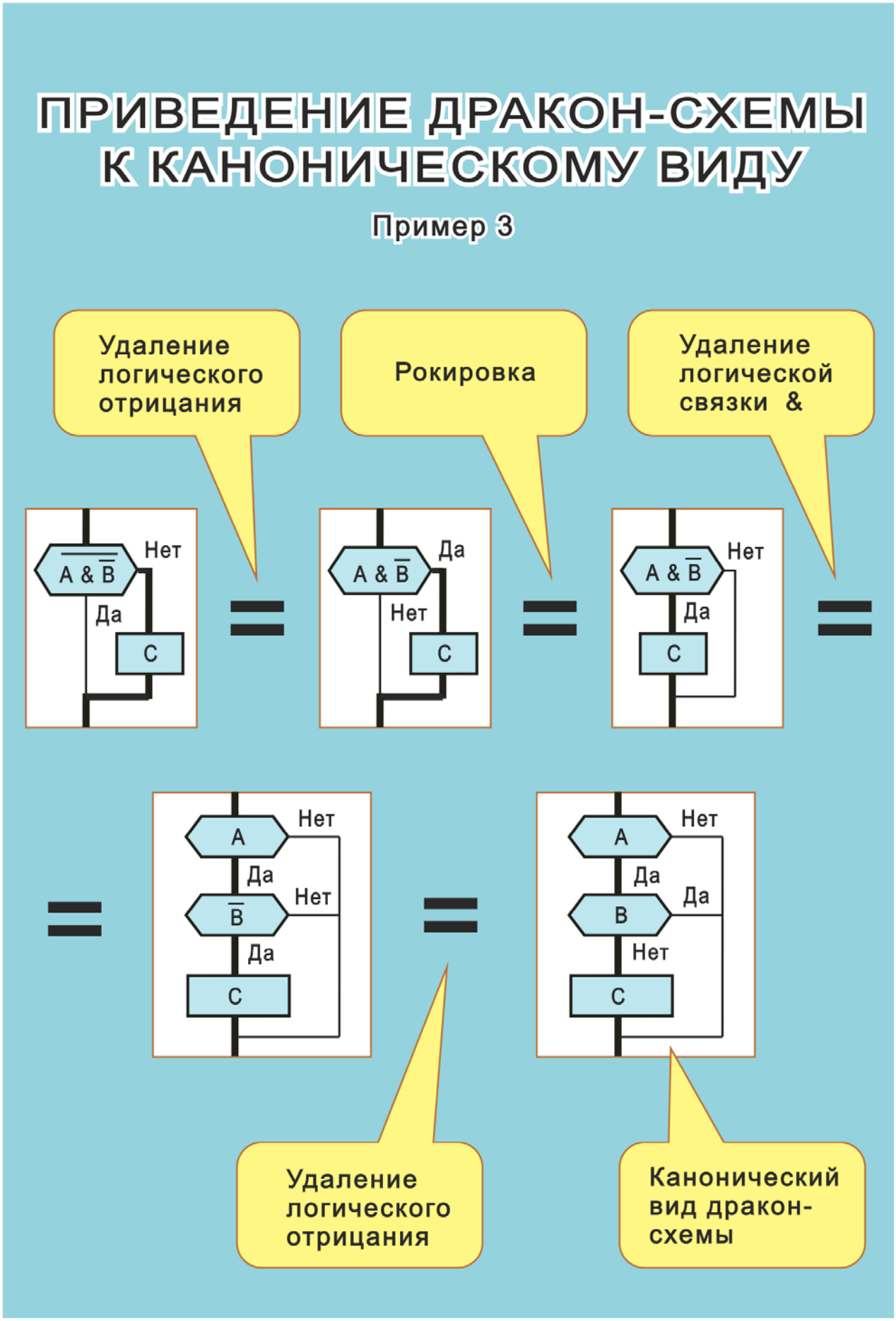


Рис. 98. Цепочка равносильных преобразований позволяет убрать три связки (& и две черты) и привести дракон-схему к каноническому виду. (Пример 3)

ПРИМЕР 4

Приведение дракон-схемы к каноническому виду продемонстрируем на более простом примере (рис. 99).

Исходная схема (вверху слева) содержит член $A \& \overline{B}$. Здесь мы видим две кандидатуры на удаление: логические связку «НЕ» и связку «И». С какой начать?

Впрочем, выбора у нас нет, поскольку связки располагаются по старшинству: сначала И, потом НЕ. Отсюда следует, что мы вынуждены начать удаление с амперсанда. Убрав &, превращаем его в две иконы Вопрос и получаем вторую схему (вверху, в центре).

После этого стираем верхнюю черту у члена $\overline{B}$ и меняем местами Да и Нет. Получаем третью схему (вверху справа).

Осталось самое простое — пересадить икону С на шампур. Для этого выполняем рокировку и получаем то, что хотели — каноническую схему (внизу слева).

Как и было задумано, в канонической дракон-схеме нет никаких логических связок — они полностью исчезли. Тем не менее, каноническая схема выполняет все положенные логические функции, что достигается с помощью графики.

В данном примере мы использовали три равносильных преобразования:

- удаление логической связки И (&);
- удаление логической связки НЕ (верхней черты);
- рокировку.

КУДА СМОТРИТ НИЖНИЙ ВЫХОД ИКОНЫ ВОПРОС

Куда смотрит — не совсем точное выражение. Правильнее сказать: как помечен нижний выход иконы Вопрос: словом «Да» или словом «Нет»?

Речь идет, конечно, о канонических схемах. Здесь имеет место удивительное разнообразие.

Взглянем на рис. 96 и 97. В обоих случаях канонические схемы построены по единому лекалу — все иконы Вопрос смотрят вниз через выход «Да».

Но так бывает не всегда. Давайте приглядимся к рис. 98. Икона А смотрит вниз через «Да», а икона В — через «Нет».

Что ж, переведем взгляд на рис. 99. Здесь все наоборот: икона А смотрит вниз через «Нет», а икона В — через «Да».

О чем это говорит? О том, что мир разнообразен и канонические схемы отражают разнообразие мира.

ДВЕ ЗОНЫ

Дракон-схема на рис. 99, *внизу* делится на две зоны: верхнюю и нижнюю. Граница между зонами обозначена в виде двух жирных точек Z и $\overline{Z}$. Верхняя зона представляет собой логический фрагмент, причем точки Z и $\overline{Z}$ символизируют главный и инверсный выходы фрагмента. Фрагмент описывает важную вещь — условие выполнение Действия.

Нижняя зона, включающая икону С (которая описывает Действие и соединительные линии), не входит в состав логического фрагмента. Она представляет собой добавление к фрагменту. Зачем нужен добавление? Оно нужно потому, что Условие и Действие желательно демонстрировать не порознь, а вместе.

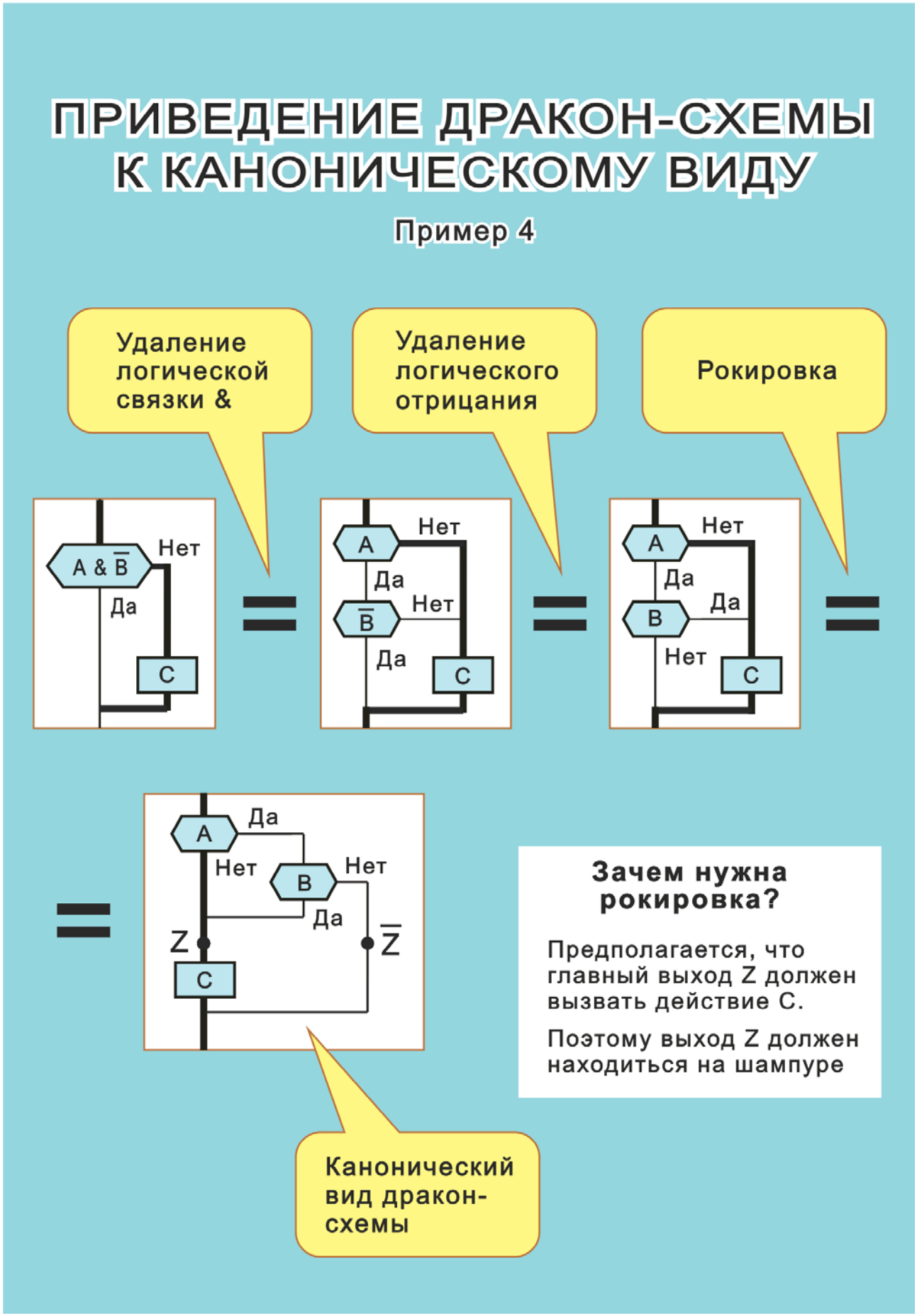
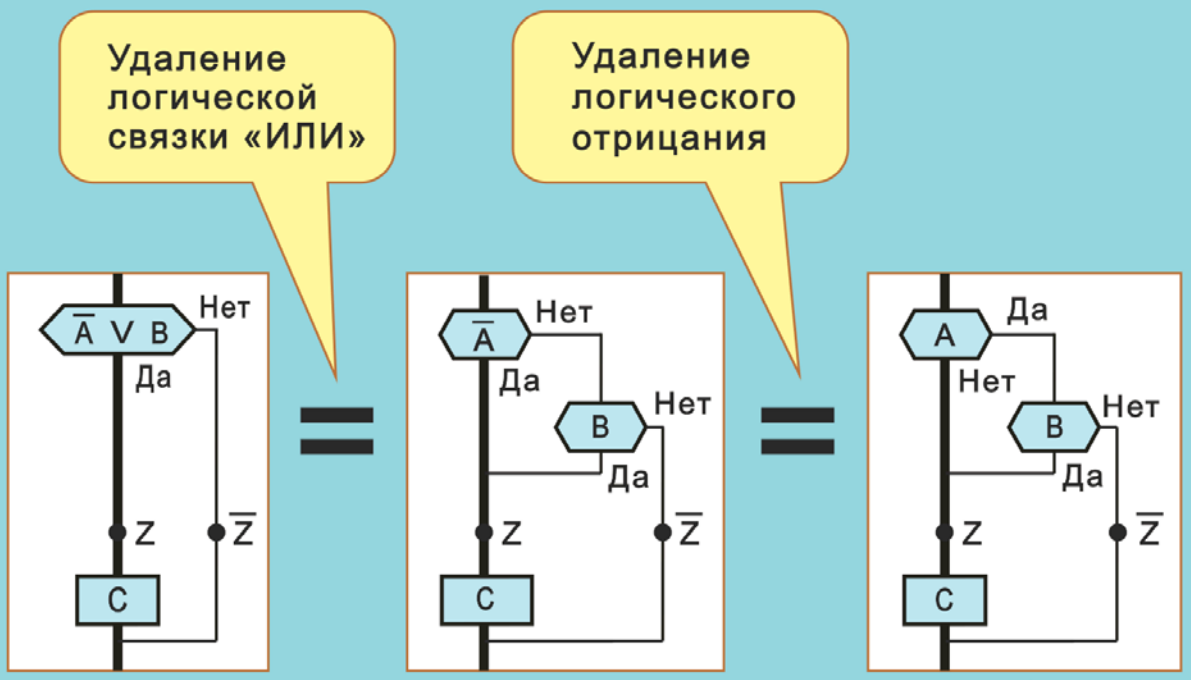


Рис. 99. Цепочка равносильных преобразований позволяет убрать две связки (& и черту) и привести дракон-схему к каноническому виду. (Пример 4)

ИМПЛИКАЦИЯ

ПРИВЕДЕНИЕ ДРАКОН-СХЕМЫ К КАНОНИЧЕСКОМУ ВИДУ

Пример 5



Обозначим импликацию через Z

$$Z = A \rightarrow B = \overline{A} \vee B$$

ИМПЛИКАЦИЯ

Таблица истинности

A	B	$Z = A \rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

Рис. 100. Импликация. Цепочка равносильных преобразований позволяет привести дракон-схему к каноническому виду. (Пример 5)

ПРИМЕР 5. ИМПЛИКАЦИЯ

Пример интересен тем, что логическая функция $\overline{A} \vee B$ играет важную роль в логике и называется «импликация» (следование).

Импликация соответствует союзу «если, то». Если на улице дождь, то асфальт мокрый. Обозначим Дождь через A, а Мокрый асфальт через B. Тогда логическая функция «Импликация» примет вид:

$$Z = \overline{A} \vee B$$

Импликация Z задается таблицей истинности:

A	**B**	$\mathbf{Z = \overline{A} \vee B}$
0	0	1
0	1	1
1	0	0
1	1	1

Таблицу надо зазубрить, потому что объяснить ее простыми словами трудно.

ПРИВЕДЕНИЕ ИМПЛИКАЦИИ К КАНОНИЧЕСКОМУ ВИДУ

Обратимся к рисунку 100. Исходная схема (вверху слева) содержит импликацию — член $\overline{A} \vee B$. Начнем удаление со связки «ИЛИ». Убираем связку V, разъединяем буквы A и B, рассаживаем их в разные иконы Вопрос. Тем самым создаем вторую схему (вверху в центре).

После этого стираем верхнюю черту у члена $\overline{A}$ и меняем местами Да и Нет. Получаем третью и последнюю схему (вверху справа). Это и есть каноническая схема импликации.

Как и было обещано, в канонической дракон-схеме нет логических связок — легко убедиться, что они отсутствуют. Тем не менее, каноническая схема выполняет все необходимые логические операции. Разница лишь в том, что вместо связок применяется математически строгая графика.

В данном примере мы использовали два равносильных преобразования:

- удаление логической связки ИЛИ (V);
- удаление логической связки НЕ (верхней черты).

ОБЪЯСНЯЕМ ИМПЛИКАЦИЮ НА СОДЕРЖАТЕЛЬНОМ ПРИМЕРЕ

Вернемся к примеру импликации «Если на улице дождь, то асфальт мокрый». Будем сравнивать таблицу истинности с канонической схемой на рис. 100а и содержательной схемой на рис. 100б.

«Если на улице дождь» — это посылка. «Асфальт мокрый» — следствие. Связь посылки со следствием описана в 4-й строке таблицы истинности: A = 1. B = 1. Z = 1.

О чем говорит 4-я строка? «Если на улице дождь, то асфальт мокрый» — это правильное утверждение, т. е. Истина.

Рис. 100б подтверждает это. В самом деле, из иконы Вопрос «Дождь идет?» выходим вправо через Да. Затем из иконы «Асфальт мокрый?» выходим вниз через Да. Далее по этому маршруту попадаем в икону Истина. Мы убедились, что 4-я строка таблицы истинности совпадает с данным маршрутом на рис. 100б.

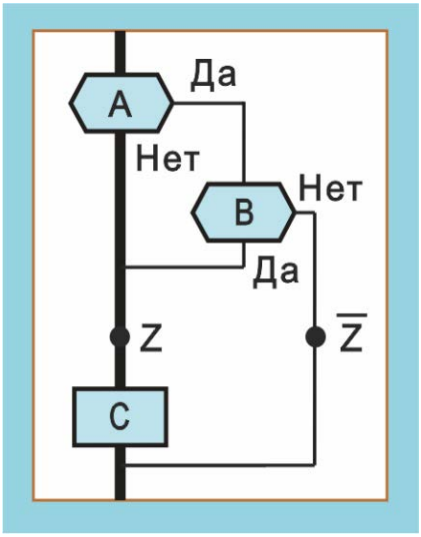


Рис. 100а. Импликация. Каноническая схема

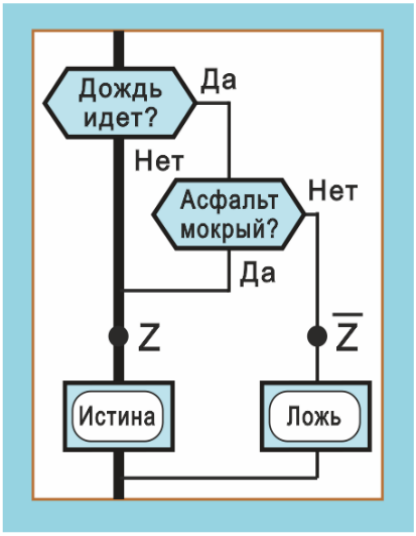


Рис. 100б. Импликация. Содержательная схема

Рассмотрим 3-ю строку таблицы истинности: A = 1. B = 0. Z = 0. В чем ее смысл? «Если на улице дождь, а асфальт сухой» — это ошибочное утверждение, т. е. Ложь.

Рис. 100б подтверждает, что это Ложь. Действительно, из иконы Вопрос «Дождь идет?» выходим вправо через Да, а из иконы «Асфальт мокрый?» выходим вправо через Нет. Далее этот маршрут приводит нас в икону Ложь. Мы выяснили, что 3-я строка таблицы истинности в точности совпадает с крайним правым маршрутом на рис. 100б.

Нам осталось рассмотреть первые две строки таблицы истинности. В обоих случаях Дождь не идет, т. е. A = 0. Как это понимать?

Если Дождь не идет, то импликация Z всегда равна 1, независимо от того, какой асфальт (мокрый или сухой).

Давайте проверим по рис. 100б. Из иконы «Дождь идет?» выходим вниз через Нет и по шампуру сразу попадаем в икону Истина.

ИМПЛИКАЦИЯ. КАК СВЯЗАНЫ ПОСЫЛКА И СЛЕДСТВИЕ

Истинная причинная связь посылки и следствия описаны в 4-й строке таблицы истинности. И в маршруте Да-Да на рис. 100а и 100б.

Если посылка истинная (Дождь идет), а следствие ложное (Асфальт сухой), причинная связь исчезает, и импликация становится ложной. Этот случай описан в 3-й строке таблицы истинности. И в маршруте Да-Нет на рис. 100а и 100б.

Если посылка ложная (Дождь не идет), причинная связь посылки и следствия также отсутствует, но импликация истинная. Этот случай описан в 1-й и 2-й строках таблицы истинности. И в маршруте, идущем по шампуру на рис. 100а и 100б.

ВЫВОДЫ

1. Каноническая дракон-схема способна описать любую логическую функцию алгебры логики без использования логических связок.
2. Процесс удаления из схемы логических связок называется приведением дракон-схемы к каноническому виду.
3. Для любой логической формулы алгебры логики можно построить каноническую дракон-схему.

Глава 17

ЛОГИЧЕСКАЯ ФУНКЦИЯ «ИСКЛЮЧАЮЩЕЕ ИЛИ»

ВВЕДЕНИЕ

В главах 14 и 16 описана логическая функция «ИЛИ». На нее отчасти похожа функция «Исключающее ИЛИ», известная также как «строгая дизъюнкция». Разница, казалось бы, небольшая и касается лишь одной комбинации, когда все логические переменные равны 1. Однако это отличие играет важную роль и нуждается в пояснениях. Мы поведем рассказ по следующему плану:

- Логическая схема «Исключающее ИЛИ». Пример.
- Таблица истинности.
- Как представить функцию «Исключающее ИЛИ» на языке ДРАКОН.
- Неполная и полная схема «Исключающее ИЛИ».
- Формула для инверсного выхода.
- Переключатель маршрутов в иконе Вопрос.
- Переключатель маршрутов в логическом фрагменте.
- Что такое хорошо и что такое плохо.
- Что лучше: один переключатель или два.
- Четыре комбинации логических переменных.
- Классическая и неклассическая алгебра логики.
- Запрещенные маршруты.
- Приведение функции «Исключающее ИЛИ» к каноническому виду.
- Простая схема для строгой дизъюнкции.

ЛОГИЧЕСКАЯ СХЕМА «ИСКЛЮЧАЮЩЕЕ ИЛИ». ПРИМЕР

Том и Гек узнали, что на том берегу разбойники спрятали клад, и решили найти его. Но как переправиться через реку?

Через реку перекинут одноразовый мост. Что значит одноразовый? Это значит, что по мосту может пройти только один мальчик. Либо Том, либо Гек, но не оба вместе. Если дети пойдут вдвоем, произойдет несчастье и они погибнут.

Как раздобыть клад?

Ответ. За кладом должен отправиться кто-то один. Или Том, или Гек.

Обозначим Тома через A, а Гека через B. Тогда логическая функция Z «Исключающее ИЛИ» (строгая дизъюнкция) примет вид.

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

ТАБЛИЦА ИСТИННОСТИ

Функцию Z (строгую дизъюнкцию) можно задать таблицей истинности:

A	B	$Z = (A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B)$
0	0	0
0	1	1
1	0	1
1	1	0

О чем говорит 1-я строка таблицы? Если Том не пошел за кладом (A = 0) и Гек тоже не пошел (B = 0), то клад, разумеется, не будет найден и логическая функция Z равна 0.

2-я строка. Если Том не пошел за кладом, а Гек пошел, то — ура! — **клад найден** и функция Z равна **1**.

3-я строка. Если Том пошел за кладом, а Гек не пошел, то — снова ура! — **клад найден** и функция Z = **1**.

4-я строка. Если же Том и Гек пойдут вдвоем, они погибнут, клад не будет найден и функция Z равна 0.

Таблицу истинности можно слегка изменить. Заменив 1 и 0 на Да и Нет, получим:

A	B	$Z = (A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B)$
Нет	Нет	Нет
Нет	Да	Да
Да	Нет	Да
Да	Да	Нет

КАК ПРЕДСТАВИТЬ ФУНКЦИЮ «ИСКЛЮЧАЮЩЕЕ ИЛИ» НА ЯЗЫКЕ ДРАКОН

Выше мы описали строгую дизъюнкцию с помощью формулы $Z = (A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B)$. На рис. 101 эта формула изображена графически в виде четырех икон Вопрос, нанизанных на две вертикали.

Рассмотрим левую вертикаль (шампур). Маршрут, идущий по шампуру, можно описать так:

- Том пошел за кладом? Да.
- Гек пошел за кладом? Нет.

Значит, за кладом пошел только один мальчик — Том. Это хороший маршрут, ведущий к успеху — к команде «Бери клад».

Рассмотрим правую вертикаль. Она описывает другой маршрут, а именно:

- Том пошел за кладом? Нет.
- Гек пошел за кладом? Да.

Следовательно, и здесь за кладом отправился только один ребенок, Гек. Это также благоприятный маршрут, позволяющий раздобыть клад.

Обратите внимание: у двух боковых икон Вопрос мы (для наглядности) убрали выходы вправо (так как они никак не влияют на главный выход).

Точка Z, в которую попадает бегунок, пробежавший по любому из двух «хороших» маршрутов, есть графическое изображение функции «Исключающее ИЛИ» (для случая, когда **клад найден**).

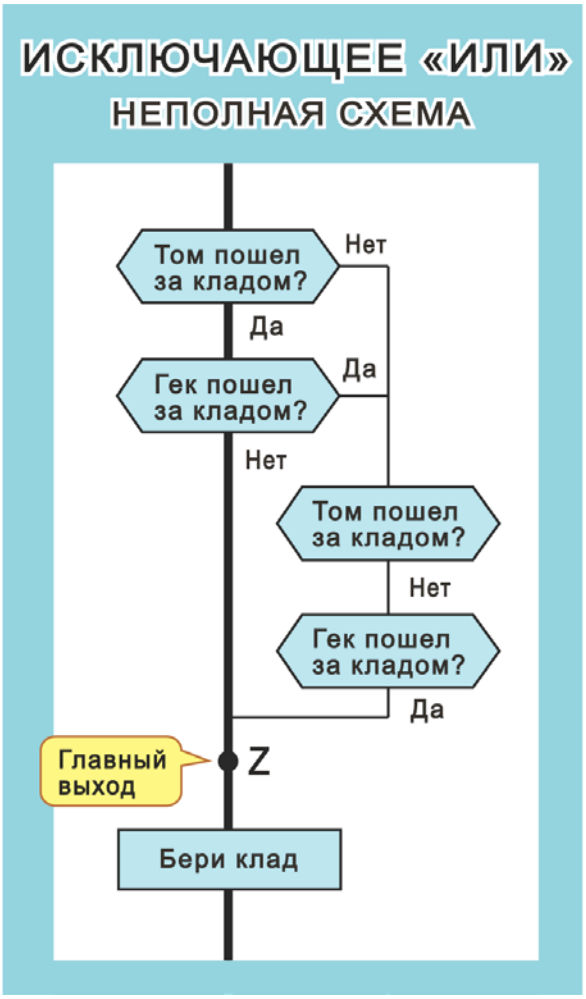


Рис. 101. Главный выход Z находится на шампуре. Мы нарочно оторвали инверсный выход $\overline{Z}$, чтобы сконцентрировать внимание на логической функции $Z = (A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B)$.

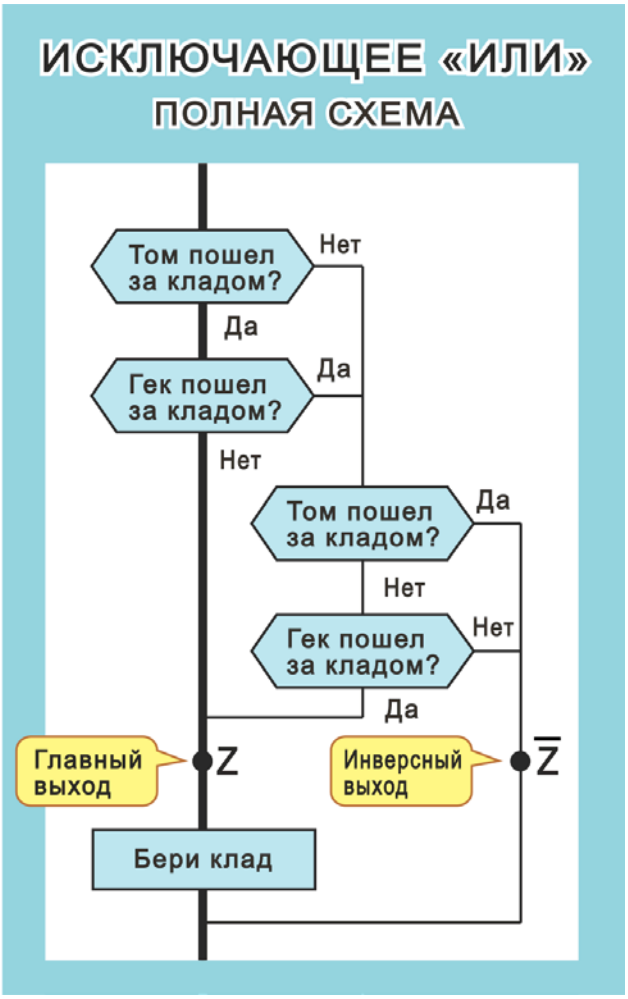


Рис. 102. Справа от шампура показан инверсный выход $\overline{Z} = (A \,\&\, B) \vee (\overline{A} \,\&\, \overline{B})$
Инверсный выход $\overline{Z}$ является логическим отрицанием главного выхода $Z = (A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B)$

НЕПОЛНАЯ И ПОЛНАЯ СХЕМА «ИСКЛЮЧАЮЩЕЕ ИЛИ»

На рис. 101 показана *неполная* дракон-схема, у которой удалены правые выходы на правой вертикали. Дефект исправлен на рис. 102, где показана *полная* схема, содержащая не только главный, но и инверсный выход.

Какая польза от неполной схемы? Она позволяет упростить задачу и облегчить понимание работы главного выхода. В чем облегчение? В том, что из поля зрения убран инверсный выход и все внимание сосредоточено на главном выходе.

На рис. 102 функция «Исключающее ИЛИ» изображена полностью. Графика реализует функцию $Z = (A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B)$.

Обратите внимание: бегунок покидает схему через главный выход, когда функция Z принимает значение 1 (т.е. когда за кладом пошел кто-то один: либо Том, либо Гек). И наоборот, бегунок движется через инверсный выход, когда $Z = 0$, а $\overline{Z} = 1$, (т.е. оба мальчика либо пошли за кладом, либо остались дома).

ФОРМУЛА ДЛЯ ИНВЕРСНОГО ВЫХОДА

Рассмотрим формулу (1). Она описывает инверсный выход $\bar{Z}$ дракон-схемы «Исключительное ИЛИ» на рис. 102.

$$\bar{Z} = \overline{(A \,\&\, \bar{B}) + (\bar{A} \,\&\, B)} \qquad (1)$$

Наша цель — упростить выражение (1). При этом будем опираться на известные соотношения алгебры логики [8].

- Закон де Моргана $\overline{x + y} = \bar{x} \,\&\, \bar{y}$
- Закон дистрибутивности $x \,\&\, (y + w) = x \,\&\, y + x \,\&\, w$
- Закон противоречия $x \,\&\, \bar{x} = 0$
- Закон двойного отрицания $\bar{\bar{x}} = x$

Для обозначения дизъюнкции вместо знака V будем использовать знак логического сложения +

$\bar{Z} = \overline{(A \,\&\, \bar{B}) + (\bar{A} \,\&\, B)} = \overline{(A \,\&\, \bar{B})} \,\&\, \overline{(\bar{A} \,\&\, B)} = (\bar{A} + \bar{\bar{B}}) \,\&\, (\bar{\bar{A}} + \bar{B}) = (\bar{A} + B) \,\&\, (A + \bar{B}) =$

$= \bar{A} \,\&\, A + B \,\&\, A + \bar{A} \,\&\, \bar{B} + B \,\&\, \bar{B} = 0 + B \,\&\, A + \bar{A} \,\&\, \bar{B} + 0 = A \,\&\, B + \bar{A} \,\&\, \bar{B}$

Итак, мы получили окончательную формулу для инверсного выхода $\bar{Z}$.

$$\bar{Z} = (A \,\&\, B) + (\bar{A} \,\&\, \bar{B}) \qquad (2)$$

Что означает формула (2)? Какой у нее физический смысл?

Бегунок выходит через инверсный выход, когда $\bar{Z} = 1$. Согласно формуле (2) это возможно в двух случаях:

- $A \,\&\, B = 1$
- $\bar{A} \,\&\, \bar{B} = 1$

Если член $A \,\&\, B$ равен 1, значит $A = B = 1$. Вспомним, что А — это Том, а В это Гек. Выражение $A \,\&\, B = 1$ означает, что Том и Гек пошли за кладом вдвоем. Согласно условию, в такой ситуации произойдет несчастье, и клад не будет найден.

Рассмотрим другой вариант. Если член $\bar{A} \,\&\, \bar{B}$ равен 1, значит $A = B = 0$. Следовательно, Том и Гек никуда не пошли и клад не найден.

Таким образом, мы выяснили смысл формулы (1), которая описывает инверсный выход дракон-схемы. Оба мальчика либо вдвоем пошли за кладом и потерпели неудачу, либо остались дома. В обоих случаях клад не найден.

ПЕРЕКЛЮЧАТЕЛЬ МАРШРУТОВ В ИКОНЕ ВОПРОС

Переключатель маршрутов — это логическая функция А, записанная в иконе Вопрос и принимающая значение 1 или 0 (рис. 103).

Если переключатель А = 1, бегунок движется через главный выход $Z = A = 1$. Главный выход помечен словом «Да».

Если же переключатель А = 0, бегунок идет через инверсный выход $\bar{Z} = \bar{A} = 1$. Инверсный выход помечен словом «Нет».

Один из двух выходов (Z и $\bar{Z}$) всегда равен 1. Бегунок всегда движется через тот выход, который равен 1.

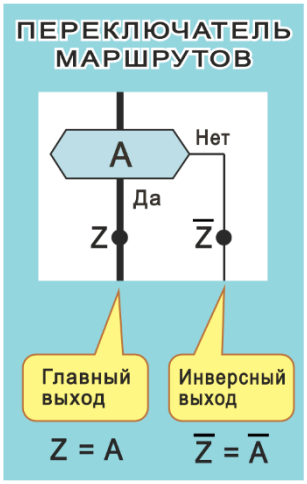


Рис. 103. В иконе Вопрос записан переключатель А

ПЕРЕКЛЮЧАТЕЛЬ МАРШРУТОВ В ЛОГИЧЕСКОМ ФРАГМЕНТЕ

Любой логический фрагмент имеет переключатель маршрутов. Переключатель есть логическая функция, которую реализует фрагмент. Например, фрагмент на рис. 102 имеет переключатель $(A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B)$.

Если этот переключатель равен 1, бегунок движется через главный выход $Z = (A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B) = 1$.

Если же переключатель равен 0, бегунок держит путь через инверсный выход $\overline{Z} = (A \,\&\, B) + (\overline{A} \,\&\, \overline{B}) = 1$.

Один из двух выходов (Z и $\overline{Z}$) логического фрагмента непременно равен 1. Бегунок всегда движется через тот выход фрагмента, который равен 1.

Примечание. В случае фрагмента привязка слов «Да» и «Нет» к главному и инверсному выходам в общем случае может не соблюдаться (см. пример на рис. 104).

ЧТО ТАКОЕ ХОРОШО И ЧТО ТАКОЕ ПЛОХО

Продолжим рассказ про Тома и Гека. Действия мальчиков разделим на «хорошие» (помогающие добыть клад) и «плохие» (когда клад не найден). В качестве критерия выберем таблицу истинности. Именно она позволяет разделить значения функции Z на хорошие (равные 1 и позволяющие найти клад) и плохие (равные 0 и ведущие к неудаче).

A	**B**	$\mathbf{Z = (A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B)}$	**Результат**
0	0	0	Клад не найден
0	1	**1**	**Клад найден**
1	0	**1**	**Клад найден**
1	1	0	Клад не найден

Таблица истинности подтверждает, что значение переключателя Z = 1 является хорошим (**клад найден**), а Z = 0 — плохим (клад не найден).

ЧТО ЛУЧШЕ: ОДИН ПЕРЕКЛЮЧАТЕЛЬ ИЛИ ДВА

Обратимся к классической алгебре логики. В ней успех (клад найден) и неудачу (клад не найден) описывают с помощью одного переключателя Z.

$$Z = (A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B) \qquad (3)$$

При этом хороший исход (клад найден) трактуется как Z = 1, а плохой (клад не найден) как Z = 0.

Язык ДРАКОН использует неклассический (графический) вариант алгебры логики. В отличие от классического варианта, он описывает успех и неуспех иначе — с помощью не одного, а двух инверсных переключателей: Z и $\overline{Z}$. При этом успех (клад найден) понимается как $Z = 1$, а неуспех (клад не найден) как $\overline{Z} = 1$. Иными словами, в обоих случаях используются только единицы.

Вспомним, что в алгоритме бегунок всегда движется через тот выход логического фрагмента, который равен 1.

ЧЕТЫРЕ КОМБИНАЦИИ ЛОГИЧЕСКИХ ПЕРЕМЕННЫХ

Таблица истинности для функции «Исключающее ИЛИ» содержит 4 строки. Каждая строка описывает свою комбинацию логических переменных A и B. Перечислим их в нужном для нас порядке.

- $A \,\&\, \overline{B}$ (комбинация 10).
- $\overline{A} \,\&\, B$ (комбинация 01).
- $A \,\&\, B$ (комбинация 11).
- $\overline{A} \,\&\, \overline{B}$ (комбинация 00).

В классической алгебре логики все четыре комбинации задаются в виде **одного** переключателя (3). Как это делается? Первые две комбинации $(A \,\&\, \overline{B})$ и $(\overline{A} \,\&\, B)$ соответствуют значениям Z = 1, последние две — значениям Z = 0 (см. таблицу истинности). Формула (3) в явном виде показывает лишь первую пару комбинаций: $A \,\&\, \overline{B}$ и $\overline{A} \,\&\, B$. А вторую пару: $A \,\&\, B$ и $\overline{A} \,\&\, \overline{B}$ она скрывает.

Перейдем к неклассическому варианту, представленному на рис. 104 и 105. Здесь указанные комбинации описываются уже не в одном, а в **двух** переключателях.

Первые две комбинации $(A \,\&\, \overline{B})$ и $(\overline{A} \,\&\, B)$ изображаются как члены (конъюнкты) в переключателе **главного** выхода $Z = (A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B)$.

Последние две комбинации $(A \,\&\, B)$ и $(\overline{A} \,\&\, \overline{B})$ изображаются в переключателе **инверсного** выхода $\overline{Z} = (A \,\&\, B) \vee (\overline{A} \,\&\, \overline{B})$.

КЛАССИЧЕСКАЯ И НЕКЛАССИЧЕСКАЯ АЛГЕБРА ЛОГИКИ

Укажем различие между классическим и неклассическим вариантами алгебры логики (на примере строгой дизъюнкции).

В классическом варианте используется линейная запись логической функции Z «Исключающее ИЛИ», которая не показывает свою инверсию $\overline{Z}$ и принципиально не может описать инверсный маршрут.

В неклассическом случае применяется графическая запись логической функции «Исключающее ИЛИ» (рис. 104 и 105), которая изображает одновременно как главный, так и инверсный выходы.

Зададим вопрос: Какой принцип используется для разграничения успеха и неудачи?

Классический вариант использует «принцип отделения единиц от нулей». Если функция равна 1 — это хорошо (клад найден), если она равна 0 — плохо.

Неклассический вариант использует «принцип переключателя маршрутов». Если выбран главный выход, это хорошо (клад найден), если инверсный — плохо.

ЗАПРЕЩЕННЫЕ МАРШРУТЫ

Читаем схему на рис. 104 по правому верхнему пути:

- Том пошел за кладом? Нет.
- Том пошел за кладом? Да.
- Клад не найден.

— Что за нелепицу мы только что прочитали? — спросит иной читатель.

Объяснение простое. Это не нелепость, а запрещенный маршрут. Почему запрещенный? Потому что на вопрос «Том пошел за кладом?» даны взаимоисключающие ответы Нет и Да. Налицо явное противоречие.

Если сказано «Нет», ответ «Да» уже невозможен. Да и Нет вступают в конфликт. Либо Да, либо Нет — третьего не дано.

Что отсюда следует? Предположим, бегунок покинул верхнюю икону «Том пошел за кладом?» через выход «Нет». Далее он попадает в нижнюю икону «Том пошел за кладом?» А теперь внимание! — во избежание противоречия Том должен идти вниз через «Нет». Путь вправо через «Да» для него закрыт!

Отсюда следует, что некоторые маршруты в алгоритмах могут оказаться запрещенными. Об этом нельзя забывать. Например, на рис. 105 маршрут по шампуру запрещен.

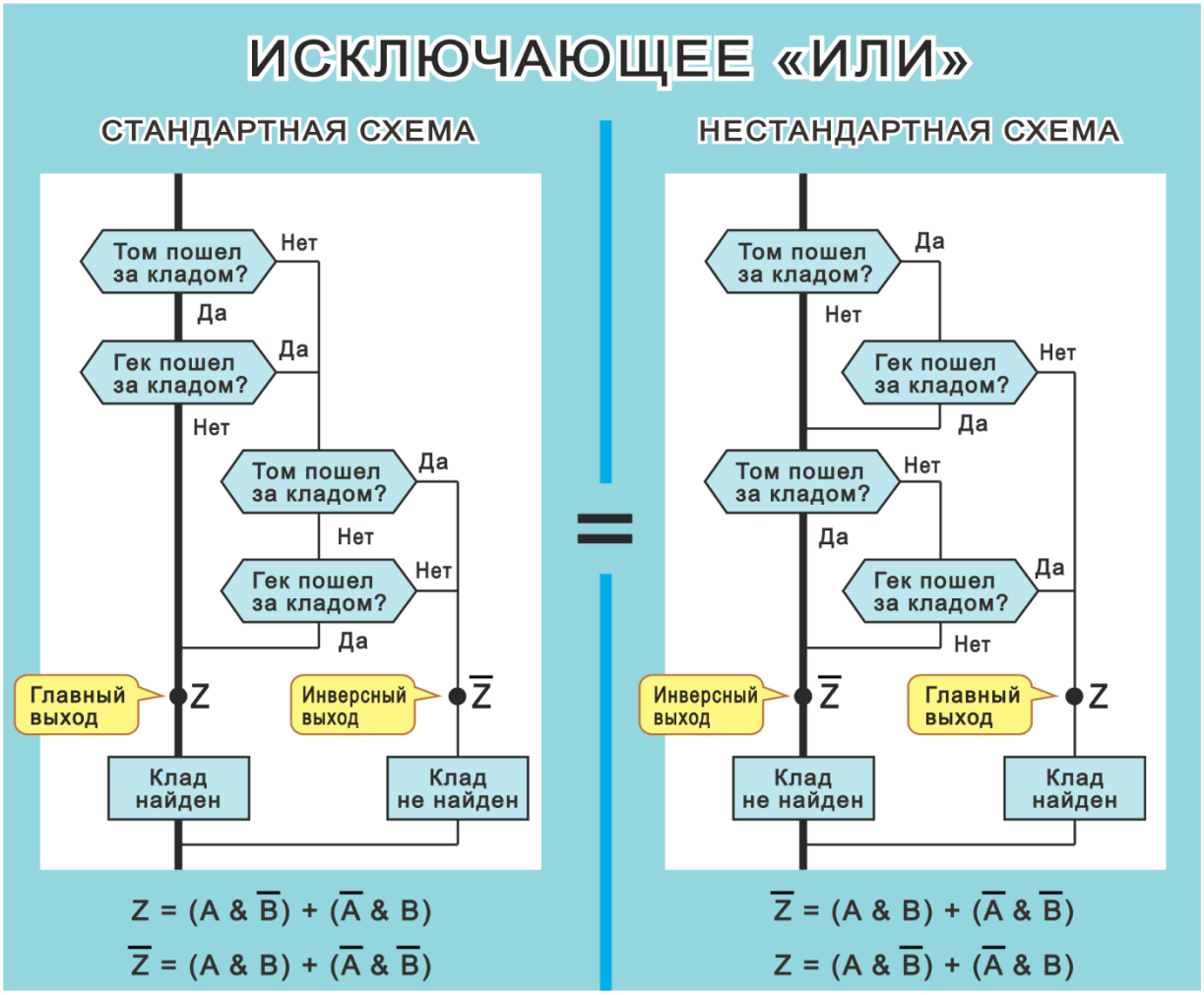


Рис. 104. Стандартная схема «Исключающее ИЛИ»

Рис. 105. Нестандартная схема «Исключающее ИЛИ»

ПРИВЕДЕНИЕ ЛОГИЧЕСКОЙ ФУНКЦИИ «ИСКЛЮЧАЮЩЕЕ ИЛИ» К КАНОНИЧЕСКОМУ ВИДУ

Приведение дракон-схемы к каноническому виду показано на рис. 106.

Исходная схема (вверху слева) содержит сложную функцию «Исключающее ИЛИ»: $(A \,\&\, \overline{B}) \vee (\overline{A} \,\&\, B)$. Здесь мы видим три кандидатуры на удаление: логические связки «ИЛИ», «И», «НЕ».

Начнем чистку со связки «ИЛИ». Убрав знак V и скобки, создаем две иконы Вопрос и получаем вторую схему (вверху, в центре).

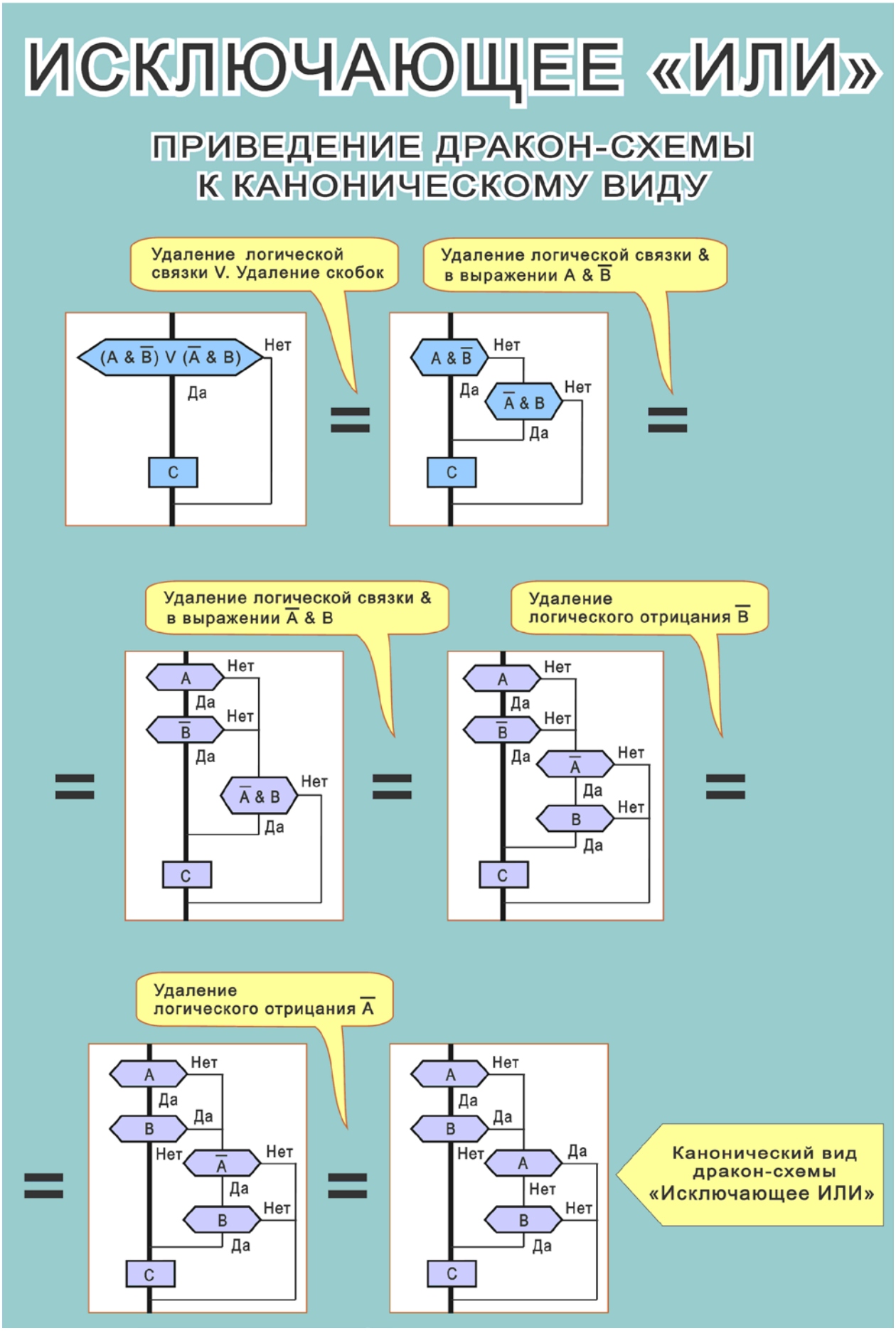


Рис. 106. Исключающее ИЛИ. Цепочка равносильных преобразований позволяет привести дракон-схему к каноническому виду

После этого стираем амперсанд & у члена $A \,\&\, \overline{B}$, размножаем иконы Вопрос и создаем третью схему (в центре, слева).

Затем устраняем последний амперсанд у члена $\overline{A} \,\&\, B$, снова удваиваем иконы Вопрос и рисуем четвертую схему (в центре, справа).

Осталось самое простое — удалить знаки отрицания. Сначала выбрасываем верхнюю черту у члена $\overline{B}$, меняем местами Да и Нет и получаем пятую схему (внизу слева).

Наконец, убираем отрицание у члена $\overline{A}$, снова переставляем Да и Нет и получаем искомую каноническую схему «Исключающее ИЛИ» (внизу справа).

Как и было намечено, в канонической схеме нет ни одной логической связки — они полностью исчезли. Тем не менее, каноническая схема выполняет все положенные логические функции, что достигается с помощью графики.

ПРОСТАЯ СХЕМА ДЛЯ СТРОГОЙ ДИЗЪЮНКЦИИ

Каноническая схема «Исключающее ИЛИ» получилась довольно сложной. Нельзя ли ее упростить?

Можно убрать четыре иконы Вопрос, заменив их на одну. Но придется использовать знак строгой дизъюнкции, например, $\dot{\vee}$ (знак v с точкой наверху).

$$A \,\dot{\vee}\, B$$

Запись в рамке означает строгую дизъюнкцию логических переменных A и B.

На рисунке 107 представлены две схемы с таким значком. Слева содержательная схема, продолжающая рассказ про Тома и Гека. Знак строгой дизъюнкции связывает две переменные:

- Том пошел за кладом.
- Гек пошел за кладом.

Смысл в том, что одна переменная (неважно какая) равна 1, а другая равна 0. Короче, значения переменных не должны совпадать; они должны быть разными.

Схема справа носит абстрактный характер; указанные переменные теряют связь с Томом и Геком и обозначены буквами A и B.

Существует много разных значков для строгой дизъюнкции: $\neq$, $\oplus$, XOR и т.д. Обсуждение этих значков (почему они такие, а не иные) не входит в нашу задачу.

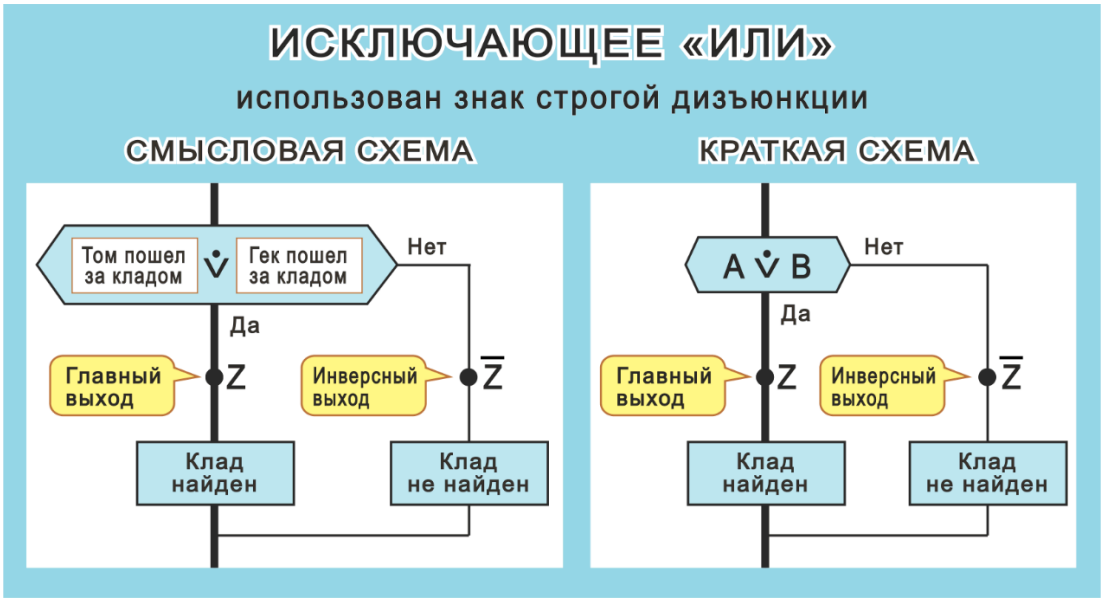


Рис. 107. Исключающее ИЛИ. Смысловая и краткая схема со знаком строгой дизъюнкции

ВЫВОДЫ

1. Строгая дизъюнкция двух переменных (исключающее ИЛИ) есть логическая функция $(\mathrm{A} \,\&\, \overline{\mathrm{B}}) \vee (\overline{\mathrm{A}} \,\&\, \mathrm{B})$.
2. Инверсный выход строгой дизъюнкции описывается формулой $\mathrm{A} \,\&\, \mathrm{B} \vee \overline{\mathrm{A}} \,\&\, \overline{\mathrm{B}}$.
3. Переключатель маршрутов алгоритма есть логическая функция, записанная в иконе Вопрос (или во фрагменте) и принимающая значение 1 или 0.
4. Если переключатель логического фрагмента равен 1, бегунок алгоритма движется через главный выход; в противном случае — через инверсный выход
5. В классической алгебре логики используется линейная запись логической функции «Исключающее ИЛИ», которая не показывает свою инверсию и принципиально не может описать инверсный маршрут.
6. В неклассической алгебре логики применяется графическая запись логической функции «Исключающее ИЛИ», которая изображает оба выхода: и главный, и инверсный.
7. В канонической схеме строгой дизъюнкции логические связки отсутствуют.
8. Некоторые маршруты в алгоритмах могут оказаться запрещенными.

Глава 18

СЛОЖНЫЕ ЛОГИЧЕСКИЕ ФУНКЦИИ

ВИЗУАЛИЗАЦИЯ СЛОЖНОЙ ЛОГИЧЕСКОЙ ФУНКЦИИ

Рассмотрим функцию

$$Z = (A \,\&\, \overline{B} \,\&\, C) \vee (D \,\&\, E \,\&\, \overline{F}) \qquad (1)$$

На рис. 108 показан визуальный способ записи этой функции. Рисунок делится на две зоны. В верхней зоне размещена формула (1), в нижней — показан дракон-алгоритм, реализующий данную логическую функцию.

На рисунке видно, что формула (1) разбивается на три части:

- конъюнкт $A \,\&\, \overline{B} \,\&\, C$,
- конъюнкт $D \,\&\, E \,\&\, \overline{F}$,
- операция «ИЛИ», соединяющая два конъюнкта.

Указанные три части подробно разрисованы в нижней зоне.

Функция $A \,\&\, \overline{B} \,\&\, C$ изображена в верхнем белом квадрате в виде трех икон A, B, C, расположенных на шампуре. Это первая функция «И», представляющая собой конъюнкцию трех переменных.

В формуле $A \,\&\, \overline{B} \,\&\, C$ два члена записаны без логического отрицания, третий — с отрицанием. Члены без отрицания превращаются на чертеже в иконы A и C, у которых нижний выход помечен словом «Да». Член с отрицанием изображается по-другому — у иконы B нижний выход помечен словом «Нет» (рис. 108).

В нижнем белом квадрате показана вторая функция «И», описывающая конъюнкцию трех переменных $D \,\&\, E \,\&\, \overline{F}$. Она изображена в виде трех икон D, E, F, нанизанных на одну вертикаль. Здесь тоже имеется особенность — в состав конъюнкции входит инверсия переменной $\overline{F}$. Это обстоятельство, разумеется, должно быть отражено на графике. Поэтому выходы иконы F, подчиняясь правилу инверсии, поменялись местами.

Мы подошли к самому интересному месту. Две функции «И» ($A \,\&\, \overline{B} \,\&\, C$ и $D \,\&\, E \,\&\, \overline{F}$) должны быть соединены между собой с помощью операции «ИЛИ».

Возникает вопрос: как изобразить функцию «ИЛИ» на дракон-схеме? Оказывается, это можно сделать с помощью самой обычной точки. Найдите точку K на рис. 108 — это и есть операция «ИЛИ».

Точка K должна соединить два результата, две функции «И». Где находится результат вычисления первой функции «И»? Он показан в верхнем белом квадрате в нижней точке шампура. А где результат второй функции «И»? В нижнем белом квадрате в нижней точке вертикали. Эти-то два пункта и нужно связать между собой с помощью точки K. Как это сделать? Очень просто — с помощью линий, соединяющих нижние выходы икон C и F и замыкающих их в точке K. Это значит, что точка K реализует функцию «ИЛИ».

Точка K играет еще одну роль — она является главным выходом логического фрагмента, который реализует функцию $Z = (A \,\&\, \overline{B} \,\&\, C) \vee (D \,\&\, E \,\&\, \overline{F})$. Справа от нее показан инверсный выход $\overline{Z}$. Можно считать, что два черных кружка K и Z — это одна и та же точка. Они разнесены всего лишь для наглядности — чтобы обеспечить удобное размещение на чертеже текстовых пояснений.

В самом низу показаны две иконы Действие: L и M. Они призваны подчеркнуть тот факт, что бегунок, проходящий по двум разным маршрутам (через главный и инверсный выходы) может выполнять разные действия.

ЛОГИЧЕСКИЙ ФРАГМЕНТ И ЕГО ФУНКЦИЯ

Важно установить связь между логическим фрагментом и логической функцией. Возникают две задачи.

- Задан логический фрагмент. Требуется написать логическую функцию, которую он выполняет.
- Задана логическая функция. Требуется нарисовать логический фрагмент, реализующий данную функцию.

Первая задача показана на рис. 109. В верхней зоне рисунка изображены два логических фрагмента. Под каждым фрагментом указана функция, которую он реализует. Поскольку фрагмент имеет два выхода (главный и инверсный), надо подчеркнуть: функция фрагмента есть функция главного выхода.

Рассмотрим левый пример на рис. 109. Какую логическую функцию он описывает? Возможны два ответа: «И» и «ИЛИ». Окончательный ответ зависит от расстановки знаков Z и $\overline{Z}$.

- Если буква Z стоит слева (обозначая главный выход), а $\overline{Z}$ справа (инверсный выход), перед нами стандартная схема «И» (независимо от расстановки слов «Да» и «Нет»).
- В противном случае (если $\overline{Z}$ слева, а Z справа), на чертеже представлена нестандартная схема «ИЛИ».

На рисунке 109 буква Z стоит слева. Поэтому функция описана как конъюнкция с помощью амперсандов.

$$Z = \overline{A} \,\&\, B \,\&\, C \,\&\, \overline{D} \,\&\, E \,\&\, \overline{F}$$

Давайте мысленно переставим букву Z и расположим ее справа, а $\overline{Z}$ слева. В таком случае функция Z превратится в дизъюнкцию.

$$Z = A \vee \overline{B} \vee \overline{C} \vee D \vee \overline{E} \vee F$$

Аналогичные рассуждения можно провести и для правого фрагмента.

УПРАЖНЕНИЯ. НАЙДИТЕ ФУНКЦИЮ ЛОГИЧЕСКОГО ФРАГМЕНТА

Отыскание логической функции фрагмента является увлекательной задачей. Попробуйте свои силы. Рисунок 109 можно рассматривать как подсказку.

На рис. 110 представлены три упражнения, которые помогут читателю закрепить материал.

Если вам логическая игра понравилась, попробуйте самостоятельно составить новые логические фрагменты и определить их функции.

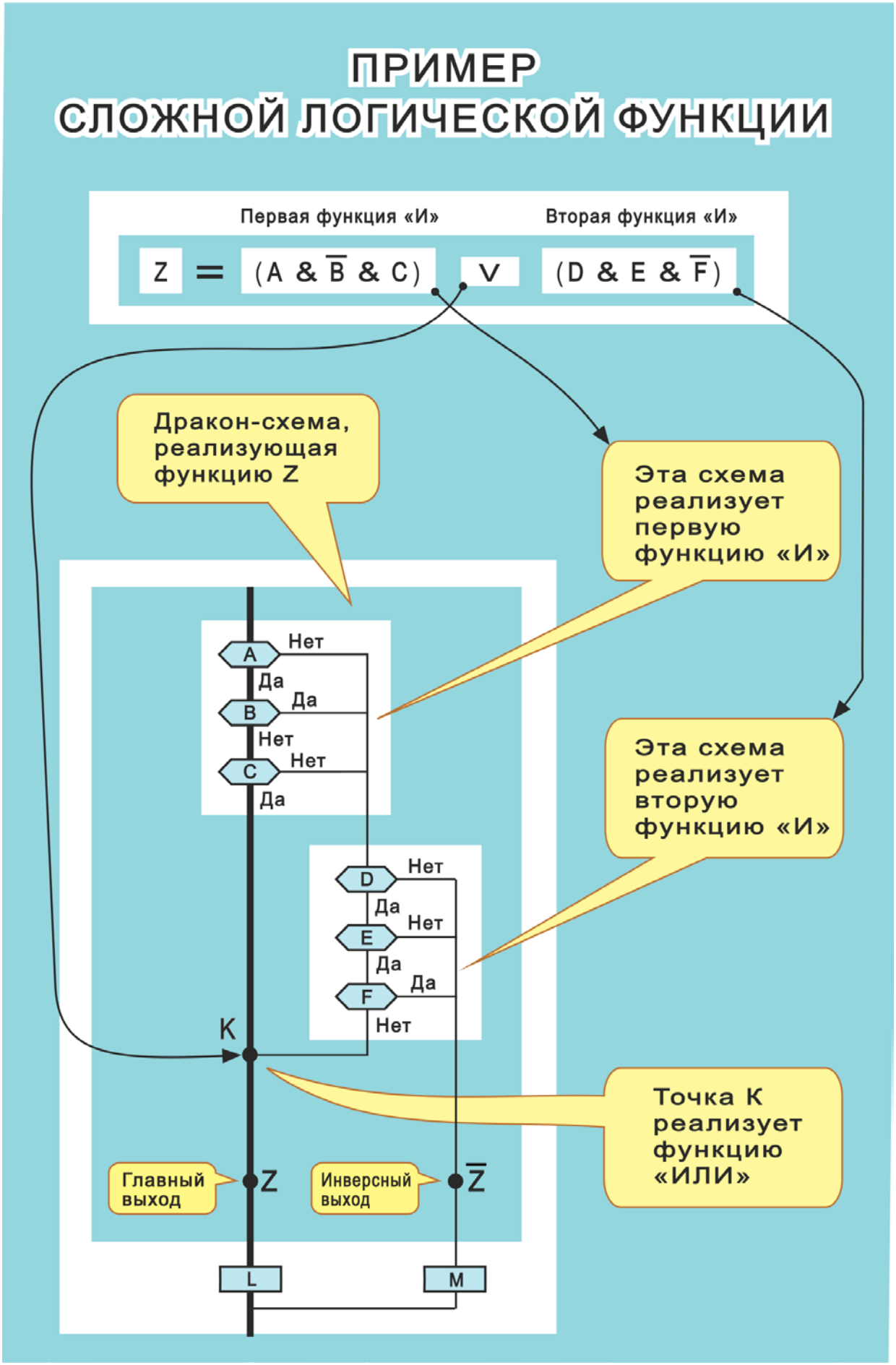


Рис. 108. Как нарисовать дракон-схему для сложной логической функции

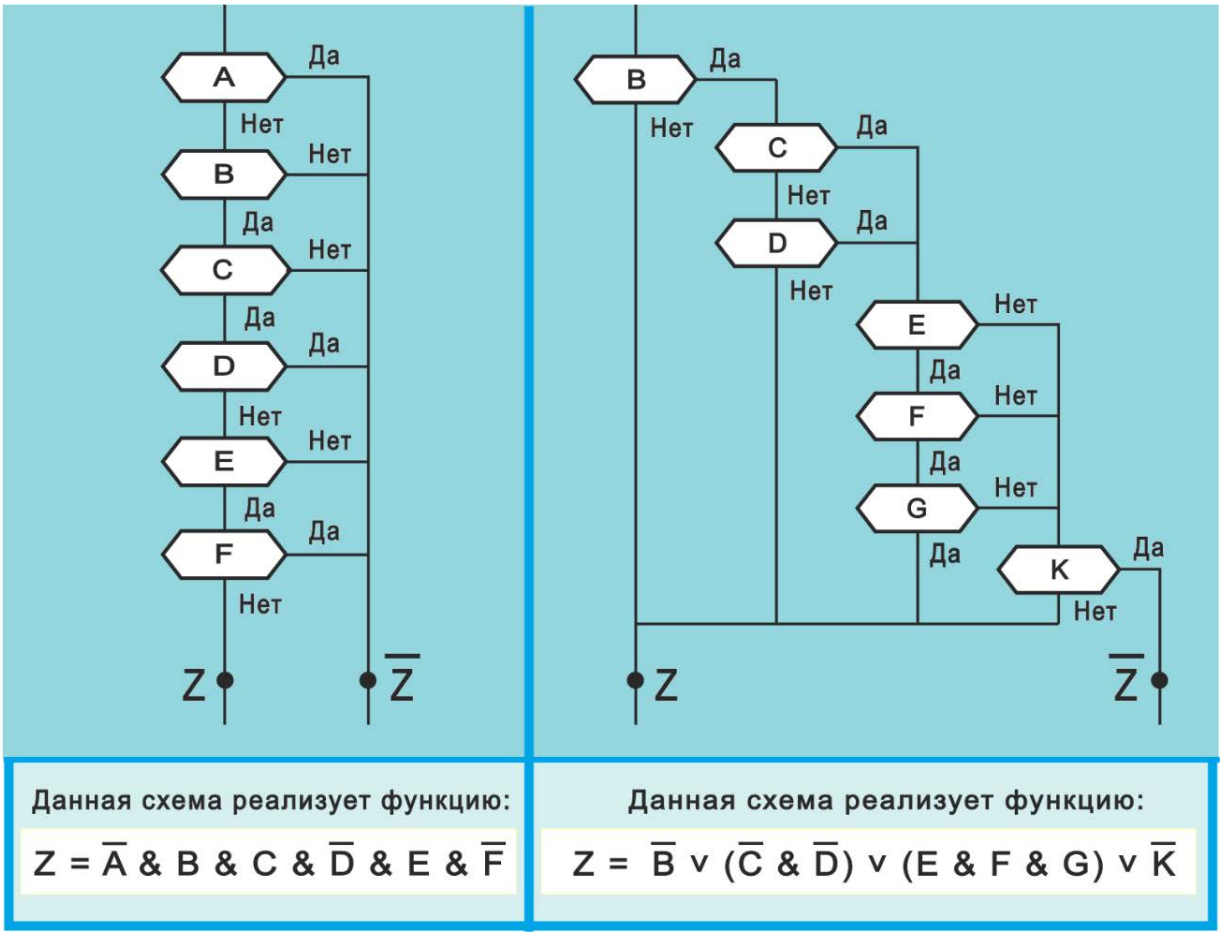


Рис. 109. Преобразование логического фрагмента в логическую функцию

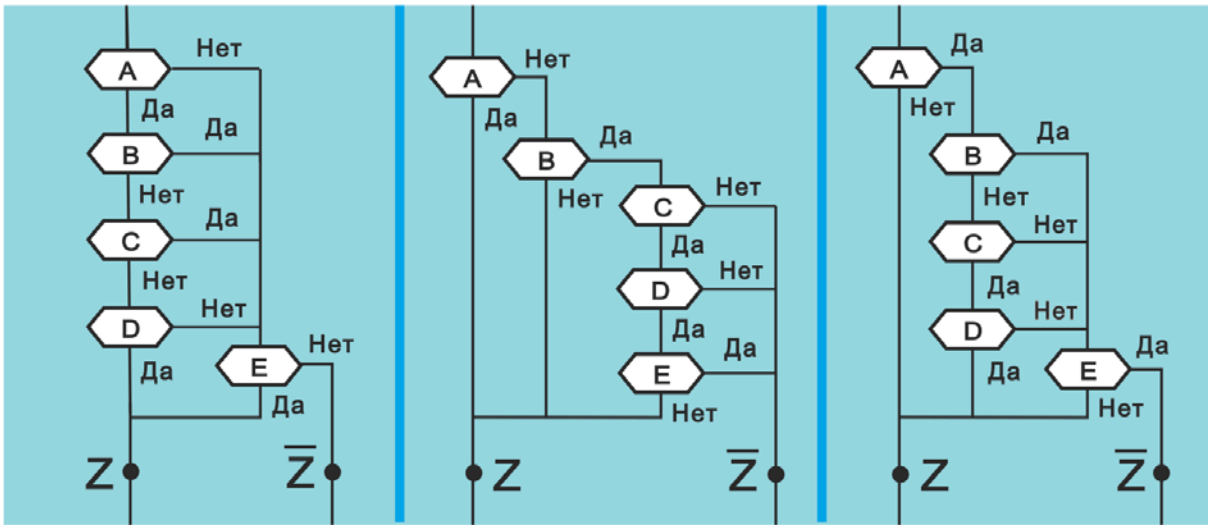


Рис. 110. Какую логическую функцию вычисляют эти фрагменты

КОГДА ЛУЧШЕ ИСПОЛЬЗОВАТЬ СТАНДАРТНУЮ СХЕМУ, А КОГДА — НЕСТАНДАРТНУЮ

На рис. 111 представлен алгоритм проверки электрических приборов — но не весь, а только его правая часть. Левая часть алгоритма опущена, о чем говорят две линии обрыва. Нас особенно интересует ветка, которая называется «Проверка оборудования». В ветке проверяются три прибора: синий, зеленый и желтый.

Если все три прибора исправны (нет ни одного отказа), то из всех икон Вопрос выходим через «Нет». Это хороший, штатный вариант развития событий, при котором запускается ветка «Проверка электросети». Далее проводится проверка трех электрических сетей (основной, резервной и слаботочной). После этого алгоритм благополучно завершается.

Теперь рассмотрим плохой, неблагополучный вариант. Вернемся еще раз к левой ветке. Предположим, что испортился (отказал) хотя бы один прибор из трех. Это значит, что хотя бы из одной иконы Вопрос выходим через «Да». Видим, что «Система неисправна» и вызываем процедуру «Выключение системы». Через икону Адрес «Завершение» переходим в последнюю ветку, где алгоритм заканчивает работу по причине неисправности системы (рис. 111).

Проведенный анализ говорит о том, что алгоритм построен правильно. Логических ошибок в нем нет. Однако нас интересует не только логика, но и эргономика. А конкретно — эргономика левой ветки. Выполняется ли в ней правило «Чем правее, тем хуже»?

Нет, это правило грубо нарушено!

В самом деле, плохой маршрут, когда «Система неисправна», должен быть справа, а он находится слева. При этом хороший маршрут, когда все работает правильно, оказался справа. Это неверно. Налицо эргономическая ошибка. Чтобы ее исправить, нужно переставить местами две иконы Адрес: «Завершение» и «Проверка электросетей». Для этого следует сделать три рокировки в трех иконах Вопрос. Можно выразиться по-другому: необходимо превратить стандартную схему «ИЛИ» в нестандартную.

ПРАВИЛЬНАЯ СХЕМА

Исправленная схема показана на рис. 112. Чем же она отличается?

Отметим, что средняя и правая ветки не изменились. Они остались такими же, как были на рис. 111. Исправления коснулись только левой ветки.

Укажем перечень изменений.

1. Произведена рокировка трех икон Вопрос:
   - Отказ синего прибора.
   - Отказ зеленого прибора.
   - Отказ желтого прибора.
2. При рокировке произведена перестановка слов «Да» и «Нет» (в трех местах).
3. Вследствие рокировки поменялись местами две иконы Адрес: «Завершение» и «Проверка электросетей».

Укажем еще один недочет дракон-схемы на рис. 111. В силуэте желательно соблюдать правило.

Правило расположения икон Адрес	Порядок пространственного расположения икон Адрес (слева направо) должен соответствовать порядку выполнения веток во времени (слева направо).

Проверим, выполняется ли это правило на рис. 111.

Известно, что в штатном режиме сначала работает ветка «Проверка электросетей», и только после этого запускается ветка «Завершение». Следовательно, в том же самом порядке должны быть нарисованы две иконы Адрес ветки «Проверка электрооборудования». Слева должен находиться Адрес «Проверка электросетей», справа — «Завершение». Однако на рис. 111 они изображены ровно наоборот. Значит, на рисунке 111 указанное правило не соблюдается. Погрешность исправлена на рис. 112.

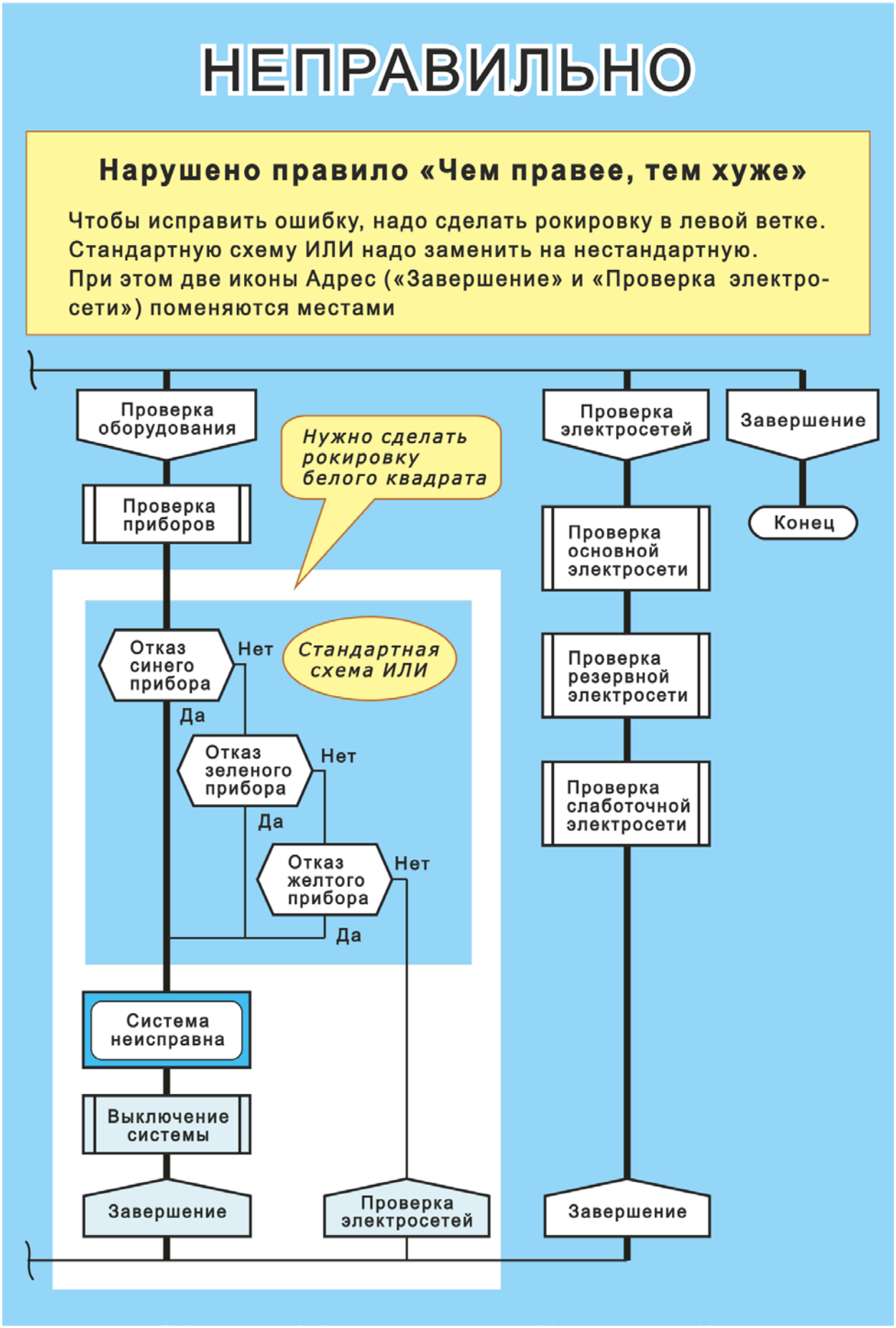


Рис. 111. Нарушено правило «Чем правее, тем хуже».
Чтобы исправить ошибку, в левой ветке надо сделать рокировку

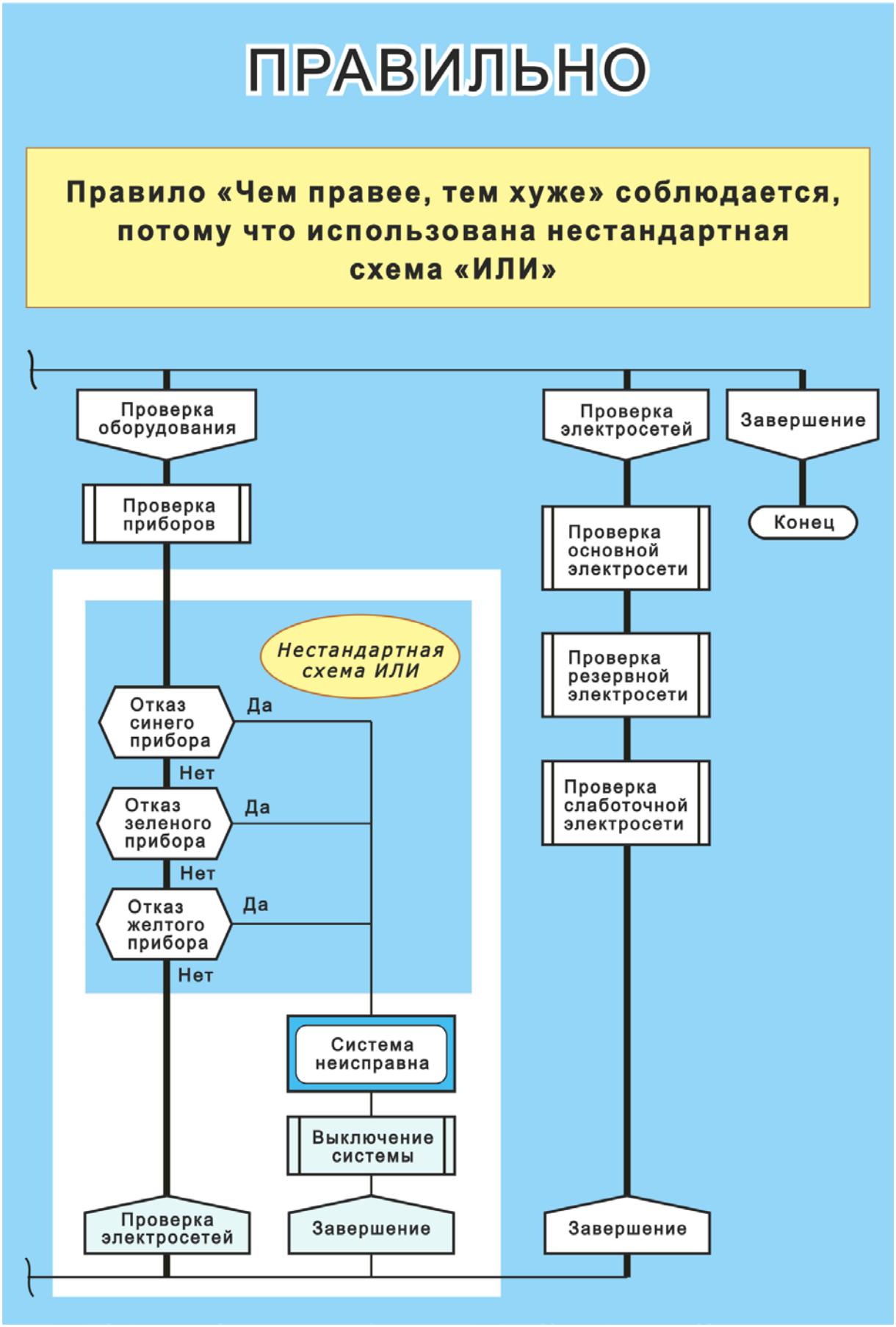


Рис. 112. Правильная схема. Все ошибки исправлены. Выполнено правило «Чем правее, тем хуже. Стандартная схема ИЛИ преобразована в нестандартную схему ИЛИ. Сравни с рис. 111

Вернемся к вопросу: «Когда следует использовать стандартную схему ИЛИ, а когда — нестандартную?» Ответ зависит от соблюдения правила «Чем правее, тем хуже». Если правило нарушено, нужно сделать рокировку в данной ветке. И заменить стандартную схему на нестандартную (или наоборот).

Рокировка делает разом два улучшения:

- Во-первых, устраняет ошибку и восстанавливает закон «Чем правее, тем хуже».
- Во-вторых, соблюдает правило «Порядок расположения Адресов должен соответствовать порядку работы веток».

Таким образом, мы показали, что замена стандартной схемы ИЛИ на нестандартную создает полезный эффект.

ВЫВОДЫ

1. Существует взаимно-однозначное соответствие между графической структурой логического алгоритма (логического фрагмента) и логической функцией, которую он реализует.
2. Зная логический фрагмент, можно написать его логическую функцию.
3. Зная логическую функцию, можно построить соответствующий ей логический фрагмент.
4. Использование графики в качестве нотации для записи сложных логических функций позволяет удалить логические связки, улучшить понимание и сократить количество ошибок.

Часть 4

АЛГОРИТМЫ РЕАЛЬНОГО ВРЕМЕНИ

Глава 19

ОПЕРАТОРЫ РЕАЛЬНОГО ВРЕМЕНИ

Можно ли в алгоритмах изображать время? Как это сделать? В данной главе мы узнаем об этом.

АЛГОРИТМ «УПРАВЛЕНИЕ СВЕТОФОРОМ»

Давайте посмотрим, как работает светофор. Самый обычный светофор, который стоит на перекрестке и регулирует уличное движение. Алгоритм управления светофором показан на рис. 113.

Шапка алгоритма представлена на рис. 114. Как известно, шапка позволяет получить ответ на три вопроса:

- Как называется задача?
- Из скольких частей она состоит?
- Как называется каждая часть?

Вот ответы для рис. 113.

1) Как называется задача? (Читаем Заголовок алгоритма).
   - Управление светофором.
2) Из скольких частей она состоит? *(Считаем иконы «Имя ветки»).*
   - Из трех.
3) Как называется каждая часть? *(Читаем текст в иконах «Имя ветки»).*
   - Управление зеленым светом.
   - Управление красным светом.
   - Ночной режим.

Первая ветка начинается с команды ВКЛЮЧИ ЗЕЛЕНЫЙ (имеется в виду «Включи зеленый сигнал светофора»).

Вторая команда — икона Пауза. Она изображается перевернутой трапецией. Команда Пауза отсчитывает время, записанное внутри иконы. Когда отсчет времени закончен, запускается следующая (после паузы) команда.

Рассмотрим три команды, записанные в первой ветке:

- ВКЛЮЧИ ЗЕЛЕНЫЙ.
- Пауза 2 минуты.
- ВЫКЛЮЧИ ЗЕЛЕНЫЙ.

Что означают эти команды? Они говорят, что зеленый свет будет гореть 2 минуты, а затем погаснет.

Рассмотрим следующие три команды:

- ВКЛЮЧИ ЖЕЛТЫЙ.
- Пауза 10 секунд.
- ВЫКЛЮЧИ ЖЕЛТЫЙ.

Смысл этой тройки команд очевиден. Желтый свет будет гореть 10 секунд, затем погаснет.

Вторая ветка называется «Управление красным светом».

Прочитаем три команды во второй ветке:

- ВКЛЮЧИ КРАСНЫЙ.
- Пауза 2 минуты.
- ВЫКЛЮЧИ КРАСНЫЙ.

Это значит: красный свет будет гореть ровно 2 минуты.

Читаем дальше:

- ВКЛЮЧИ ЖЕЛТЫЙ.
- Пауза 10 секунд.
- ВЫКЛЮЧИ ЖЕЛТЫЙ.

Смысл ясен: желтый свет будет гореть 10 секунд, после чего погаснет.

Обобщим сказанное и опишем последовательность смены сигналов светофора:

- зеленый горит 2 минуты;
- желтый — 10 секунд;
- красный — 2 минуты;
- желтый — 10 секунд.

После этого последовательность все время повторяется.

Алгоритм на рис. 113 работает в двух режимах, которые постоянно чередуются:

- дневной режим;
- ночной режим.

Ночной режим описан в третьей ветке. Ночью красный и зеленый сигналы светофора отключены. Вместо них всю ночь мерцает желтый мигающий.

На рис. 113 в иконах Пауза использована избыточная запись:

- Пауза 2 минуты.
- Пауза 10 секунд.

Все это можно записать короче. Слово «Пауза» обычно опускают. Вместо «10 секунд» пишут «10с» и т.д.

БЕСКОНЕЧНЫЕ АЛГОРИТМЫ

Алгоритмы, которые мы рассматривали во всех предыдущих главах, обязательно имели конец. Проще говоря, они непременно заканчивались, то есть прекращали работу с помощью иконы Конец.

Однако на рис. 113 икона Конец отсутствует. Она не нужна, так как в данном случае мы имеем дело с бесконечным алгоритмом.

Разумеется, прекращение работы алгоритма возможно. Оно происходит в результате внешней причины. Например, при выключении питания системы, при поломке оборудования и т.д.

Мы убедились, что алгоритм, управляющий светофором, не имеет конца. Он работает круглосуточно. Потому что дорожное движение — бесконечный процесс. Который, как и сама жизнь, никогда не останавливается.

Бесконечный алгоритм

- Это алгоритм, который работает все время, круглосуточно.
- В таком алгоритме предусмотрен бесконечный цикл. Выход из бесконечного цикла не предусмотрен.

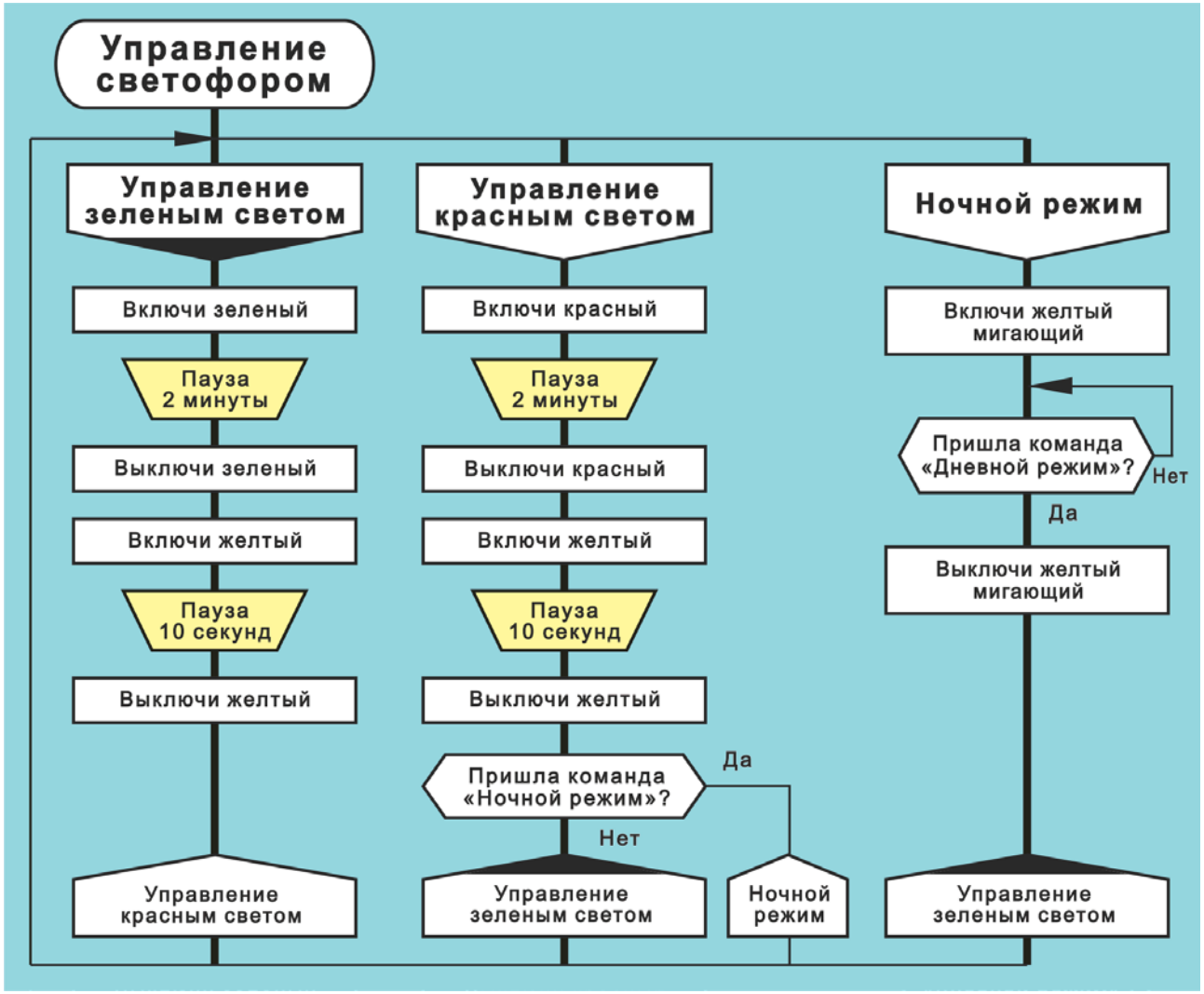


Рис. 113. Алгоритм «Управление светофором»

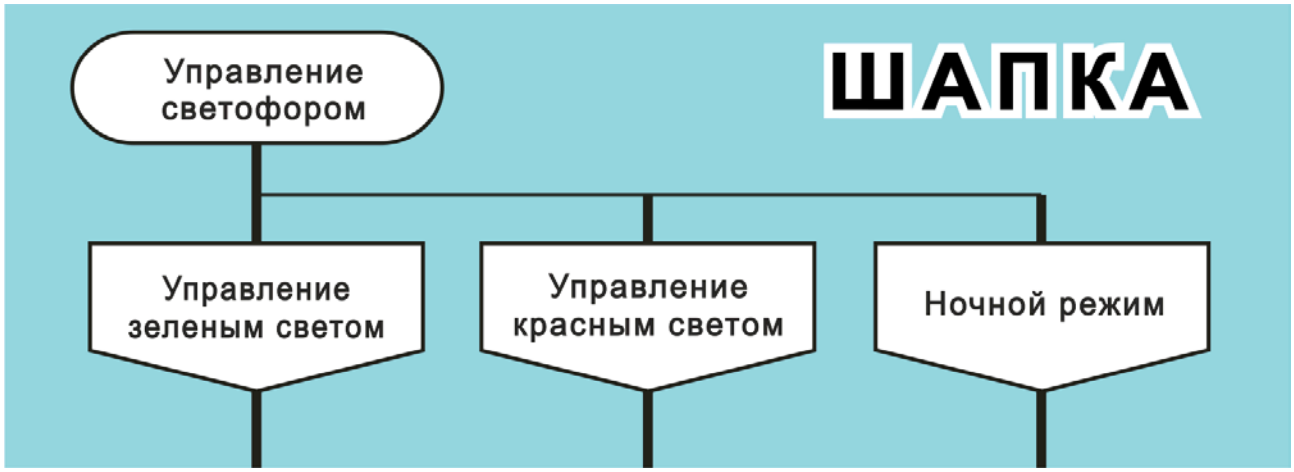


Рис. 114. Шапка алгоритма «Управление светофором». Показаны икона Заголовок и три иконы «Имя ветки»

Что такое оператор

- Это инструкция (команда), которая исполняет код программы.
- По-английски *statement.*

СПИСОК ОПЕРАТОРОВ РЕАЛЬНОГО ВРЕМЕНИ

В языке ДРАКОН имеется пять икон реального времени (рис. 19 п.18-20; рис. 20, п. 21, 28):

- Пауза.
- Период.
- Таймер.
- Синхронизатор.
- Параллельный процесс.

Три из них (Пауза, Таймер и Параллельный процесс) — простые операторы. Две другие иконы (Период и Синхронизатор) служат «кирпичиками» для построения составных операторов и вне последних не используются.

Икона Период является принадлежностью цикла ЖДАТЬ (рис. 21 макроикона 12). Икона Синхронизатор служит для образования десяти составных операторов (рис. 21, макроиконы 14-23).

Назначение операторов поясним, как всегда, на примерах.

ОПЕРАТОРЫ ВВОДА-ВЫВОДА

В языке ДРАКОН предусмотрены четыре оператора ввода-вывода (рис. 19 п. 14-17):

- Вывод.
- Простой вывод.
- Ввод.
- Простой ввод.

На рис. 19 видно, что иконы ввода-вывода имеют мнемоническую форму. Иконы 14 и 15 содержат полую стрелку, направленную наружу, что символизирует Вывод.
Иконы 16 и 17 имеют стрелку, направленную внутрь, что значит Ввод

Ввод и Вывод — «двухэтажные» операторы. На верхнем этаже пишется ключевое слово или ключевая фраза (например, «Выдать команду» — см. рис. 115, 116). На нижнем (в прямоугольнике) — содержательная информация, подлежащая вводу и выводу.

В простейших случаях, где «два этажа» не нужны, используют экономичные иконы «Простой ввод» и «Простой вывод» (полые стрелки).

ВЫДАЧА УПРАВЛЯЮЩИХ КОМАНД

Предположим, управляющий компьютер должен выдать серию электрических команд, которые по линиям связи передаются в исполнительные органы и вызывают срабатывание электромеханических реле. В результате происходит открытие трубопровода, включение насоса и другие операции, необходимые для функционирования управляемого объекта.

Такая ситуация может встретиться во многих системах управления реального времени. Например, при заправке топливом космических ракет, на атомных электростанциях, нефтеперерабатывающих заводах и т. д.

Рассмотрим пример. Предположим, управляющий компьютер должен:

- выдать команду ОТКРЫТЬ.ТРУБОПРОВОД;
- подождать две минуты;

- выдать две команды: ВКЛЮЧИТЬ.НАСОС и ОТКРЫТЬ.ЗАСЛОНКУ;
- подождать 45 секунд;
- выдать команду ПОДАЧА.ТОПЛИВА;
- подождать три минуты;
- выдать команду ПУСК.АГРЕГАТА.

Соответствующий алгоритм представлен на рис. 115.

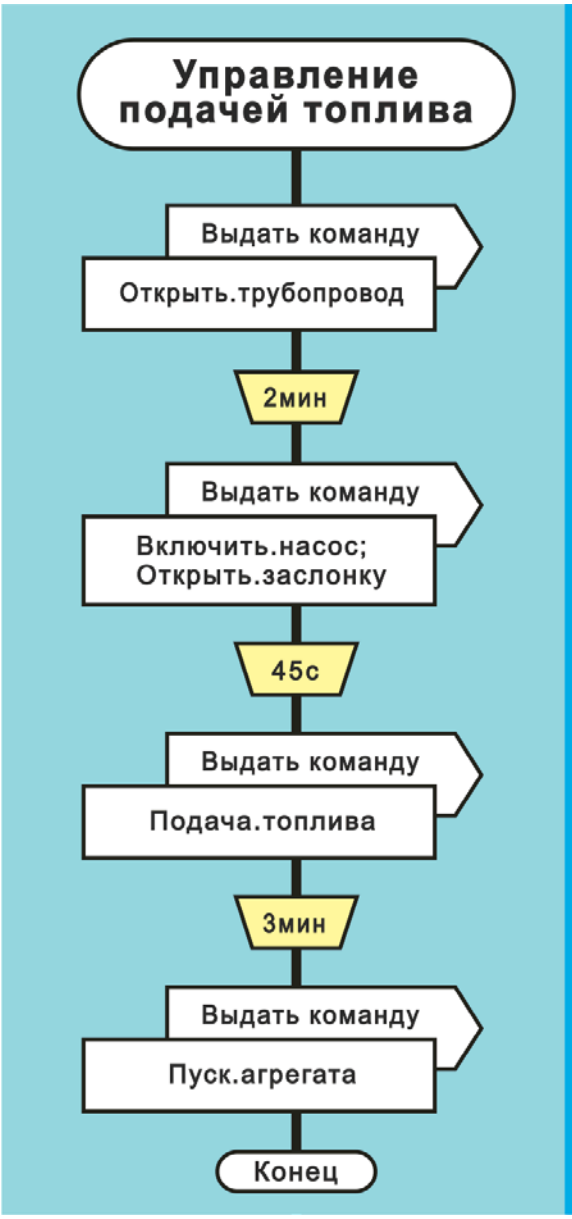


Рис. 115. Пример использования оператора Пауза

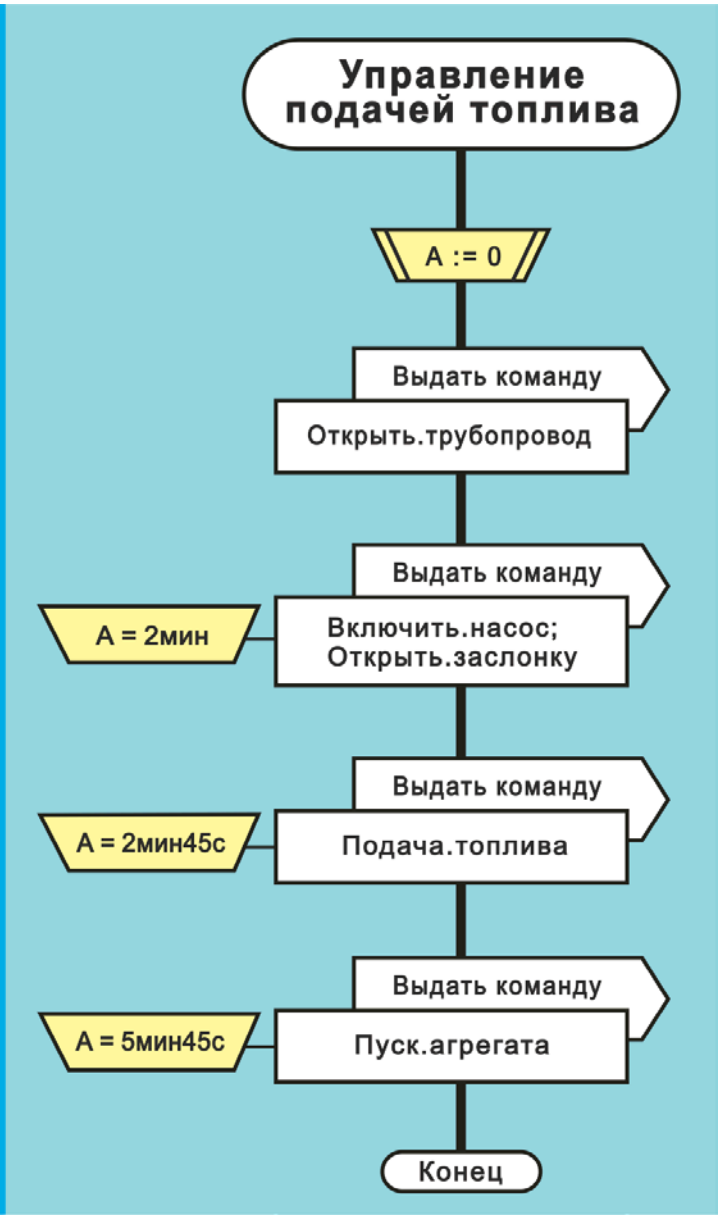


Рис. 116. Пример использования операторов Таймер и Синхронизатор

ОПЕРАТОР «ПАУЗА»

На рис. 115 задержка выдачи команд реализуется с помощью иконы Пауза. Внутри последней пишут время необходимой задержки. Например, 2 мин (2 минуты), 45 с (45 секунд) и т. д.

Если говорить более точно, верхний оператор Пауза на рис. 115 работает так. После выдачи команды ОТКРЫТЬ.ТРУБОПРОВОД в управляющем компьютере или в

контроллере запускается программный счетчик времени на 2 минуты. По истечении этого времени компьютер выдает в линию связи команды ВКЛЮЧИТЬ.НАСОС и ОТКРЫТЬ.ЗАСЛОНКУ.

ОПЕРАТОРЫ «ТАЙМЕР» И «СИНХРОНИЗАТОР»

Вернемся еще раз к задаче на рис. 115 и слегка изменим ее. Будем считать, что разработчик управляемого объекта хочет указать время выдачи команд не по принципу «задержка после предыдущей команды», а по принципу секундомера. Это значит, что все времена отсчитываются от единого начального момента (совпадающего с пуском секундомера).

Исходя из этого, сформулируем задачу управляющего компьютера. Он должен:

- включить секундомер, то есть обнулить и запустить таймер;
- выдать команду ОТКРЫТЬ.ТРУБОПРОВОД;
- когда таймер отсчитает две минуты, выдать пару команд ВКЛЮЧИТЬ.НАСОС и ОТКРЫТЬ.ЗАСЛОНКУ;
- когда таймер отсчитает 2 минуты 45 секунд, выдать команду ПОДАЧА.ТОПЛИВА;
- когда таймер отсчитает 5 минут 45 секунд, выдать команду ПУСК. АГРЕГАТА.

Описанный алгоритм изображен на рис. 116. В нем используются операторы Таймер и Синхронизатор, совместная работа которых обеспечивает нужный эффект.

Например, синхронизатор *A* = 2мин 45с задерживает выдачу команды ПОДАЧА.ТОПЛИВА до момента, когда таймер *A* отсчитает 2 минуты 45 секунд.

Сравнивая алгоритмы на рис. 115 и 116, можно заметить, что они почти эквивалентны. Почему почти?

Чтобы разобраться, рассмотрим идеальный случай. Представим, что время, необходимое для выдачи одной команды равно нулю. Имеется в виду, что перечисленные ниже команды выдаются за время, равное нулю:

- ОТКРЫТЬ.ТРУБОПРОВОД;
- ВКЛЮЧИТЬ.НАСОС;
- ОТКРЫТЬ.ЗАСЛОНКУ;
- ПОДАЧА.ТОПЛИВА;
- ПУСК.АГРЕГАТА.

В этом случае оба алгоритма будут выдавать команды синхронно.

Однако в действительности идеальные случаи встречаются редко, ибо время выдачи одной команды больше нуля. Поэтому наши алгоритмы работают несинхронно.

Практика показывает, что в некоторых ситуациях предпочтительным является принцип паузы (когда используется икона Пауза). А в других — принцип таймера (когда используются иконы Таймер и Синхронизатор).

Оба инструмента оказываются необходимыми и полезными.

Оператор Таймер	Порождает, обнуляет и запускает таймер и присваивает ему имя, например *A*
Оператор Синхронизатор	Задерживает выполнение размещенной справа от него иконы до момента, указанного в иконе Синхронизатор

ОПЕРАТОР «ЖДАТЬ»

Вернемся на минутку во вторую главу к рассказу о попугаях. На рис. 14 можно найти кусочек, который мы извлекли и перенесли на рис. 117. Это и есть цикл ЖДАТЬ.

Однако здесь есть подвох. Дело в том, что на рис. 117 представлена некорректная запись, пригодная только для жизнеритма. Но не для алгоритма!

Недочет в том, что в иконе Период написано слово «Жди». Ошибка исправлена в алгоритме на рис. 118, где указана точная длительность периода 32мс (32 миллисекунды).

Рис. 117. Цикл ЖДАТЬ. Нестрогая запись, допустимая в жизнеритме и запрещенная в алгоритме

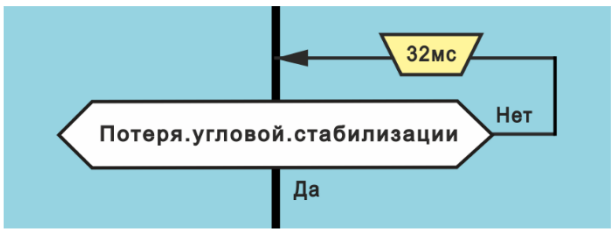


Рис. 118. Цикл ЖДАТЬ. Строгая запись используемая в алгоритме

ПРИЗНАК АВАРИИ РАКЕТЫ

Потеря угловой стабилизации — это признак аварии космической ракеты. Признак может принимать два значения 0 и 1.

Если признак равен 0, значит все хорошо, ракета в полном порядке и успешно летит к цели. Если же признак равен 1, значит, произошла авария, ракета отклонилась от заданной траектории и летит неизвестно куда.

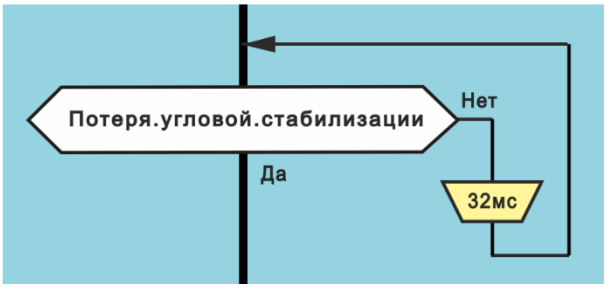


Рис. 119. Цикл ЖДАТЬ. Нерекомендуемая схема

Поэтому нужно тщательно следить, не появился ли опасный признак ПОТЕРЯ.УГЛОВОЙ.СТАБИЛИЗАЦИИ. Именно этим занимается цикл ЖДАТЬ на рис. 118. Каждые 32 миллисекунды производится опрос признака. Если признак появился, бегунок покидает цикл ЖДАТЬ через выход «Да». После этого нужно выполнить неотложные аварийные операции.

Обозначим отрезок времени через *Δt.* Цикл ЖДАТЬ регулярно повторяет опрос признака через время *Δt.* Величина *Δt* называется периодом опроса и записывается в иконе Период. В нашем примере период *Δt* = 32мс.

НЕРЕКОМЕНДУЕМАЯ СХЕМА

Легко сообразить, что алгоритмы на рис. 118 и 119 в точности совпадают. На рис. 118 использована икона Период, а на рис. 119 — икона Пауза. Однако эти иконы выполняют в точности одинаковую функцию.

Нет необходимости изображать цикл ЖДАТЬ двумя разными способами. Поэтому рис. 118 считается правильным, а рис. 119 — нерекомендуемым и запрещенным. Более того, если пользователь попытается создать цикл ЖДАТЬ, как на рис. 119, программа «ДРАКОН-конструктор» автоматически исправит ошибку и преобразует цикл в форму рис. 118.

ИКОНА «ПЕРИОД»

Сравнивая макроиконы 4 и 12 на рис. 21 (цикл со стрелкой и цикл ЖДАТЬ), мы видим, что они очень похожи. Поэтому во избежание путаницы нужно иметь какой-то различительный признак. Эту функцию выполняет икона Период. Если она есть в петле цикла — перед нами цикл ЖДАТЬ. Если нет — цикл со стрелкой.

Человек, который стоит на остановке и ждет трамвая, воспринимает ожидание как нечто непрерывное. Однако алгоритм реального времени организует ожидание как дискретный процесс и запускает цикл ЖДАТЬ периодически. Отсюда вытекает, что период — важная характеристика цикла ЖДАТЬ.

Как работает оператор «Период»? На рис. 119 цикл ЖДАТЬ «крутится» по своей петле с периодичностью 32 миллисекунды, пока не выполнится условие в иконе Вопрос. После чего произойдет выход из цикла. Таким образом, оператор «Период» задает период повторения цикла ЖДАТЬ.

ЦИКЛ «ЖДАТЬ» И ОПЕРАТОР «ТАЙМЕР»

Задача. Включить главный двигатель. Если через 3 минуты он не включится, следует включить резервный двигатель

Для решения задачи можно использовать цикл ЖДАТЬ и оператор Таймер (рис. 120).

В начале алгоритма выдается команда ВКЛЮЧИ ГЛАВНЫЙ ДВИГАТЕЛЬ. Затем в иконе Таймер запускается таймер А.

Далее мы видим цикл ЖДАТЬ, работающий с периодом 1 секунда. В цепи обратной связи цикла размещена икона Вопрос с условием А > 3 минуты.

Что это значит? Цикл ЖДАТЬ проверяет два условия:

- Включен ли главный двигатель?
- Прошли ли 3 минуты?

Если главный двигатель включится в течение трех минут, алгоритм успешно завершает работу.

Если же, спустя три минуты, он не запустится, надо перейти на запасной двигатель. Бегунок выходит из правой иконы Вопрос через «Да» и выдается команда ВКЛЮЧИ РЕЗЕРВНЫЙ ДВИГАТЕЛЬ.

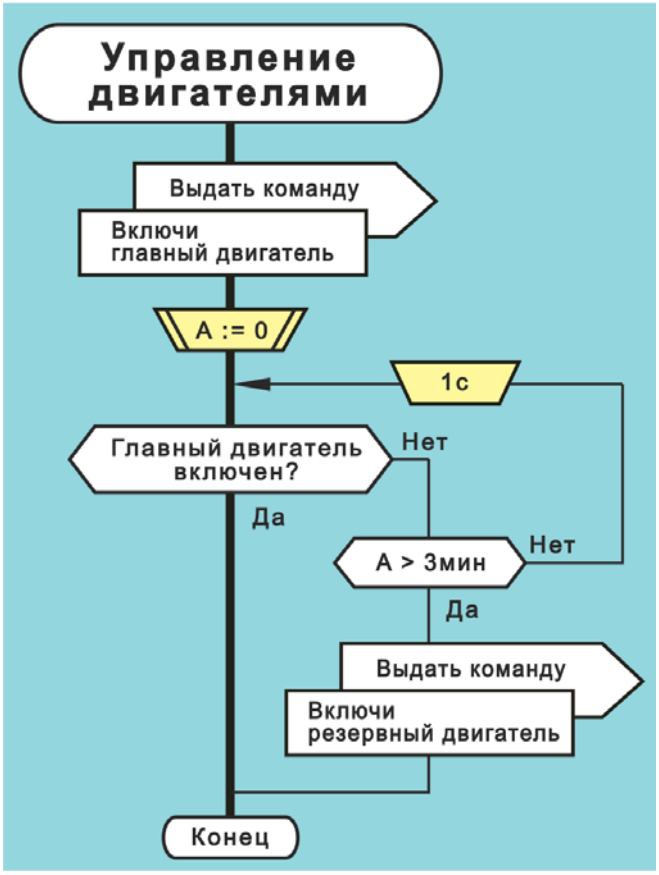


Рис. 120. Цикл ЖДАТЬ и икона Таймер

Таким образом, работу в реальном времени обеспечивают три элемента (рис. 120):

- Икона Таймер.
- Икона Период.
- Икона Вопрос с условием А > 3 мин.

ЦИКЛ «ЖДАТЬ» И ОПЕРАТОР «СИНХРОНИЗАТОР»

Предположим, мы ожидаем из космоса приветствие от пришельцев, посылающих нам сигнал Сигма (который мы ловим с помощью цикла ЖДАТЬ). Предположим также, что, получив желанный сигнал, мы должны послать в космос серию ответных сигналов, жестко привязанных по времени. Серия может быть длинной и насчитывать сто или даже тысячу кодограмм.

На рис. 121 представлено упрощенное решение, позволяющее отправить в космос серию из трех сигналов. Для этого нужны цикл ЖДАТЬ, икона Таймер и три иконы Синхронизатор. Когда таймер досчитает до 39-й минуты, мы пошлем пришельцам дружеский привет (рис. 121).

Данный рисунок демонстрирует совместное использование цикла ЖДАТЬ, таймера и синхронизаторов.

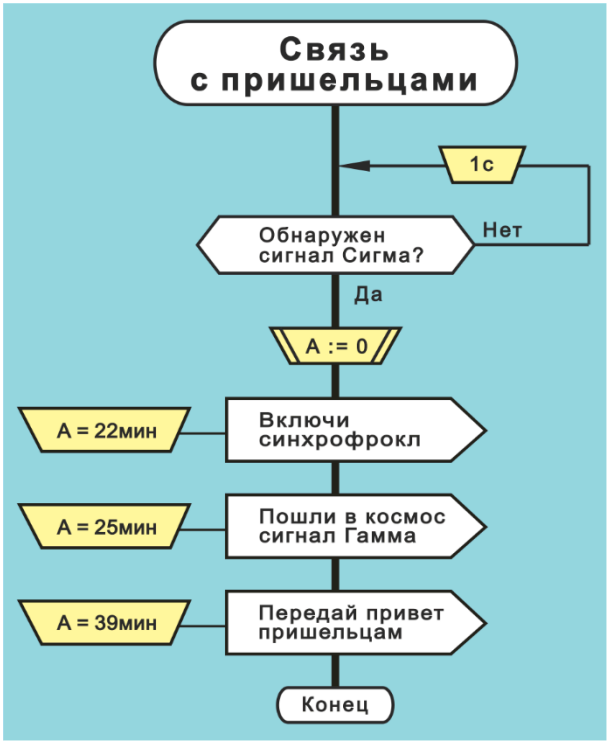


Рис. 121. Цикл ЖДАТЬ и три иконы «Синхронизатор»

ЦИКЛ «ЖДАТЬ» В ОБЩЕМ ВИДЕ

В общем виде цикл ЖДАТЬ показан на рис. 122. Он позволяет организовать режим ожидания признаков *B*, *C*, *D*, ..., *E*.

- Если первым появится признак *B*, выполняется действие *F*.
- Если *B* отсутствует и первым придет *C*, реализуется действие *G*.
- Если *B* и *C* отсутствуют и первым будет *D,* выполняется *H*. *И* так далее.
- Операторы *A* и *L* обычно не используются.

Задача ожидания нескольких признаков (когда система должна по-разному реагировать на каждый признак) является типичной при разработке систем реального времени.

Цикл ЖДАТЬ — удобное средство для решения подобных задач.

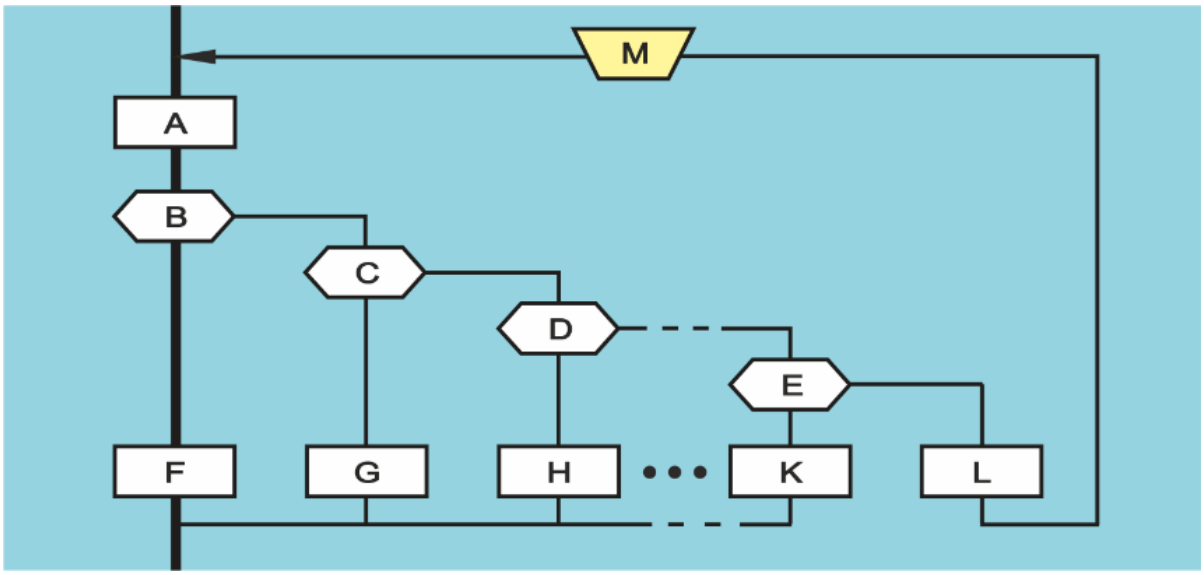


Рис. 122. Цикл ЖДАТЬ в общем виде

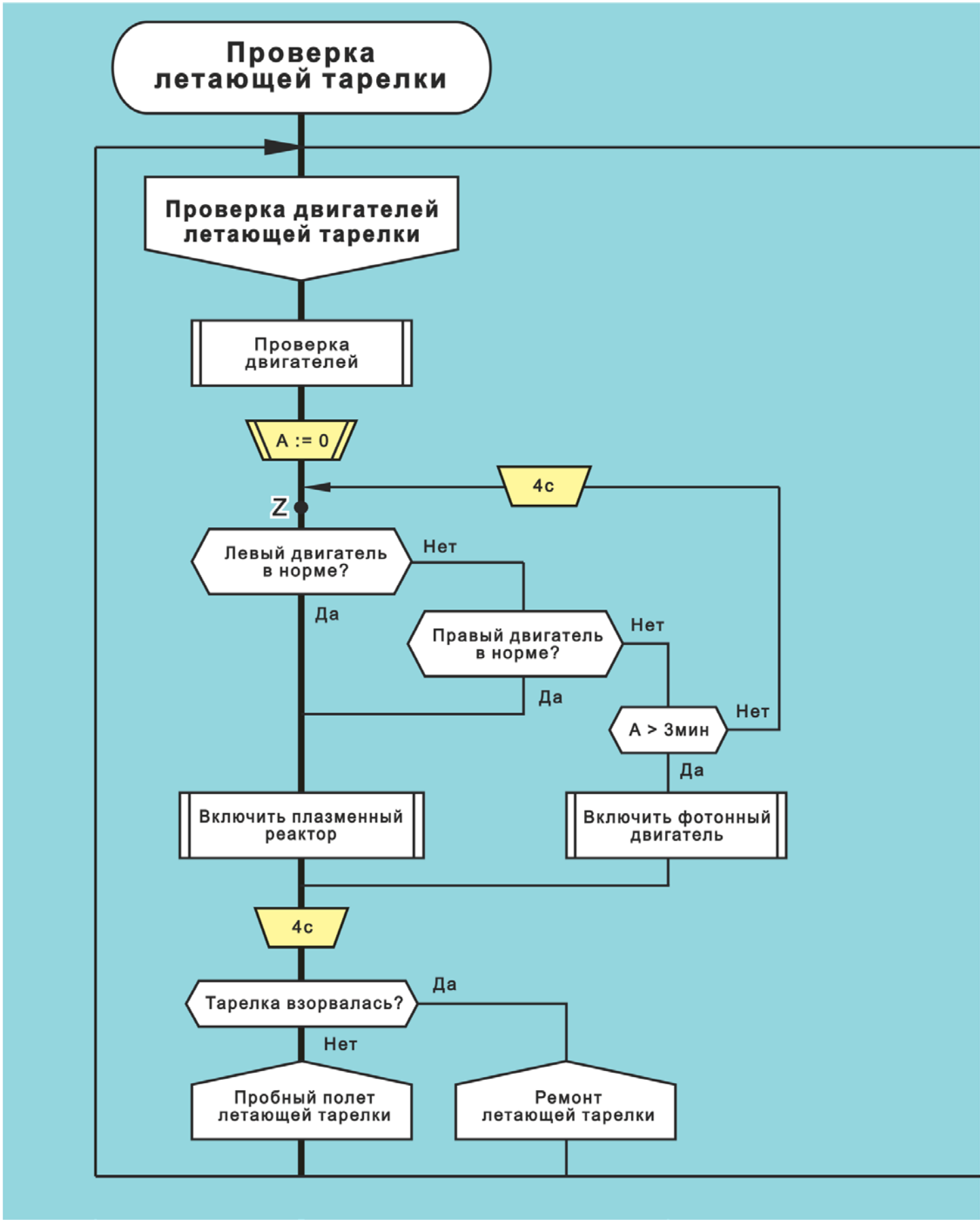
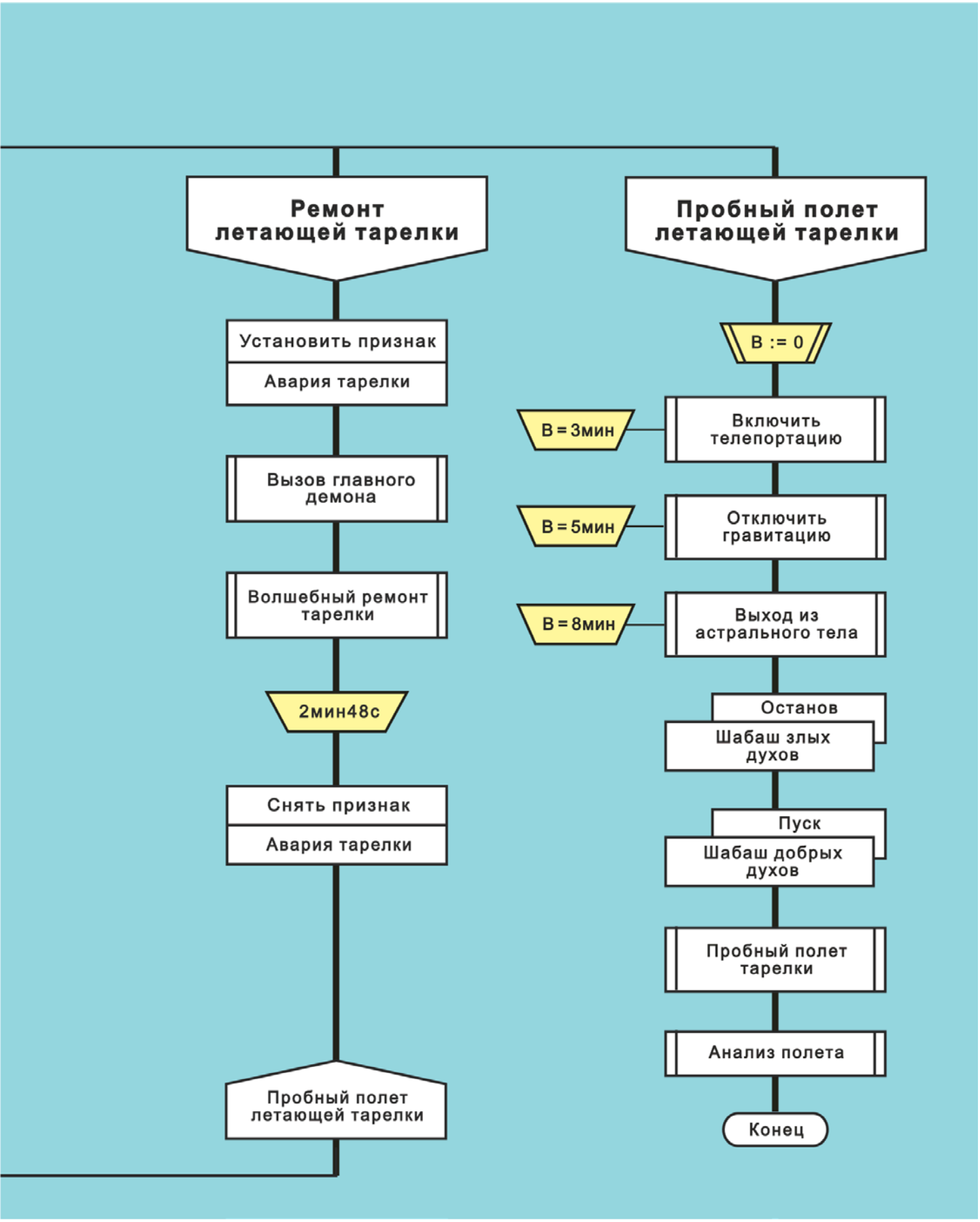


Рис. 123. Алгоритм реального времени «Проверка летающей тарелки»


Ремонт летающей тарелки
Установить признак
Авария тарелки
Вызов главного демона
Волшебный ремонт тарелки
2мин48с
Снять признак
Авария тарелки
Пробный полет летающей тарелки
Пробный полет летающей тарелки
B := 0
B = 3мин
Включить телепортацию
B = 5мин
Отключить гравитацию
B = 8мин
Выход из астрального тела
Останов
Шабаш злых духов
Пуск
Шабаш добрых духов
Пробный полет тарелки
Анализ полета
Конец

АЛГОРИТМ РЕАЛЬНОГО ВРЕМЕНИ «ПРОВЕРКА ЛЕТАЮЩЕЙ ТАРЕЛКИ»

На рис. 123 представлен более сложный алгоритм, в котором применяются, операторы Таймер, Период, Пауза, Синхронизатор, «Параллельный процесс» в различных сочетаниях.

Силуэт состоит из трех веток:

- Проверка двигателей летающей тарелки.
- Ремонт летающей тарелки.
- Пробный полет летающей тарелки.

Самой сложной является первая ветка, содержащая цикл ЖДАТЬ. С него-то мы и начнем

ЦИКЛ ЖДАТЬ В ЛЕТАЮЩЕЙ ТАРЕЛКЕ

Предположим, нужно в течение трех минут ждать появления хотя бы одного из двух признаков ЛЕВЫЙ ДВИГАТЕЛЬ В НОРМЕ и ПРАВЫЙ ДВИГАТЕЛЬ В НОРМЕ. При наступлении этого события (появлении одного из признаков) необходимо включить плазменный реактор. Если же названные признаки отсутствуют, по истечении трех минут следует включить фотонный двигатель.

Для решения задачи на рис. 123 используются два оператора:

- Таймер *A*, отсчитывающий три минуты;
- Цикл ЖДАТЬ.

В состав последнего входит икона Период и три иконы Вопрос. В последних размещены надписи:

- ЛЕВЫЙ ДВИГАТЕЛЬ В НОРМЕ?
- ПРАВЫЙ ДВИГАТЕЛЬ В НОРМЕ?
- *A* > 3мин

Последний оператор проверяет: отсчитал ли таймер *A* три минуты?

Если оба признака отсутствуют, а три минуты еще не прошли, опрос условий периодически повторяется. При этом период опроса указывается в иконе Период. В данном примере он равен 4 секундам.

Когда закончится цикл ЖДАТЬ на рис. 123? В момент обнаружения одного из ожидаемых признаков, а если они не появятся — через 3 минуты.

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ ОПЕРАТОРОВ РЕАЛЬНОГО ВРЕМЕНИ

Всего на рис. 123 имеется 10 операторов реального времени (найдите их).

Таймеры принято обозначать латинскими буквами A, B, C и т. д. В данной схеме используются два таймера: A и B.

На рис. 123 видно, что оператор Таймер можно применять двумя способами:

- совместно с иконой Вопрос,
- совместно с иконой Синхронизатор.

В первой ветке икона Таймер используется в паре с иконой Вопрос. Икона Таймер имеет надпись *A* := 0. Данный оператор порождает, обнуляет и запускает таймер *A*.

Где используется таймер A? В иконе Вопрос, где написано A > 3мин. Последний оператор позволяет спасти положение и включить фотонный двигатель в опасном случае, когда остальные двигатели вышли из строя.

В третьей ветке установлен второй таймер — таймер B (см. икону Таймер с текстом B := 0). Он работает совместно с тремя иконами Синхронизатор. Последние снабжены надписями:

- *B* = 3мин,
- *B* = 5мин,
- *B* = 8мин.

В результате вызов процедуры ВКЛЮЧИТЬ ТЕЛЕПОРТАЦИЮ произойдет не сразу, а только после того, как таймер *B* отсчитает 3 минуты. Соответственно включение в работу процедур ОТКЛЮЧИТЬ ГРАВИТАЦИЮ и ВЫХОД ИЗ АСТРАЛЬНОГО ТЕЛА будет задержано до тех пор, пока не пройдут 5 и 8 минут соответственно.

Икона Пауза предусмотрена в первой ветке, обеспечивая задержку на 4 секунды. Только после этого производится проверка условия ТАРЕЛКА ВЗОРВАЛАСЬ.

Еще одна икона Пауза размещена в средней ветке с надписью 2мин48с. Это означает, что после завершения процедуры ВОЛШЕБНЫЙ РЕМОНТ ТАРЕЛКИ отсчитывается пауза длительностью 2 минуты 48 секунд. И только после этого производится снятие признака АВАРИЯ ТАРЕЛКИ.

На рис. 123 имеется восемь операторов реального времени в форме трапеции; они окрашены в желтый цвет. Кроме того, в третьей ветке расположены два оператора «Параллельный процесс», которые описаны ниже.

ПОСЛЕДОВАТЕЛЬНАЯ И ПАРАЛЛЕЛЬНАЯ РАБОТА АЛГОРИТМОВ

Пусть заданы два алгоритма *S* и *T*, причем *S* — основной алгоритм, а *T* — вспомогательный. Алгоритмы *S* и *T* могут работать последовательно (рис. 124) или параллельно (рис. 125).

Чтобы организовать последовательную работу, необходимо в дракон-схеме основного алгоритма *S* нарисовать икону Вставка с надписью *T*. В этом случае алгоритм *T* называется *процедурой.*

Чтобы организовать параллельную работу, нужно в дракон-схеме основного алгоритма *S* нарисовать икону «Параллельный процесс» (рис. 20, п. 28).

Икона «Параллельный процесс» двухэтажная. На верхнем этаже пишут ключевое слово, обозначающее команду, изменяющую состояние параллельного процесса, например, «Пуск», «Останов» и т. д. На нижнем этаже помещают идентификатор (название) параллельного процесса.

Циклограммы на рис. 124 и 125 наглядно показывают, в чем отличие двух режимов. На рис. 124 — после начала работы вызываемого алгоритма *T* — вызывающий алгоритм *S* прекращает работу. И возобновляет ее лишь тогда, когда алгоритм *T* выполнит задание. Следовательно, алгоритмы *S* и *T* работают по очереди и никогда вместе.

На рис. 125, напротив, вызывающий алгоритм *S* не останавливается и работает одновременно с вызываемым алгоритмом *T.*

Таким образом, отличие параллельного режима от последовательного состоит в том, что после начала вспомогательного алгоритма *T* основной алгоритм *S* не прекращает работу, а действует одновременно с алгоритмом *T*.

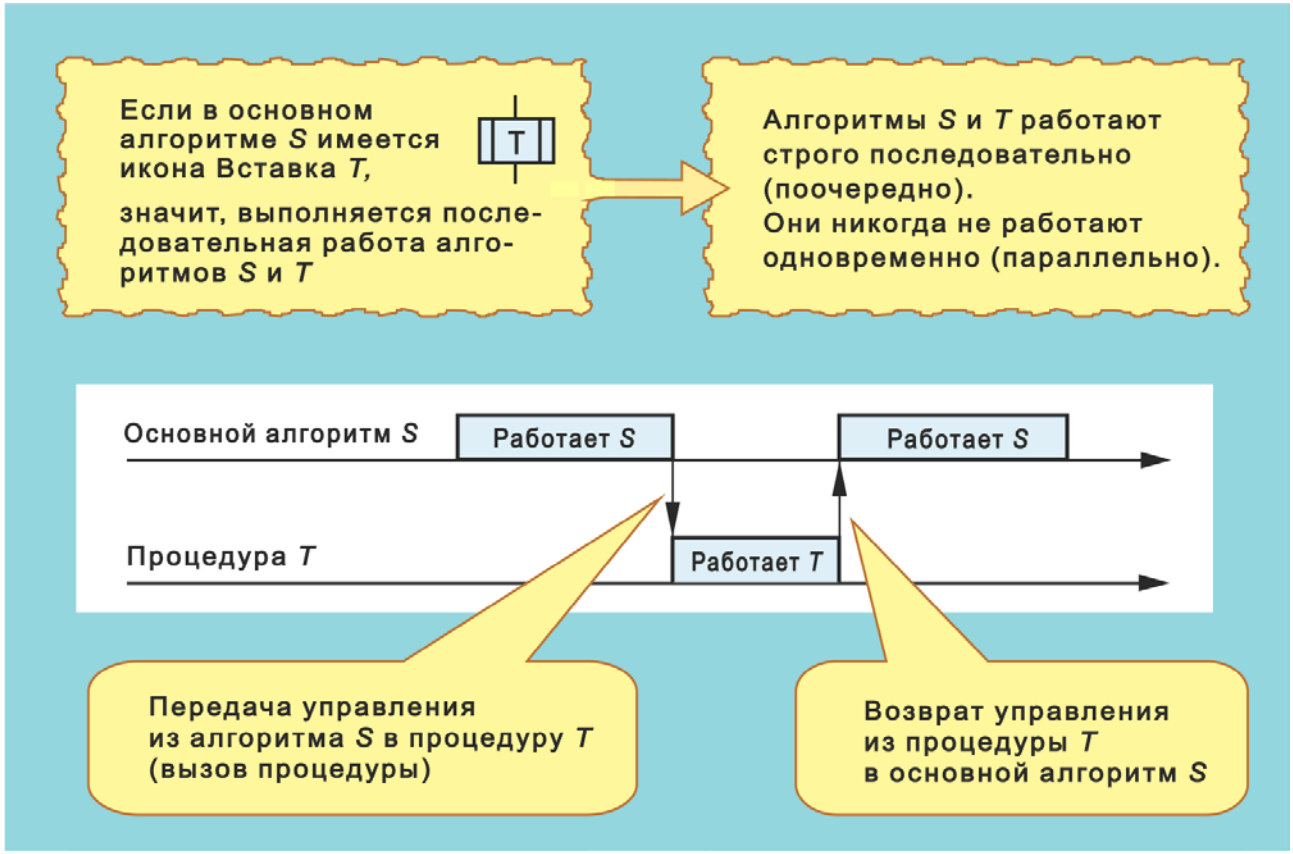


Рис. 124. Последовательная работа алгоритмов

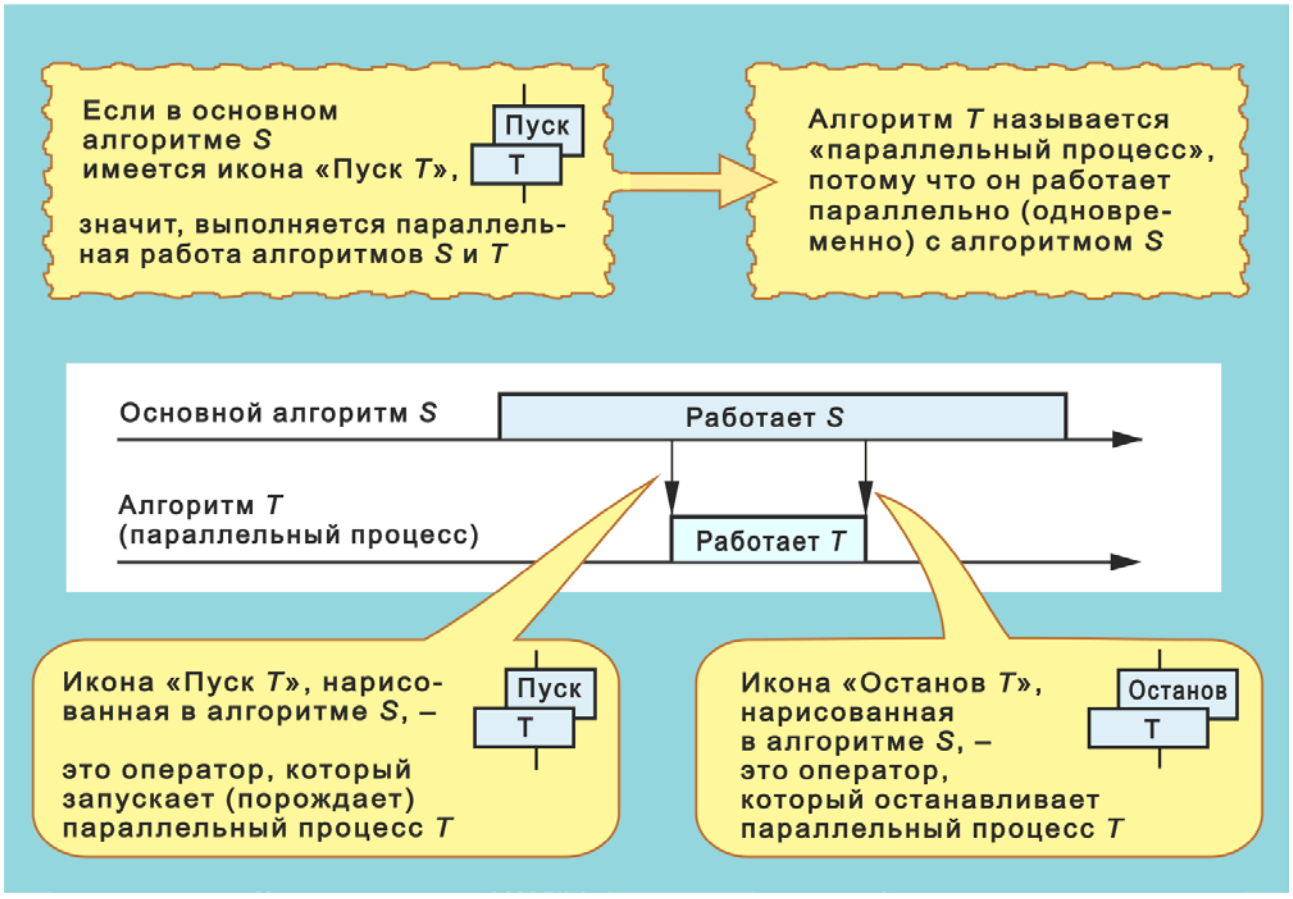


Рис. 125. Параллельная работа алгоритмов

ОПЕРАТОР «ПАРАЛЛЕЛЬНЫЙ ПРОЦЕСС» В ЛЕТАЮЩЕЙ ТАРЕЛКЕ

Вернемся к примеру на рис. 123. В правой ветке находятся два оператора управления параллельными процессами. После окончания процедуры ВЫХОД ИЗ АСТРАЛЬНОГО ТЕЛА производится Останов параллельного процесса ШАБАШ ЗЛЫХ ДУХОВ и Пуск процесса ШАБАШ ДОБРЫХ ДУХОВ.

При этом предполагается, что до начала алгоритма ПРОВЕРКА ЛЕТАЮЩЕЙ ТАРЕЛКИ некий третий алгоритм выдал команду «Пуск» и запустил параллельный процесс ШАБАШ ЗЛЫХ ДУХОВ. Последний работает одновременно с алгоритмом ПРОВЕРКА ЛЕТАЮЩЕЙ ТАРЕЛКИ вплоть до момента выдачи команды «Останов» (см. последнюю ветку на рис. 123).

Указанная команда ликвидирует параллельный процесс ШАБАШ ЗЛЫХ ДУХОВ. В этот момент одновременная работа заканчивается.

Однако следующая команда «Пуск» запускает другой параллельный процесс — ШАБАШ ДОБРЫХ ДУХОВ, который начинает работать одновременно с алгоритмом ПРОВЕРКА ЛЕТАЮЩЕЙ ТАРЕЛКИ.

ОПЕРАТОРЫ И КАРТИНКИ

Операторы реального времени — это формальные операторы языка ДРАКОН. Однако их можно использовать и при неформальном изображении алгоритмов. Например, для построения наглядных картинок, позволяющих легко объяснить ту или иную идею, относящуюся к системам реального времени.

Пример картинки представлен на рис. 113. В цикле ЖДАТЬ икону Период обычно опускают, чтобы не загромождать рисунок (см. последнюю ветку на рис. 113). Однако если длительность периода нужна для понимания, икону Период можно сохранить.

СООБЩЕНИЕ ДЛЯ ПРОГРАММИСТОВ

На рис. 113-123 показаны операторы реального времени, нарисованные внутри алгоритмов. Поэтому может создаться впечатление, что они реализуются этими алгоритмами (то есть прикладными программами реального времени). Но это не всегда так.

Рассмотрим случай, когда используется Операционная система реального времени (ОСРВ). В таком случае операторы реального времени реализуются совместно:

- прикладной программой реального времени;
- дракон-диспетчером, входящим в состав операционной системы реального времени.

Когда в прикладной программе встречается оператор Пауза, происходят события, не показанные на наших рисунках. А именно, выход из прикладной программы и передача управления в дракон-диспетчер (с одновременной передачей параметра, записанного в иконе Пауза).

Получив параметр, диспетчер отсчитывает время, указанное в Паузе. Когда время истечет, диспетчер возвращает управление в прикладную программу — в точку, расположенную после иконы Пауза.

Иными словами, всякий раз, когда на рисунке алгоритма изображена Пауза происходят два события:

- выход из прикладной программы (в начале Паузы);

- вход в прикладную программу (в конце Паузы).

Рассмотрим еще один пример — оператор Период. Длительность периода отсчитывает не прикладная программа на рис. 123, а дракон-диспетчер, входящий в состав операционной системы реального времени.

Оператор Период означает выход из прикладной программы. Управление переходит к дракон-диспетчеру (с одновременной передачей параметра 4с). Через каждые 4 секунды дракон-диспетчер передает управление в начало цикла ЖДАТЬ (точка Z на рис. 123). Если все три условия дают ответ «нет», оператор Период всякий раз возвращает управление в дракон-диспетчер. Таким образом, функционирование цикла ЖДАТЬ обеспечивается совместными усилиями прикладной программы и дракон-диспетчера.

Этот вывод относится и к другим операторам реального времени.

На рисунках 113-123 показаны алгоритмы, которые имеют одно начало (один вход) и один конец (один выход). В действительности программы реального времени имеют много входов и много выходов. Дополнительные входы и выходы появляются всякий раз, когда в алгоритм добавляется оператор Пауза, Период и др. Но эти дополнительные входы и выходы на рисунках не показаны. Они не показаны из эргономических соображений — чтобы не загромождать рисунок.

ВЫВОДЫ

1. Наличие операторов реального времени резко расширяет изобразительные возможности языка ДРАКОН и позволяет использовать его при проектировании и разработке не только информационных, но и управляющих систем. Это обстоятельство существенно увеличивает область применения языка.
2. Дополнительным преимуществом является лаконичность выразительных средств, их универсальность. В языке всего пять икон реального времени, однако их алгоритмическая мощь — в сочетании с другими возможностями языка — позволяет охватить обширный спектр задач, связанных с созданием алгоритмов и программ для управляющих систем.
3. Важную роль играет эргономичность операторов реального времени. Как и другие операторы языка ДРАКОН, они имеют визуальный характер, что позволяет сделать операции реального времени более наглядными и легкими для понимания по сравнению с традиционной текстовой записью.
4. Четыре иконы (Пауза, Период, Таймер и Синхронизатор) — «близкие родственники» в том смысле, что внутри каждой из них указывается значение времени. Эта родственная связь находит свое эргономическое отражение в том, что перечисленные операторы имеют визуальное «семейное сходство». Все они построены (с вариациями) на основе одной и той же геометрической фигуры — перевернутой равнобедренной трапеции.

Глава 20

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ

ВВЕДЕНИЕ

Параллельные процессы играют важную роль в технике и многих других областях.

Определение параллельного процесса дано на рис. 125. Икона «Параллельный процесс» показана на рис. 20, п. 28.

Краткое описание процесса дано в предыдущей главе в параграфе «Последовательная и параллельная работа алгоритмов».

В данной главе следующие выражения рассматриваются как синонимы:

- параллельный процесс;
- процесс;
- параллельный алгоритм.

ПАРАЛЛЕЛЬНЫЕ ПРОЦЕССЫ В АЛГОРИТМЕ «ПРОВЕРКА АГРЕГАТА И РАКЕТЫ»

На рис. 126 изображены 15 параллельных процессов:

- вызывающий алгоритм ПРОВЕРКА АГРЕГАТА И РАКЕТЫ;
- 14 вызываемых алгоритмов, каждый из которых обозначен иконой «Параллельный процесс» (7 алгоритмов в первой ветке и 7 — во второй).

Все вызываемые процессы запускаются сигналом ПУСК. Момент запуска точно определен оператором Синхронизатор. Например, процесс КОНТРОЛЬ ПРИБОРОВ запускается в момент 103с (103 секунды).

Параллельные процессы могут заканчиваться двумя способами:

- по команде «Останов» (см. пример на рис. 123, правая ветка);
- без использования команды «Останов», то есть естественным путем, когда каждый процесс решит свою задачу и достигнет конца.

На рис. 126 показан случай, когда все вызываемые процессы заканчиваются естественным путем. Поэтому команда ОСТАНОВ не используется.

ВРЕМЕННАЯ ДИАГРАММА ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ

На рис. 127 показана временная диаграмма, иллюстрирующая алгоритм на рис. 126. В верхней строке темным цветом выделен вызывающий алгоритм. Он имеет самую большую длительность. Ниже расположены вызываемые процессы.

В самом верху указано время запуска всех процессов по таймеру. Процессы имеют разную длительность, потому что каждый процесс выполняет задачу за разное время.

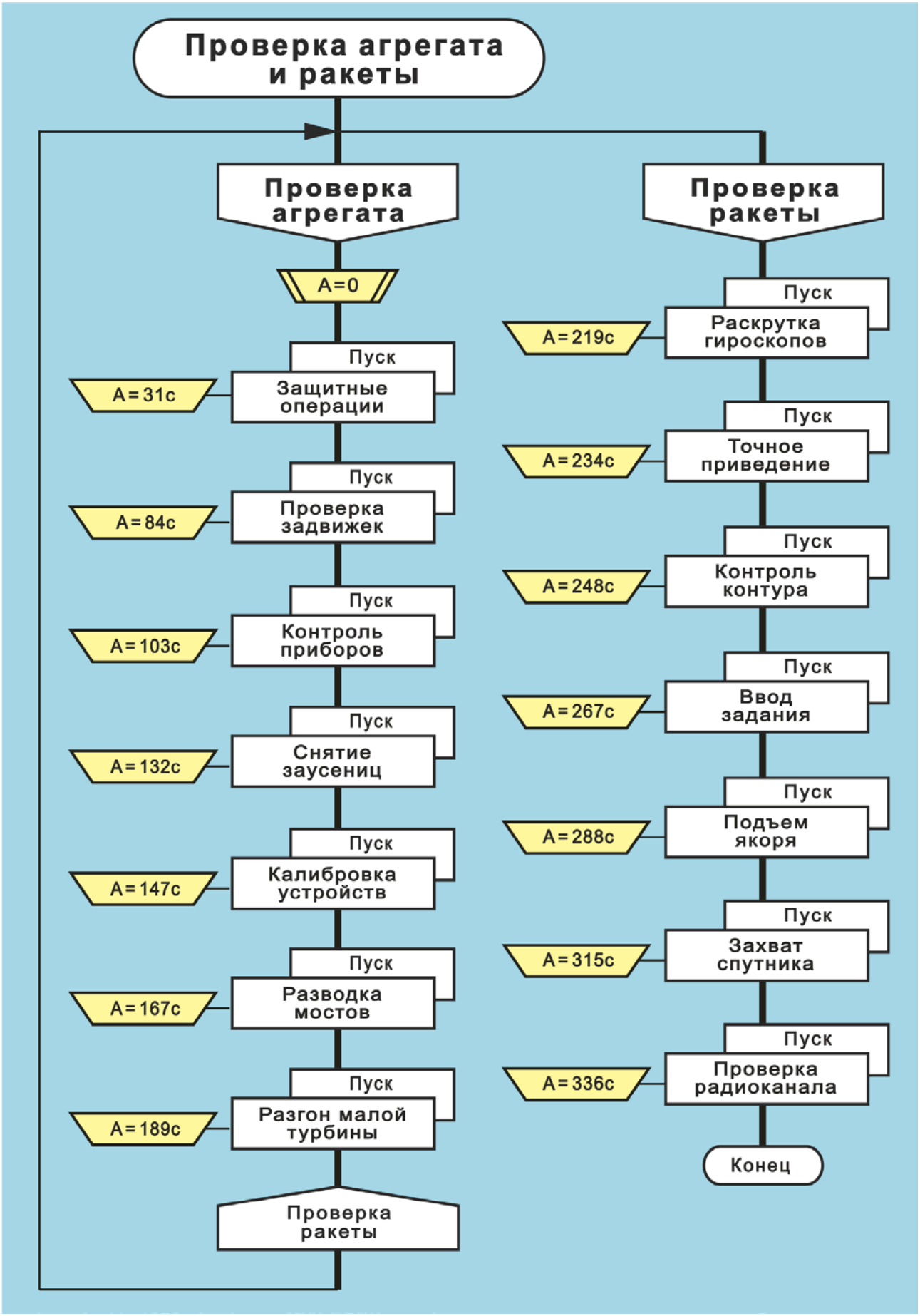


Рис. 126. Алгоритм с параллельными процессами «Проверка агрегата и ракеты»

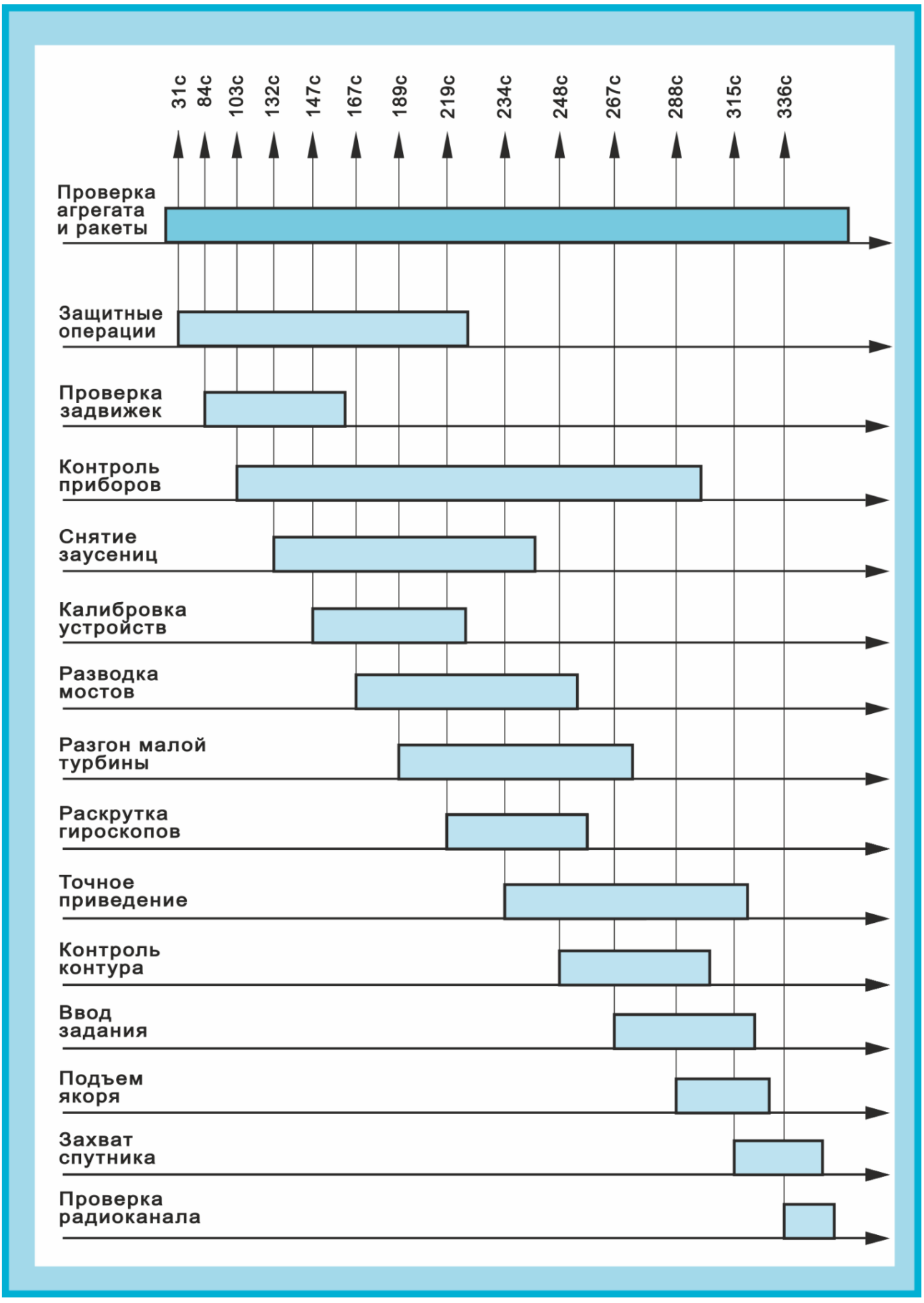


Рис. 127. Циклограмма параллельных процессов, запускаемых из алгоритма на рис. 126

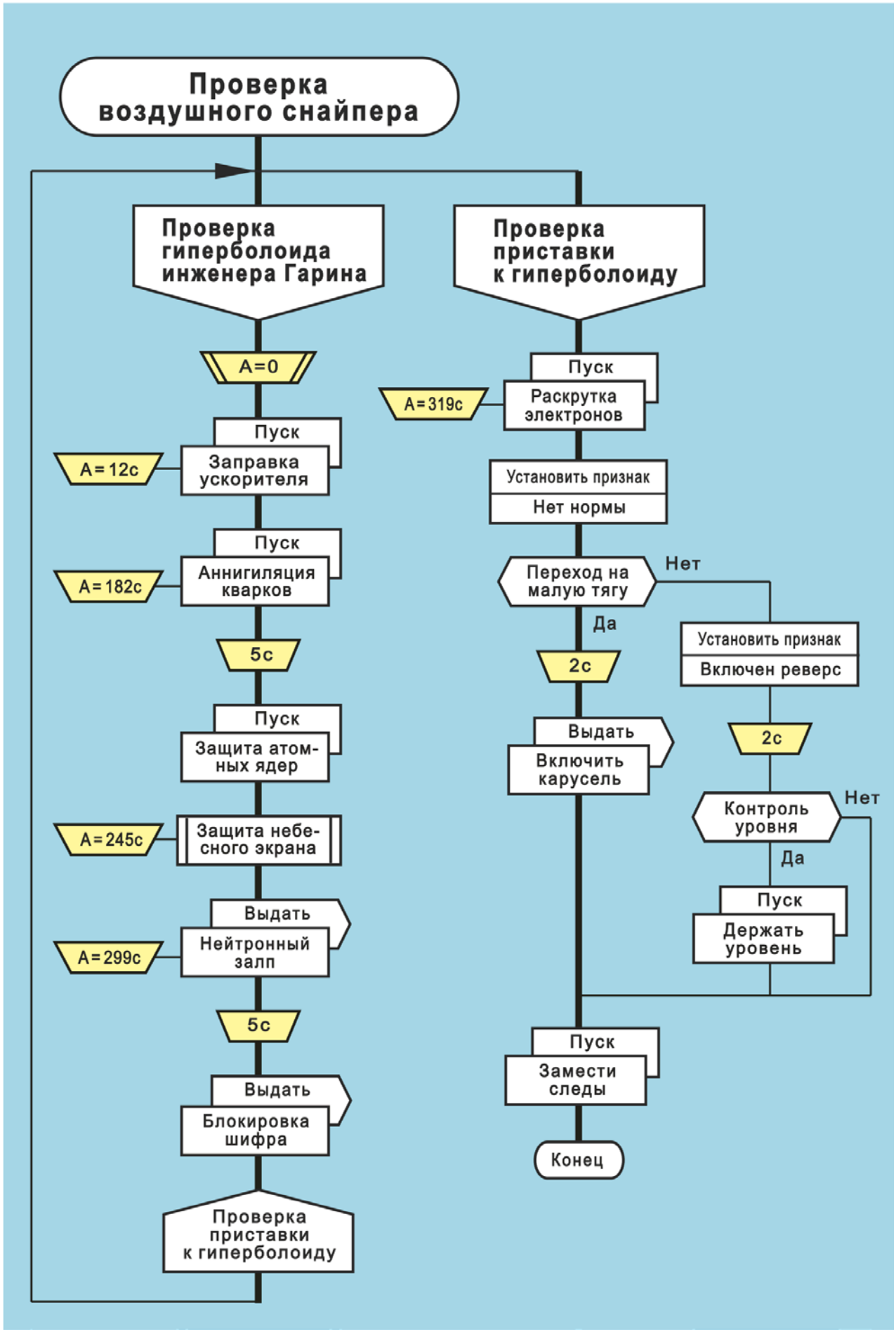


Рис. 128. Алгоритм с параллельными процессами «Проверка воздушного снайпера»

ПАРАЛЛЕЛЬНЫЕ ПРОЦЕССЫ В АЛГОРИТМЕ «ПРОВЕРКА ВОЗДУШНОГО СНАЙПЕРА»

На рис. 126 показан упрощенный случай. Одна и та же операция «Пуск параллельного процесса» повторяется 14 раз. При этом применяются 14 Синхронизаторов, которые задают 14 моментов времени, определяющих запуск 14 параллельных процессов.

На рис. 128 показан более сложный случай. Наряду с Таймером, Синхронизатором и Процессами применяются следующие иконы: Вывод, Вставка, Вопрос и Полка.

В первой ветке имеются 4 Синхронизатора. Два из них запускают процессы ЗАПРАВКА УСКОРИТЕЛЯ и АННИГИЛЯЦИЯ КВАРКОВ. Третий включает процедуру ЗАЩИТА НЕБЕСНОГО ЭКРАНА. Четвертый выдает команду НЕЙТРОННЫЙ ЗАЛП.

Используются не только Синхронизаторы, но и две Паузы длительностью 5 секунд каждая. Первая пауза задерживает пуск процесса ЗАЩИТА АТОМНЫХ ЯДЕР. Вторая задерживает выдачу команды БЛОКИРОВКА ШИФРА.

Во второй ветке выполняются операции:

- Синхронизатор *«A = 319c»* задерживает процесс РАСКРУТКА ЭЛЕКТРОНОВ до тех пор, пока Таймер *A* не отсчитает 319 секунд;
- устанавливаются два признака НЕТ НОРМЫ и ВКЛЮЧЕН РЕВЕРС;
- применяются две иконы вопрос: ПЕРЕХОД НА МАЛУЮ ТЯГУ? и КОНТРОЛЬ УРОВНЯ?
- выдается команда ВКЛЮЧИТЬ КАРУСЕЛЬ;
- запускается процесс ДЕРЖАТЬ УРОВЕНЬ;
- и т.д.

В алгоритме на рис. 128 используются пять вызываемых параллельных процессов.

В первой ветке (через Синхронизаторы) запускаются два процесса.

Во второй ветке применяются три процесса. Один запускается через Синхронизатор. Второй — через иконы Пауза и Вопрос. Третий — через более сложную схему.

КОМАНДЫ УПРАВЛЕНИЯ ПАРАЛЛЕЛЬНЫМИ ПРОЦЕССАМИ

Команды управления пишут на верхнем этаже иконы «Параллельный процесс».

Ниже представлен перечень команд управления:

Пуск	Осуществляет пуск параллельного процесса
Останов	Осуществляет останов параллельного процесса
Стоп	Осуществляет приостановку параллельного процесса
Рестарт	Осуществляет повторный пуск приостановленного параллельного процесса

ДРУГОЙ СПОСОБ ИЗОБРАЖЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ

На рис. 129 показан алгоритм «Миша и Коля носят кирпичи», где вводятся новые обозначения и правила.

Вместо иконы «Параллельный процесс» использованы две иконы:

- Начало параллельных действий;
- Конец параллельных действий.

Двойная горизонтальная линия обозначает начало нескольких (в данном примере, двух) параллельных действий.

Расположенная ниже одиночная горизонтальная линия обозначает конец параллельных действий.

О чем речь на рис. 129? Самосвал выгрузил кучу кирпичей. Их нужно перенести к месту стройки на носилках. За дело взялись два грузчика: Миша и Коля. На чертеже видно, что каждый из них выполняет три параллельных действия:

- подними носилки…,
- перенеси кирпичи…,
- опусти носилки…

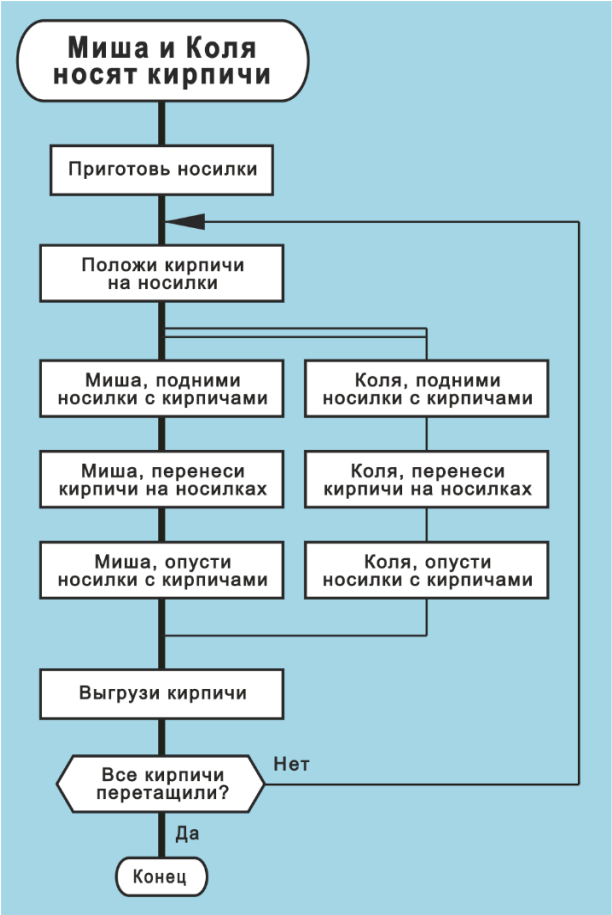


Рис. 129. Алгоритм «Миша и Коля носят кирпичи»

СИНХРОННО ИЛИ НЕ СИНХРОННО

На рис. 129 показан случай, когда параллельные действия выполняются синхронно. Действительно, Миша и Коля должны поднять (и опустить) носилки одновременно. Если Миша спереди подымет носилки, а Коля сзади зазевается, возможен несчастный случай.

Однако, требование синхронности предъявляется не всегда. Предположим, Миша должен купить к обеду буханку хлеба, а Коля — бутылку кефира. В каком порядке они будут это делать — одновременно или поочередно — неважно. Таким образом, покупка хлеба и покупка кефира — это два параллельных действия, где синхронность не играет роли. Главное, чтобы к обеду на столе были и хлеб, и кефир.

КАК ПОКАЗАТЬ ПАРАЛЛЕЛЬНУЮ РАБОТУ ДВУХ ВЕТОК СИЛУЭТА

Рассмотрим силуэт Z на рис. 130. Предположим, что ветки B и C должны работать параллельно. Как это изобразить?

В нижней части ветки A размещена икона «Начало параллельных действий». Двойная горизонтальная линия связана с иконами: «Адрес B» и «Адрес C». Это значит, что ветки B и C *начнут* работать параллельно.

А как показать, что ветки B и C *кончат* работать параллельно? В нижней части веток B и C надо поместить одну и ту же икону Адрес — в данном случае икону «Адрес D» (рис. 130). Наличие одинакового Адреса в конце двух веток означает прекращение параллельной работы.

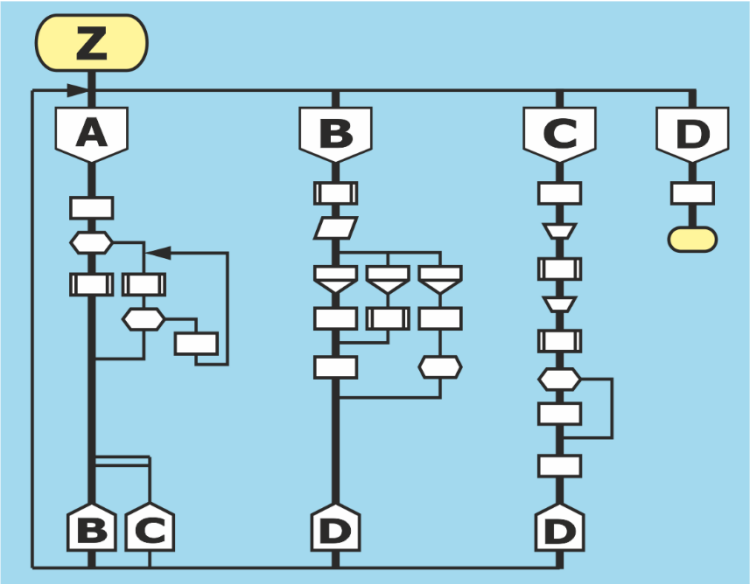


Рис. 130. Алгоритм Z. Показана параллельная работа веток B и C

РАЗДЕЛЕНИЕ И СЛИЯНИЕ ПАРАЛЛЕЛЬНЫХ ДЕЙСТВИЙ

Икона «Начало параллельных действий» представляет собой пункт разделения процесса (concurrent fork). На рис. 130 пункт разделения находится в ветке A внизу.

Икона «Конец параллельных действий» представляет собой пункт слияния (concurrent join). На рис. 130 отсутствует икона «Конец параллельных действий». Вместо нее роль пункта слияния выполняют две иконы «Адрес D» в ветках B и C.

Пункт разделения (concurrent fork)	● Обозначает разделение одного процесса на несколько параллельных действий (процессов) ● Обозначает разделение одного маршрута на несколько параллельных маршрутов
Пункт слияния (concurrent join)	● Обозначает слияние параллельных действий в один процесс. ● Обозначает слияние нескольких параллельных маршрутов в один маршрут.

РИТМИЧЕСКИЕ ПРОМЕЖУТКИ МЕЖДУ ВЕТКАМИ

Расстояние по горизонтали между соседними ветками называется *ритм.* Чтобы силуэт был удобным для зрительного восприятия, желательно между ветками предусмотреть пустой промежуток, который раздвигает ветки в стороны. Этот промежуток и есть *ритм* (рис. 130). Каждая ритмическая полоса зрительно отделяет ветки друг от друга и облегчает чтение.

Расстояние по горизонтали между соседними иконами называется *метр.* Правило ритма и метра гласит: чтобы чертеж был красивым, ритм должен быть намного больше метра.

Метр можно сравнить с пробелами, которые отделяют слова друг от друга в любой книге. А ритм — со свободным от текста «белым пространством», предшествующим началу новой главы в книге.

ВЫВОДЫ

1. Следует различать два понятия: параллельный процесс и параллельные действия.
2. Параллельный процесс запускается командой Пуск.
3. Параллельные процессы могут заканчиваться двумя способами:
    - командой Останов;
    - без использования команды Останов, когда каждый процесс выполнит свою задачу и достигнет конца.
4. Кроме Пуска и Останова, для управления параллельным процессом служат команды Стоп и Рестарт.
5. Для реализации параллельных действий служат иконы:
    - Начало параллельных действий *(concurrent fork).*
    - Конец параллельных действий *(concurrent join).*
6. Начало параллельных действий обозначается двойной горизонтальной линией.
7. Конец параллельных действий обозначается одиночной горизонтальной линией.
8. Конец параллельной работы двух или более веток обозначается иконой Адрес, в которой записан одинаковый адрес.
9. Правило ритма и метра облегчает чтение алгоритма.

Часть 5

АЛГОРИТМЫ ПРАКТИЧЕСКОЙ ЖИЗНИ

Глава 21

АЛГОРИТМЫ В МЕДИЦИНЕ

ЯЗЫК ДРАКОН ПОЗВОЛЯЕТ ПРЕДСТАВИТЬ МЕДИЦИНСКИЕ АЛГОРИТМЫ В УДОБНОЙ ФОРМЕ

В предыдущем разделе (главы 1—20) мы познакомились с языком ДРАКОН.

В этом разделе (главы 21—26) перейдем к практике и рассмотрим примеры. Мы покажем, что ДРАКОН позволяет наглядно изображать алгоритмы в любых областях жизни и во всех отраслях деятельности.

Что это даст? Поскольку процесс создания алгоритмов оказывается чрезвычайно легким, он становится доступным для любого человека.

С появлением ДРАКОНа специалисты приобретают новые возможности:

- они могут выражать свои знания на своем родном профессиональном языке, но в формализованном графическом виде;
- они могут рисовать алгоритмы с большой скоростью, которая раньше считалась невозможной;
- мысли одного специалиста благодаря ДРАКОНу становятся понятными профессионалам других областей.

В итоге люди получают мощное средство профессионального общения.

Начнем с медицины. И покажем, что ДРАКОН легко отображает любые действия и решения медицинского персонала.

ИЗМЕРЕНИЕ КРОВЯНОГО ДАВЛЕНИЯ

На рис. 131 представлен знакомый почти каждому пример — измерение кровяного давления.

Пояснение

Артериальное давление — это давление внутри кровеносных сосудов (артерий), обеспечивающих продвижение крови по кровеносной системе. Оно измеряется в миллиметрах ртутного столба.

Артериальное давление считается нормальным, если верхнее (систолическое) давление равно 120 мм. ртутного столба. А нижнее (диастолическое) давление равно 80 мм. ртутного столба.

Если давление систематически повышается, это плохо для здоровья.

О повышенном артериальном давлении говорят, когда верхнее давление превышает 140 мм. рт. столба. Или когда нижнее давление превышает 90 мм. рт. столба.

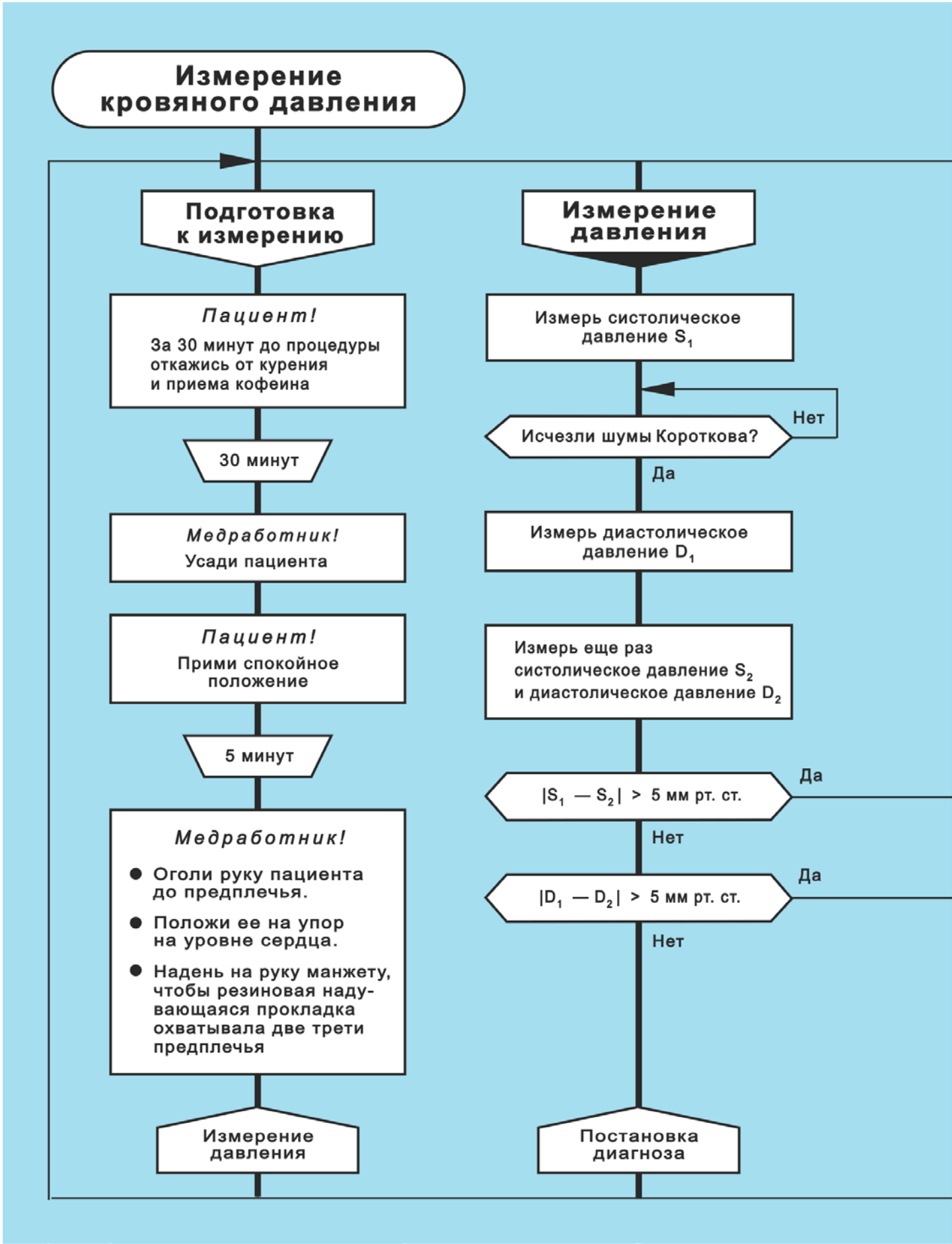
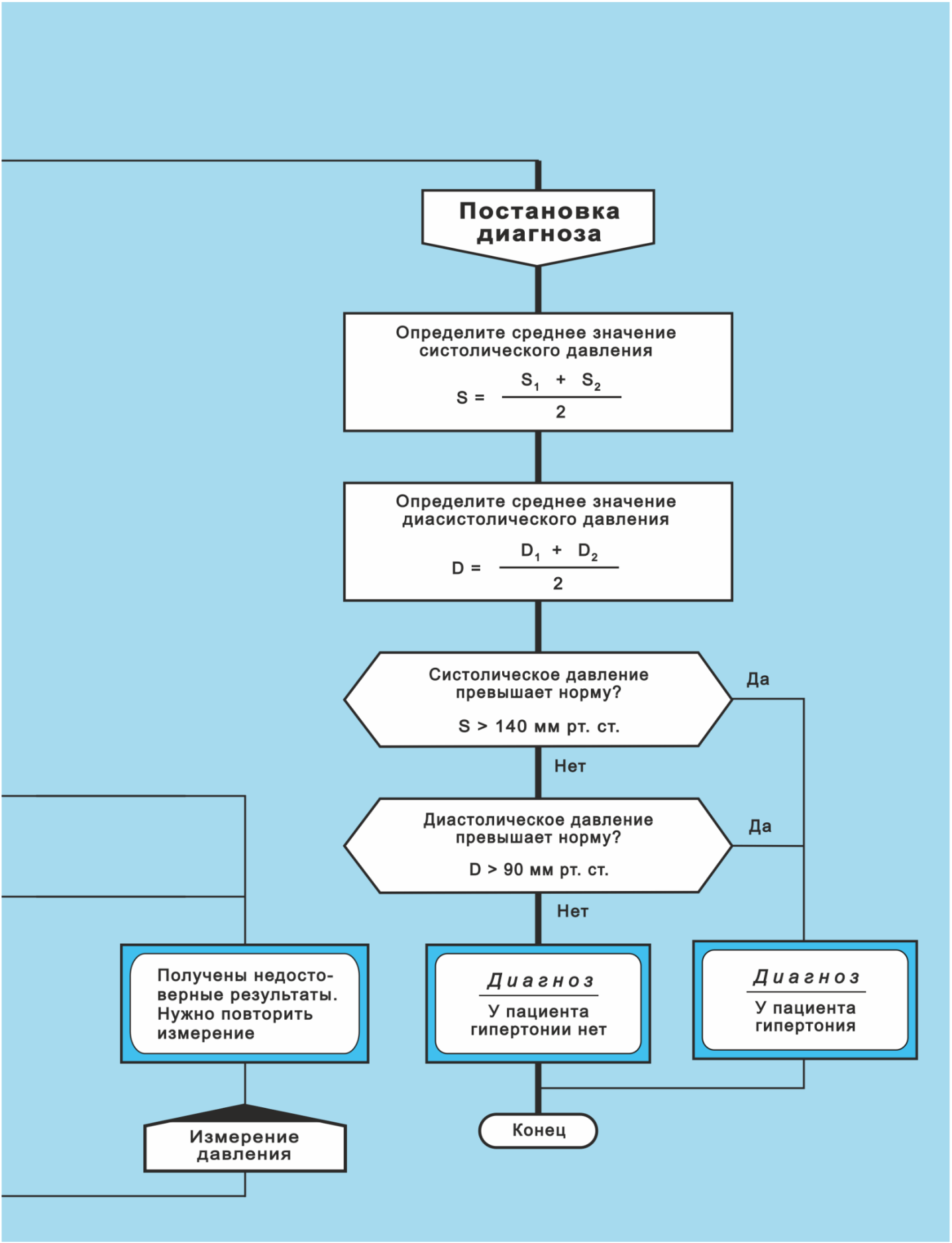


Рис. 131. Алгоритм «Измерение кровяного давления»


Постановка диагноза
Определите среднее значение систолического давления
$S = \frac{S_1 + S_2}{2}$
Определите среднее значение диастолического давления
$D = \frac{D_1 + D_2}{2}$
Систолическое давление превышает норму?
S > 140 мм рт. ст.
Да
Нет
Диастолическое давление превышает норму?
D > 90 мм рт. ст.
Да
Нет
Получены недостоверные результаты. Нужно повторить измерение
Диагноз
У пациента гипертонии нет
Диагноз
У пациента гипертония
Измерение давления
Конец

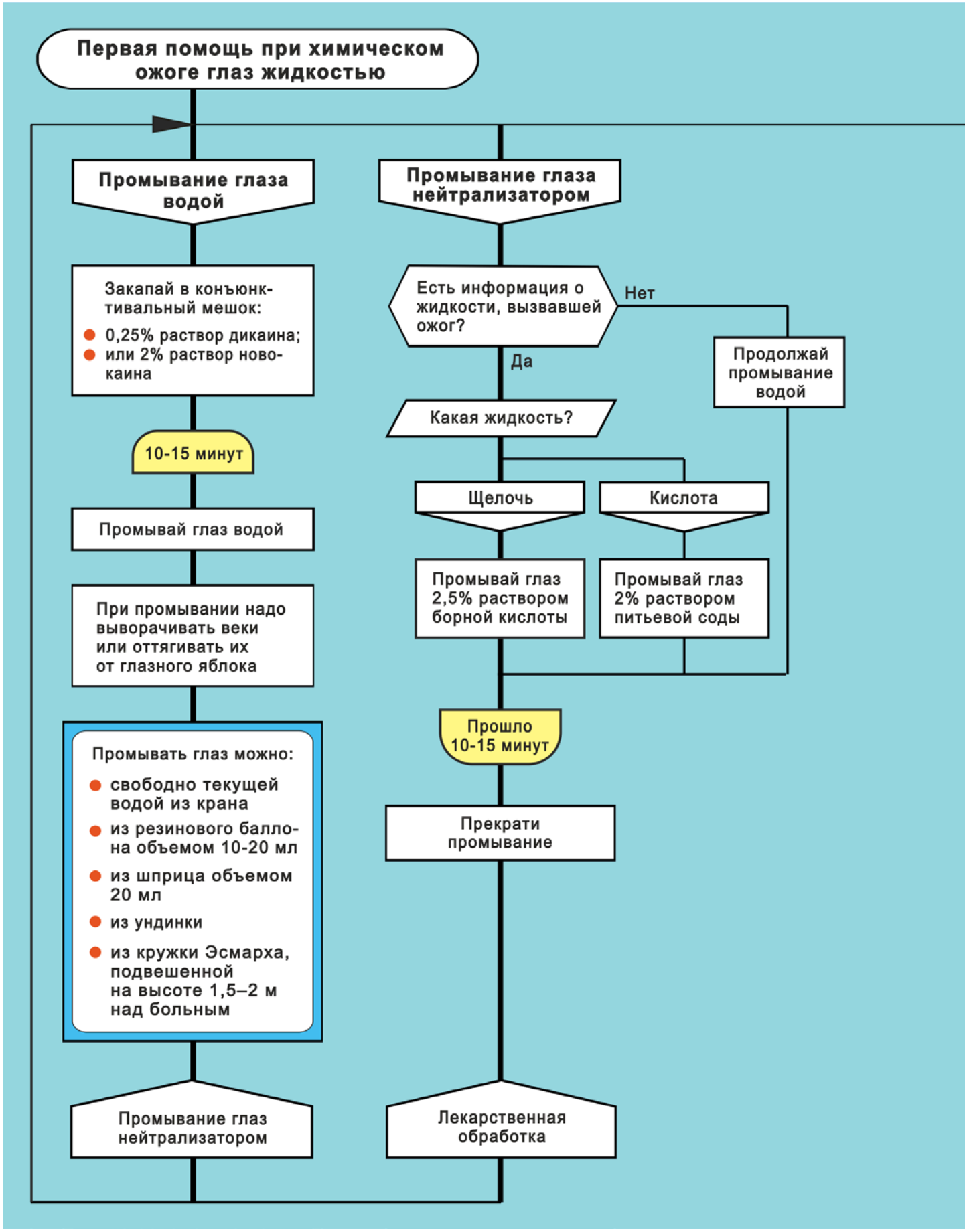


Рис. 132. Алгоритм «Первая помощь при химическом ожоге глаз жидкостью»

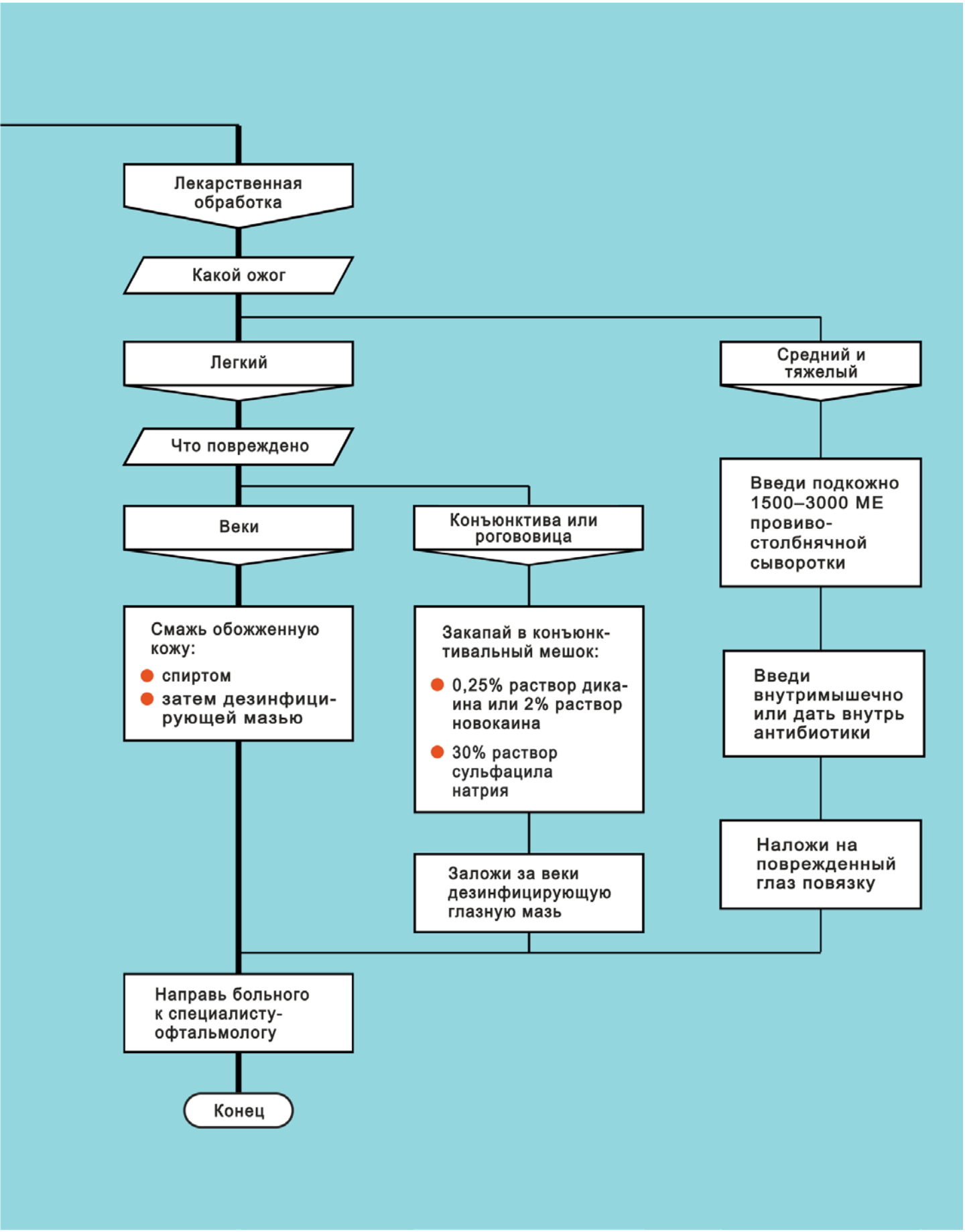


Рис. 132. Алгоритм «Первая помощь при химическом ожоге глаз жидкостью»

Алгоритм на рис. 131 получен путем точного воспроизведения инструкции для врачей, подготовленной комитетом США по профилактической медицине [12].

Обычной практикой является однократное измерение артериального давления. Но американские медики рекомендуют измерять давление не один, а ДВА раза.

Если разница между двумя измерениями меньше 5 мм рт. столба, результат считается достоверным. Если же разница превышает указанную величину, заокеанские врачи рекомендуют аннулировать результат, как недостоверный. И повторить измерение заново.

Алгоритм на рис. 131 в точности отражает американские рекомендации.

ПЕРВАЯ ПОМОЩЬ ПРИ ХИМИЧЕСКОМ ОЖОГЕ ГЛАЗА

Ожог — это повреждение тканей, вызванное тепловым, химическим, электрическим или радиационным воздействием.

Химические ожоги органов зрения вызваны прямым действием химических веществ: кислот, щелочей и других химических агентов (клеи, красители и пр.). В общем случае ожоги глаз могут быть вызваны как твердыми веществами, так и жидкостями.

На рис. 132 изображена первая помощь при химическом ожоге глаз. Дракон-схема составлена на основании «Практического руководства для врачей общей (семейной) практики» [13].

Деление алгоритма на три ветки показывает, что первая помощь при химическом ожоге состоит из трех этапов:

- промывание глаз водой;
- промывание глаз нейтрализатором;
- лекарственная обработка.

Содержание каждой ветки позволяет дать целостную картину проблемы. И, сверх того, указать точную последовательность действий, выполняемых при оказании первой помощи пострадавшему глазу.

Сколько времени нужно промывать глаз водой? 10-15 минут. Как указать начало и конец этого временного промежутка? Для этого служат иконы:

- Начало контрольного срока (рис. 20, п. 24);
- Конец контрольного срока (рис. 20, п. 25).

Они используются в первой и второй ветке на рис. 132. В первой ветке в желтой иконе написано «10-15 минут» — это и есть Начало контрольного срока. Во второй ветке в желтой иконе есть надпись «Прошло 10-15 минут» — это Конец срока.

АЛГОРИТМ «СНЯТИЕ ШЛЕМА С МОТОЦИКЛИСТА»

На рис. 133 показана деятельность двух работников скорой помощи. Алгоритм определяет порядок выполнения неотложных действий по спасению пострадавшего мотоциклиста. Он находится без сознания после дорожной аварии с подозрением на перелом позвоночника.

Помощь мотоциклисту оказывают два спасателя. Они работают одновременно и параллельно. На рисунке им присвоены номера **1** и **2**. Действия первого спасателя описаны в левом столбце, второго – в правом.

Важно отметить, что действия спасателей скоординированы во времени. Координация показана с помощью двух икон:

- Начало синхронной работы спасателей обозначается двойной линией.

- Конец синхронного участка изображают одиночной (горизонтальной) линией.

Алгоритм разбит на две ветки:

- Обеспечение стабильности головы.
- Снятие шлема.

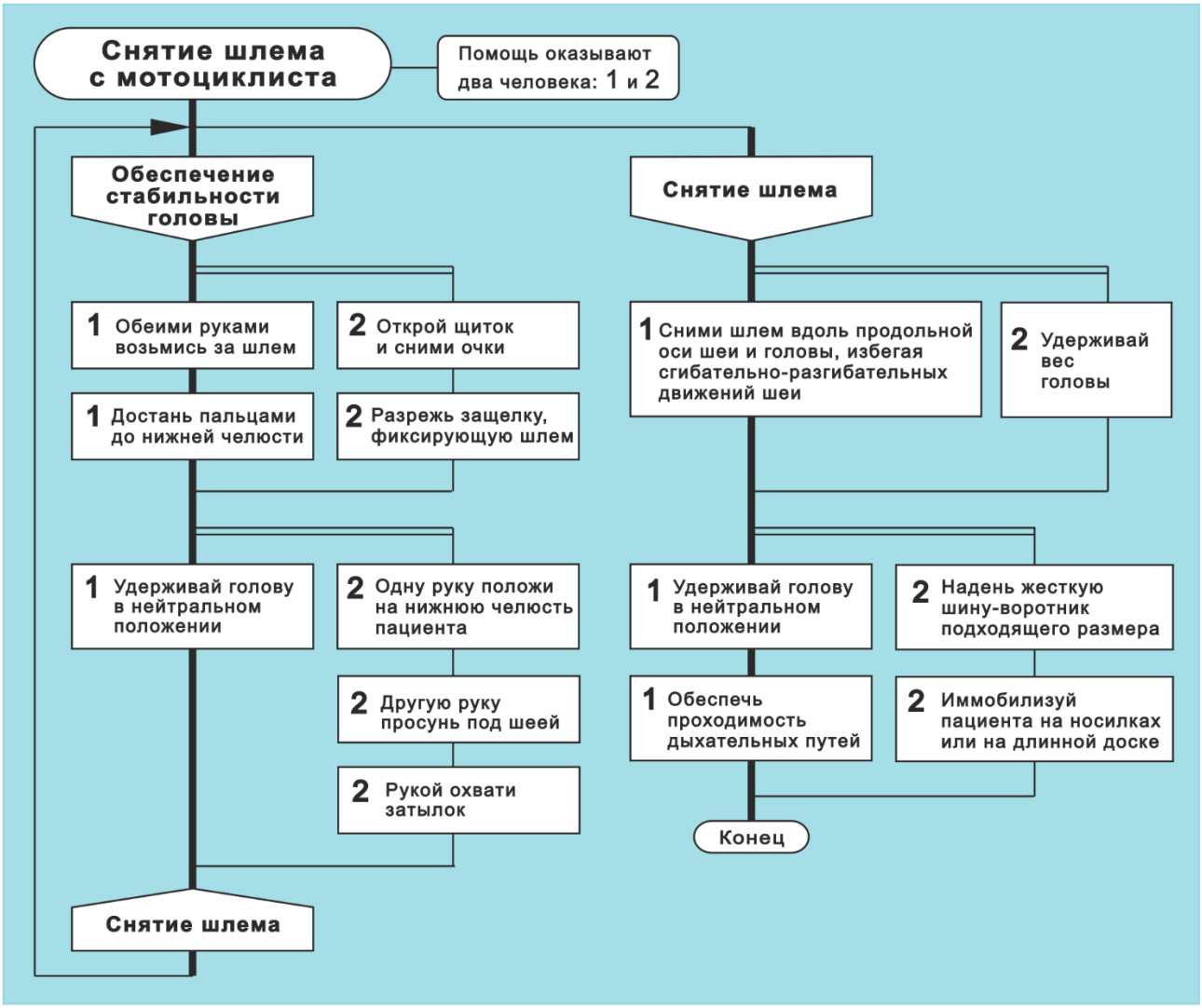


Рис. 133. Алгоритм «Снятие шлема с мотоциклиста»

ПРАВИЛА НУМЕРАЦИИ СПЕЦИАЛИСТОВ

При разработке параллельного медицинского алгоритма необходимо заранее установить четкую нумерацию членов врачебной бригады. Например, для бригады из двух специалистов, необходимо:

- Справа от иконы Заголовок присоединить икону Пояснение с надписью: «Помощь оказывают два человека: **1** и **2**».
- В левой части всех без исключения икон алгоритма поставить номер ответственного за данную операцию члена бригады: **1** или **2**.
- Цифры должны быть крупными и жирными (в пределах разумного), чтобы они сразу бросались в глаза и не вызывали затруднений у читателей.

- Если какую-то операцию медики выполняют вдвоем, в соответствующей иконе слева пишут два номера через запятую: **1, 2**.

Подробности показаны на рис. 133.

ДВУХПОТОЧНЫЙ УЧАСТОК

Двухпоточный участок – часть параллельного алгоритма, обладающая следующими свойствами:

- Начало участка обозначено двойной линией, а конец – одиночной.
- В начале участка единый маршрут (поток) раздваивается и превращается в два: левый и правый. В конце участка, наоборот, два потока объединяются и превращаются в один.
- Двухпоточный участок решает строго определенную функциональную задачу, требующую синхронных действий членов медицинской бригады.

Сверхзадача алгоритма на рис. 133 соответствует принципу «Не навреди». Любое небрежное или неосторожное действие спасателей может усугубить состояние мотоциклиста и нанести ему непоправимый вред.

Исходя из сверхзадачи, алгоритм разбит на четыре двухпоточных участка, каждый из которых выполняет свою частную задачу.

- 1-й участок. Выполнить подготовительные операции: открыть щиток, снять очки, разрезать защелку.
- 2-й участок. Установить руки второго спасателя в нужное положение, чтобы подготовиться 3-му участку (где надо удерживать вес головы).
- 3-й участок. Снять шлем мотоциклиста.
- 4-й участок. Надеть жесткую шину-воротник и иммобилизовать пострадавшего на носилках или доске.

ВЫВОДЫ

1. Важным недостатком современной медицины является отсутствие эффективных способов описания медицинских алгоритмов.
2. В медицинской литературе доминирует текст и некачественные рисунки, доставшиеся врачам в наследство от предыдущих эпох, что существенно тормозит фиксацию, понимание и передачу медицинских знаний.
3. Чтобы осуществить новый мощный прорыв в развитии медицины, необходимы не только новые медицинские знания, но и новые способы описания знаний.
4. Визуализация медицинских алгоритмов представляет собой самостоятельное направление исследований в области медицины.
5. Язык ДРАКОН — новое средство, пригодное для описания медицинских алгоритмов. Оно позволяет выявить действия и решения врачей, сделать их явными, зримыми, доступными для всех врачей и студентов-медиков.
6. Язык ДРАКОН кардинальным образом облегчает труд формализации процедурных медицинских знаний и повышает его производительность.
7. Широкомасштабное внедрение языка ДРАКОН для описания медицинских алгоритмов позволит сделать процедурные медицинские знания более точными, ясными и понятными. Это даст возможность повысить качество медицинского образования.

Глава 22

АЛГОРИТМЫ В ПРОМЫШЛЕННОСТИ

АЛГОРИТМ «ПРОВЕРКА САМОЛЕТА»

Современная промышленность — царство алгоритмов. Эти алгоритмы можно разделить на две части:

- выполняемые автоматически (классические алгоритмы),
- выполняемые вручную (жизнеритмы).

На рис. 134 изображена проверка самолета, где часть операций выполняется автоматически, часть — вручную.

Является ли проверка самолета алгоритмом? С классической точки зрения — нет. Потому что кое-что делается вручную. Примером ручных операций на рис. 134 являются ремонтные работы:

- ремонт двигателей;
- ремонт крыльев;
- ремонт шасси
- ремонт топливной системы;
- ремонт бортовых систем;
- ремонт системы пожаротушения;
- ремонт электрооборудования.

ЕДИНЫЙ СТАНДАРТ ДЛЯ АЛГОРИТМОВ И ЖИЗНЕРИТМОВ

Язык ДРАКОН предоставляет единый эргономичный стандарт для изображения и алгоритмов, и жизнеритмов. Стандарт в равной степени пригоден для описания как автоматических, так и ручных операций. В обоих случаях алгоритмическая конструкция силуэт является удобным средством для структуризации и декомпозиции алгоритмов и жизнеритмов. Рисунок 134 является хорошей иллюстрацией.

АЛГОРИТМ «КАК ДЕЛАЮТ ФРУКТОВЫЕ КОНСЕРВЫ»

На рис. 135 изображена технология изготовления сиропа и маринада. Показаны разнообразные физические действия, которые выполняет не компьютер, а различные технологические установки и агрегаты: мешкоопрокидыватель, вибросито, магнитная ловушка, бункер-накопитель и т.д.

Технология, показанная на рис. 135, спроектирована, реализована и эксплуатируется человеком. Следовательно, она является *человеческой деятельностью.* Схема на рис. 135 является не алгоритмом, а жизнеритмом.

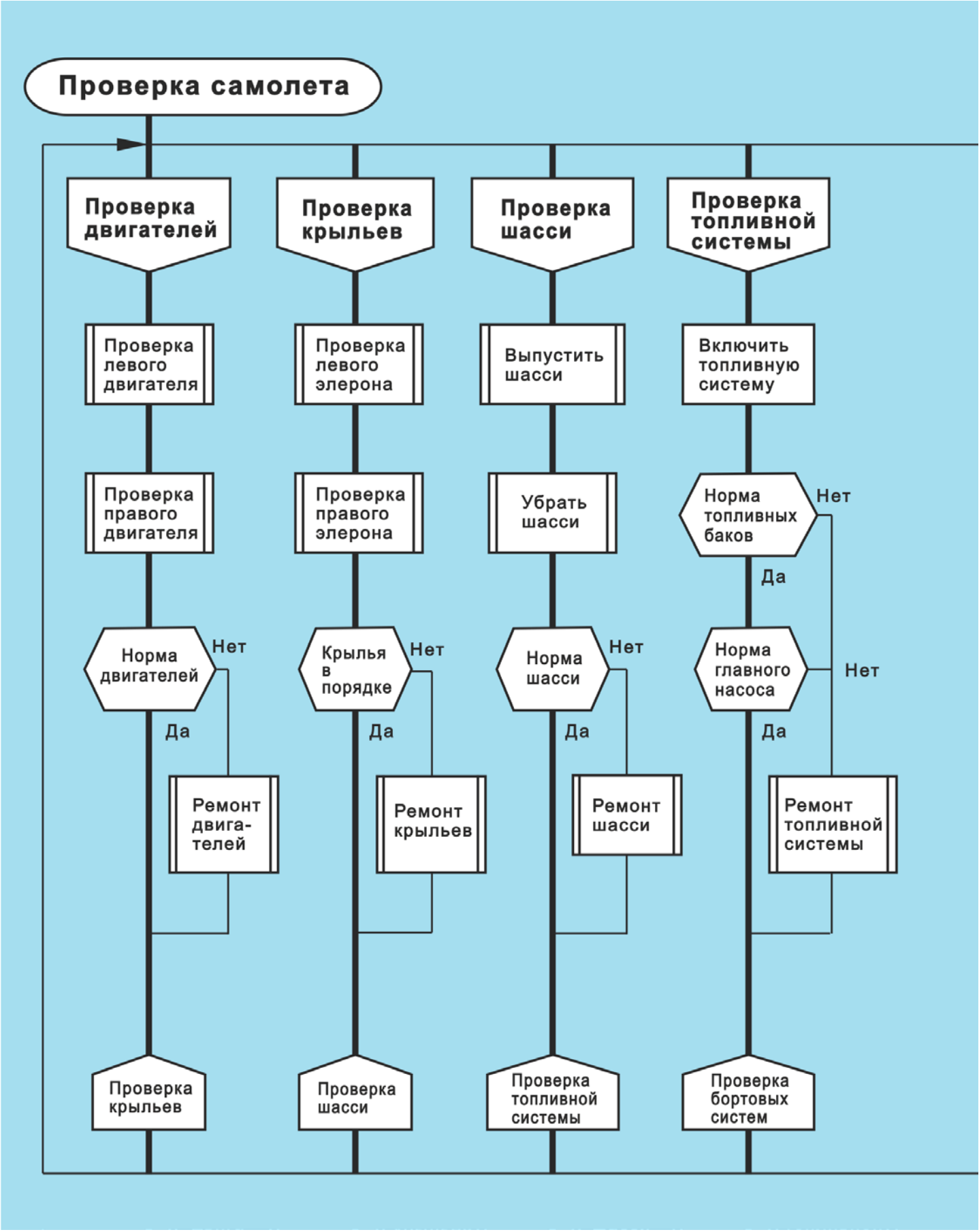
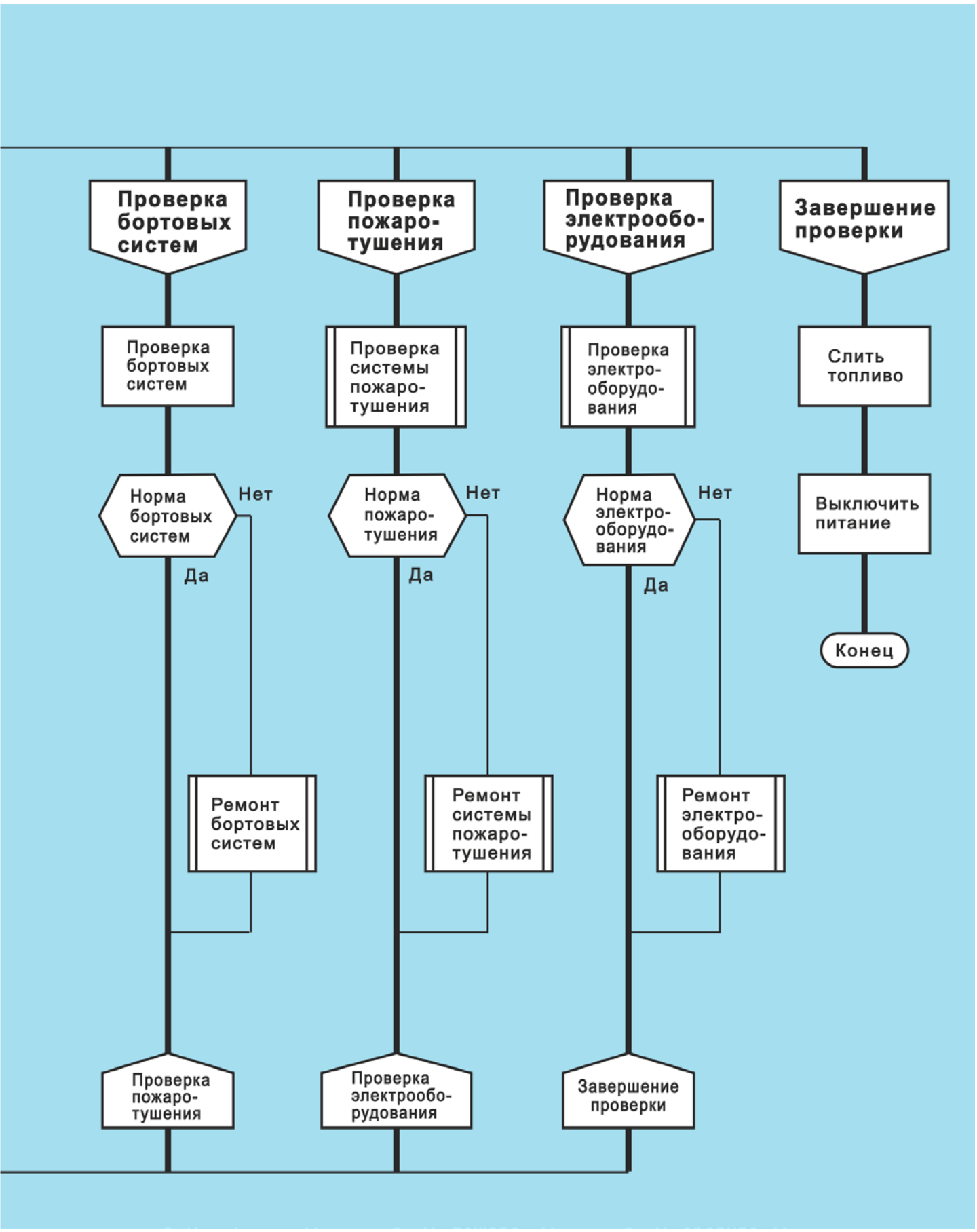


Рис. 134. Алгоритм «Проверка самолета»


Проверка бортовых систем
Проверка пожаро-тушения
Проверка электрообо-рудования
Завершение проверки
Проверка бортовых систем
Проверка системы пожаро-тушения
Проверка электро-оборудо-вания
Слить топливо
Норма бортовых систем
Нет
Да
Норма пожаро-тушения
Нет
Да
Норма электро-оборудо-вания
Нет
Да
Выключить питание
Конец
Ремонт бортовых систем
Ремонт системы пожаро-тушения
Ремонт электро-оборудо-вания
Проверка пожаро-тушения
Проверка электрообо-рудования
Завершение проверки

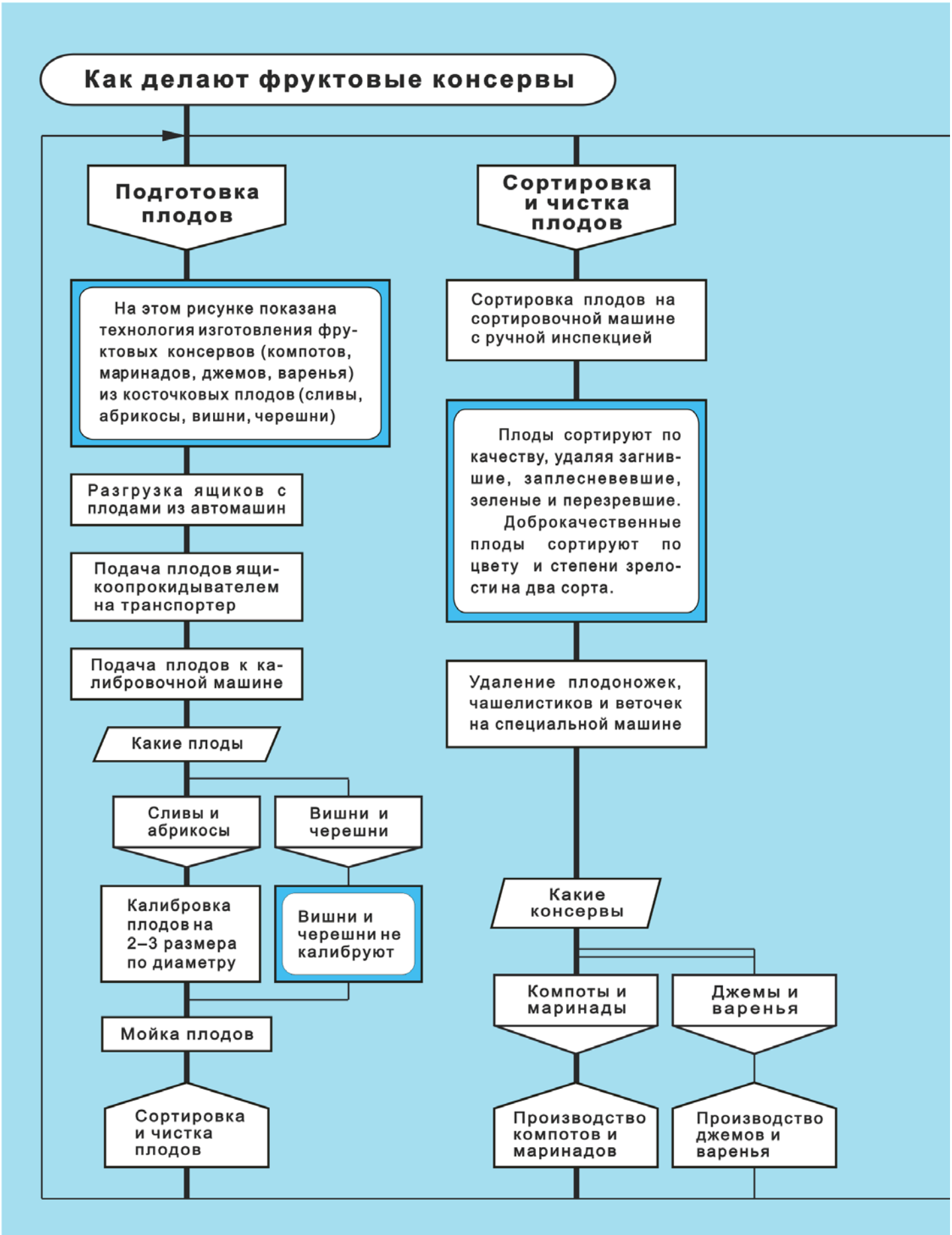
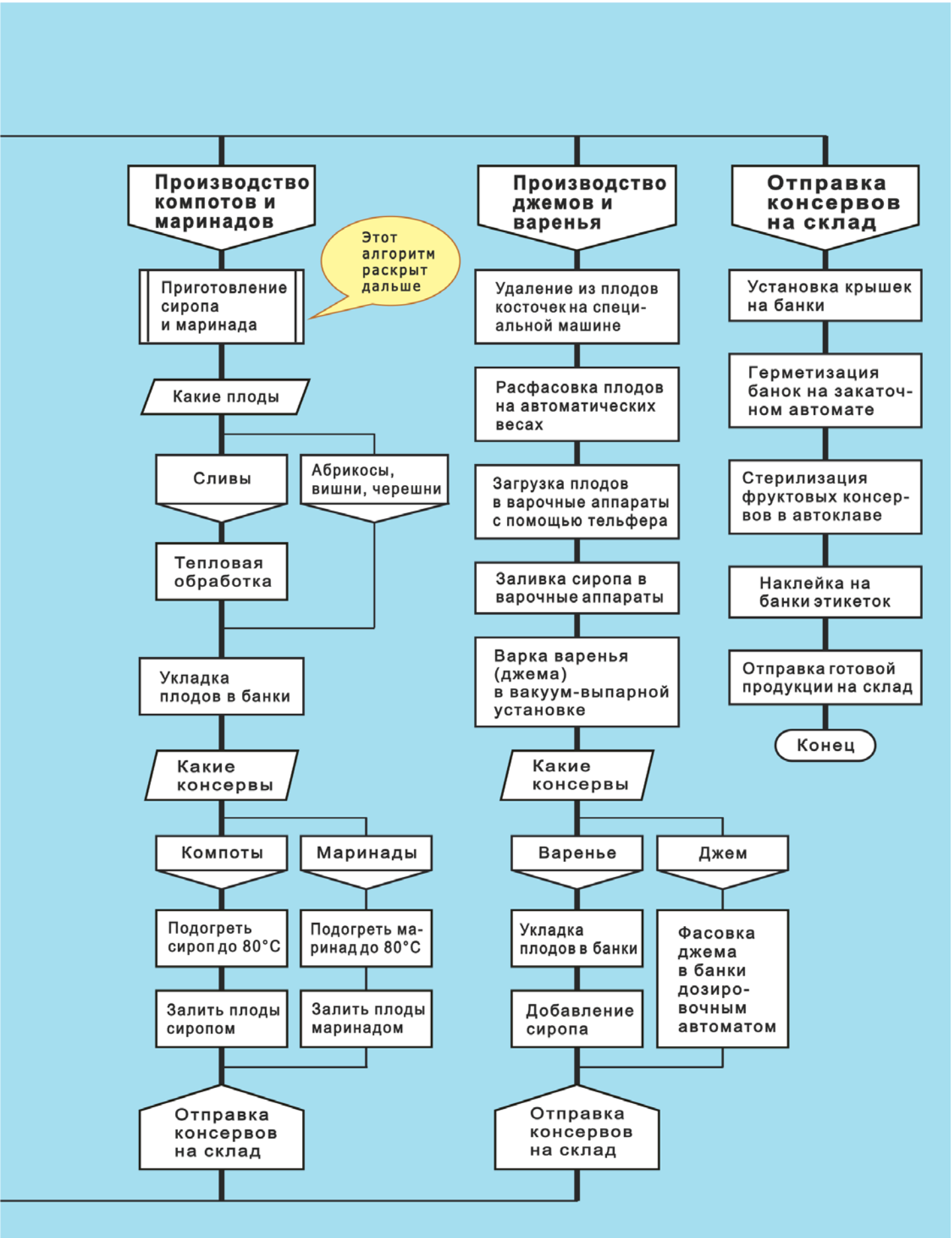


Рис. 135. Алгоритм «Как делают фруктовые консервы»


Производство компотов и маринадов
Этот алгоритм раскрыт дальше
Приготовление сиропа и маринада
Какие плоды
Сливы
Абрикосы, вишни, черешни
Тепловая обработка
Укладка плодов в банки
Какие консервы
Компоты
Маринады
Подогреть сироп до 80°С
Подогреть маринад до 80°С
Залить плоды сиропом
Залить плоды маринадом
Отправка консервов на склад
Производство джемов и варенья
Удаление из плодов косточек на специальной машине
Расфасовка плодов на автоматических весах
Загрузка плодов в варочные аппараты с помощью тельфера
Заливка сиропа в варочные аппараты
Варка варенья (джема) в вакуум-выпарной установке
Какие консервы
Варенье
Джем
Укладка плодов в банки
Добавление сиропа
Фасовка джема в банки дозировочным автоматом
Отправка консервов на склад
Отправка консервов на склад
Установка крышек на банки
Герметизация банок на закаточном автомате
Стерилизация фруктовых консервов в автоклаве
Наклейка на банки этикеток
Отправка готовой продукции на склад
Конец

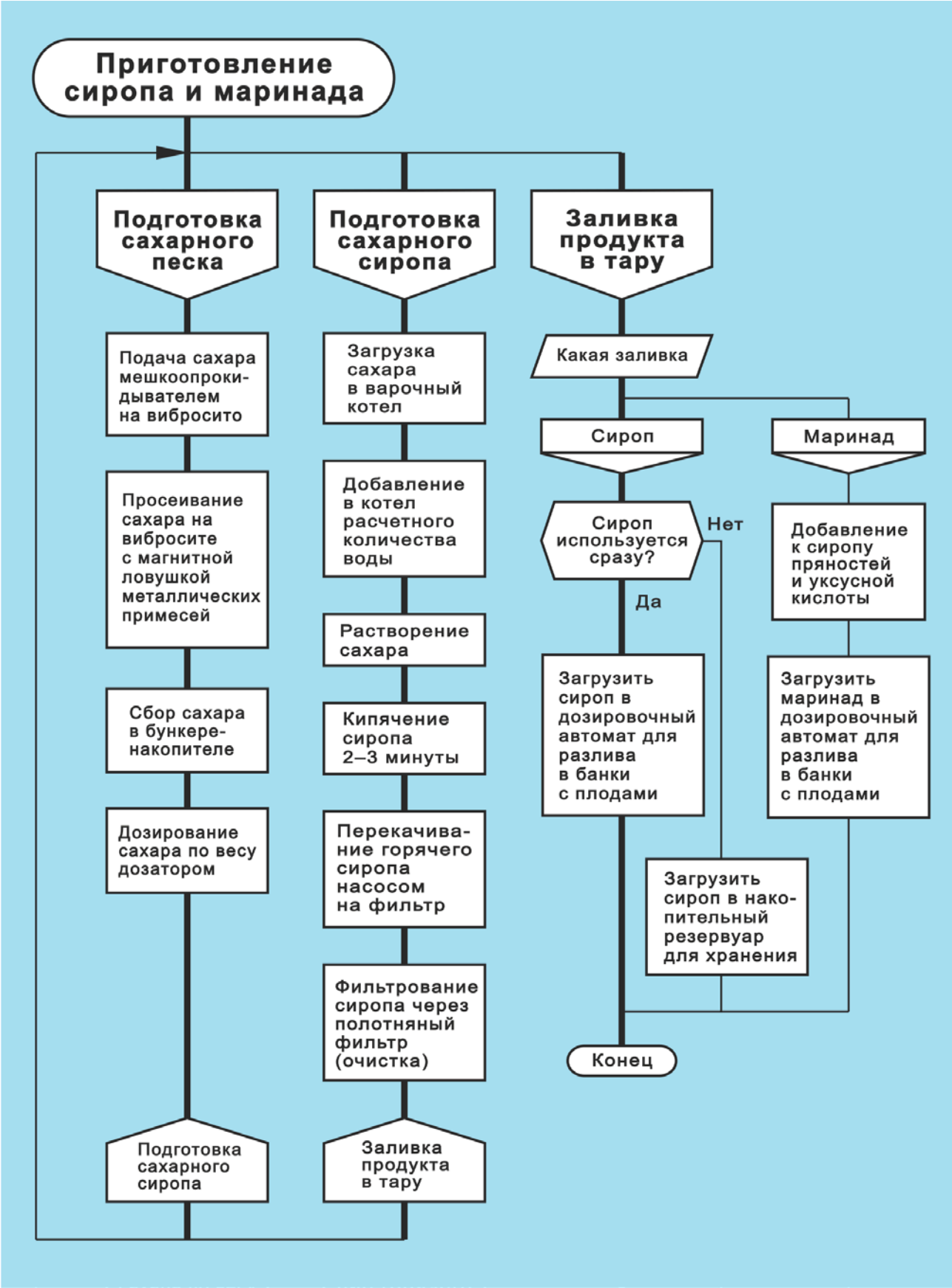


Рис. 136. Заводская технология изготовления сиропа и маринада

ИЗГОТОВЛЕНИЕ ФРУКТОВЫХ КОНСЕРВОВ

На рис. 135 и 136 представлен технологический процесс изготовления фруктовых консервов из косточковых плодов.

Реальный технологический процесс может быть очень сложным. Обычно его описывают как головной процесс, содержащий большое число вставок. Например, в процессе на рис. 135 показана вставка «Изготовление сиропа и маринада», раскрытая на рис. 136.

Эргономичная графика обладает серьезными преимуществами по сравнению с текстовым описанием технологических процессов. Она позволяет за короткое время (буквально за считанные минуты) составить подробное представление о технологических операциях и последовательности их выполнения.

ПАРАЛЛЕЛЬНОЕ ВЫПОЛНЕНИЕ ВЕТОК

Рассмотрим две ветки на рис. 135:

- производство компотов и маринадов;
- производство джемов и варенья.

Они могут выполняться одновременно (параллельно). Что для этого нужно? Во второй ветке ниже иконы Выбор «Какие консервы» имеется двойная горизонталь. Именно она обозначает Начало параллельных действий.

В иконах Адрес параллельных веток сделана одинаковая запись «Отправка консервов на склад». Одинаковая запись означает Конец параллельных действий.

ДРАКОН И ТЕХНОЛОГИЧЕСКИЕ ПРОЦЕССЫ

Дракон-схемы технологических процессов могут найти применение в следующих случаях:

- выпуск технологической документации;
- проектирование и моделирование технологических процессов;
- создание визуальной базы данных о техпроцессах;
- создание экспертных систем для проектирования технологических процессов, а также тренажеров для эксплуатационников;
- изготовление альбомов и каталогов технологических процессов для обучения или рекламы;
- создание наглядных плакатов, дающих целостное представление о процессе во всей его многосложности;
- при создании демонстрационных материалов в иконе Комментарий могут помещаться чертежи, фотографии, схемы установок, станков, сетей трубопроводов и другого оборудования.

ДРАКОН И ДЕЯТЕЛЬНОСТЬ

При проектировании сложных промышленных объектов необходимо иметь ЦЕЛОСТНЫЙ взгляд на проблему, показывающий все ее аспекты. На этом пути возникает трудность. В силу сложившихся привычек специалисты обычно делят проблему на две части:

- алгоритм, который можно превратить в компьютерную программу;

- операции, выполняемые вручную, которые невозможно превратить в программу.

В результате у проектировщиков сложных промышленных объектов нередко возникает не целостный взгляд на проблему, а кусочно-рваный, что может привести к неоптимальным или ошибочным решениям. Одной из причин является отсутствие языковых средств, помогающих разработчикам вырабатывать целостное видение проблемы.

Язык ДРАКОН позволяет устранить недостаток. Он дает разработчикам надежные средства, позволяющие изображать не кусочно-рваную, а целостную картину любых промышленных (и не только промышленных) алгоритмических процессов.

ВЫВОДЫ

1. Человеческие знания, используемые в промышленности, делятся на процедурные и декларативные.
2. Процедурные знания тесно связаны с человеческой деятельностью и трудовыми процессами. Они выявляют, закрепляют в сознании и объективируют структуру деятельности.
3. В настоящее время отсутствуют эффективные способы описания человеческой деятельности (работы). Язык ДРАКОН позволяет устранить этот пробел.
4. Процедурные знания охватывают алгоритмы и жизнеритмы.
5. Язык ДРАКОН позволяет единообразно, стандартным способом описать как алгоритмы, так и жизнеритмы.

Глава 23

АЛГОРИТМЫ В ТОРГОВЛЕ

РАЗНООБРАЗИЕ ТОРГОВЫХ АЛГОРИТМОВ

В торговле используется большое количество разнообразных алгоритмов. Примерами являются алгоритмы оптовой и розничной торговли для частных и бюджетных предприятий, бухгалтерские расчеты, управление складскими операциями, обработка платежных поручений и многое другое.

Все эти операции можно изобразить на языке ДРАКОН.

ДРАКОН представляет процедурные аспекты торговой деятельности в наиболее ясной и понятной форме. Понятной всем участникам торгового процесса — от рядового бухгалтера до руководителя торговой фирмы.

ПРОДАЖА ДЕТСКИХ ИГРУШЕК

На рис. 137 представлен алгоритм «Продажа детских игрушек». Суть дела поясним на примере. Оптовый покупатель желает закупить 1000 детских игрушек, например, куклу Барби. Возможны три ситуации:

- на складе продавца 5000 кукол Барби. В этом случае заявка оптовика будет удовлетворена полностью;
- на складе лишь 90 кукол Барби. Тут возможны варианты. Оптовик либо берет то, что есть (покупает 90 кукол), либо отказывается от товара;
- на складе нет ни одной куклы, стало быть, сделка сорвалась.

Силуэт состоит их четырех веток:

- Оформление заказа.
- Выставление счета.
- Отмена заказа на игрушки.
- Завершение.

Первая ветка (оформление заказа) полностью отражает описанные выше ситуации. Вторая (выставление счета) содержит действия:

- рассчитай, сколько стоит заказ;
- выставь счет покупателю;
- покупатель согласен оплатить счет? и т.д.

Третья говорит об отмене заказа на игрушки.

ПРОДАЖА АВИАБИЛЕТОВ

В следующем алгоритме речь идет о продаже авиабилетов (рис. 138).

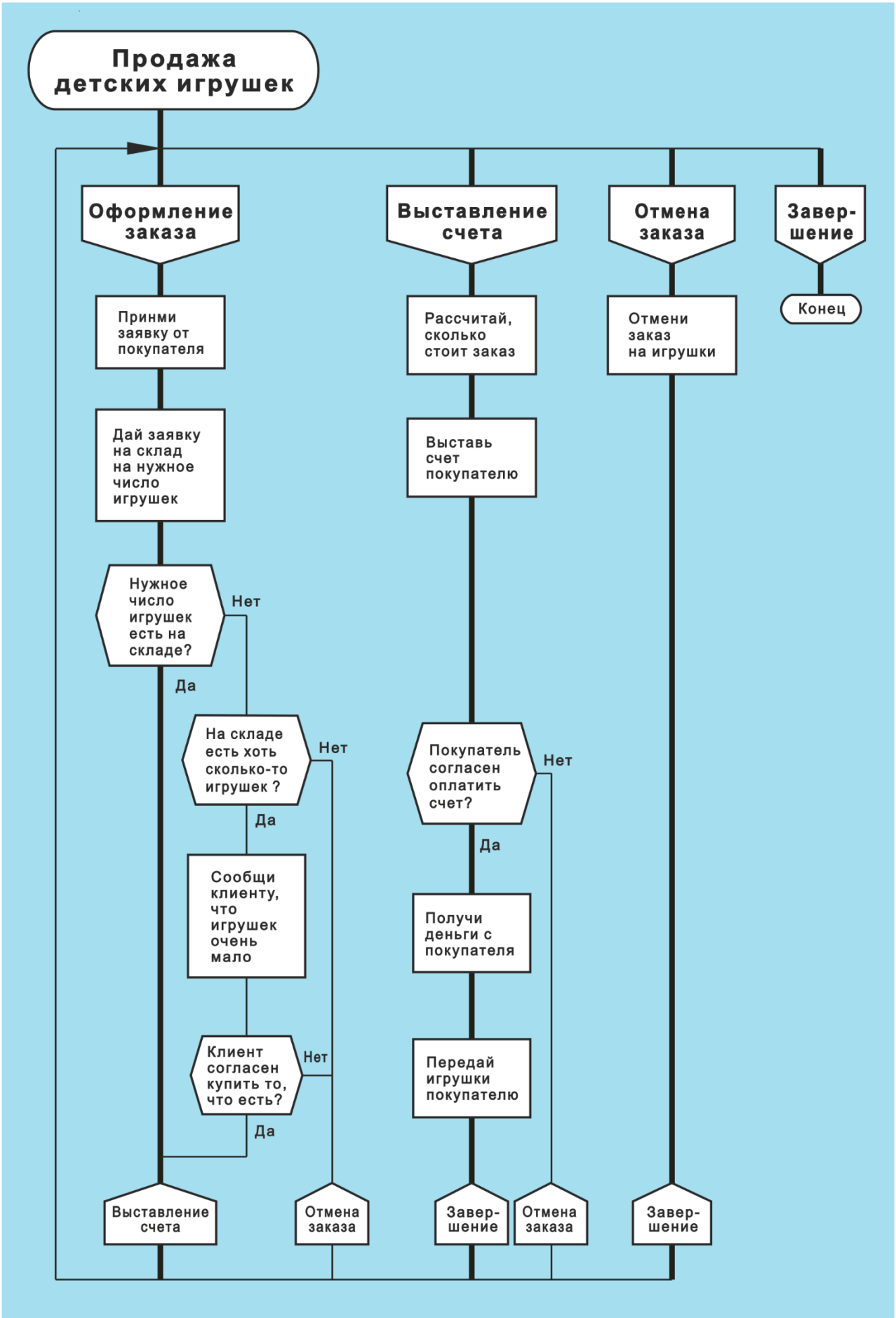


Рис. 137. Алгоритм «Продажа детских игрушек»

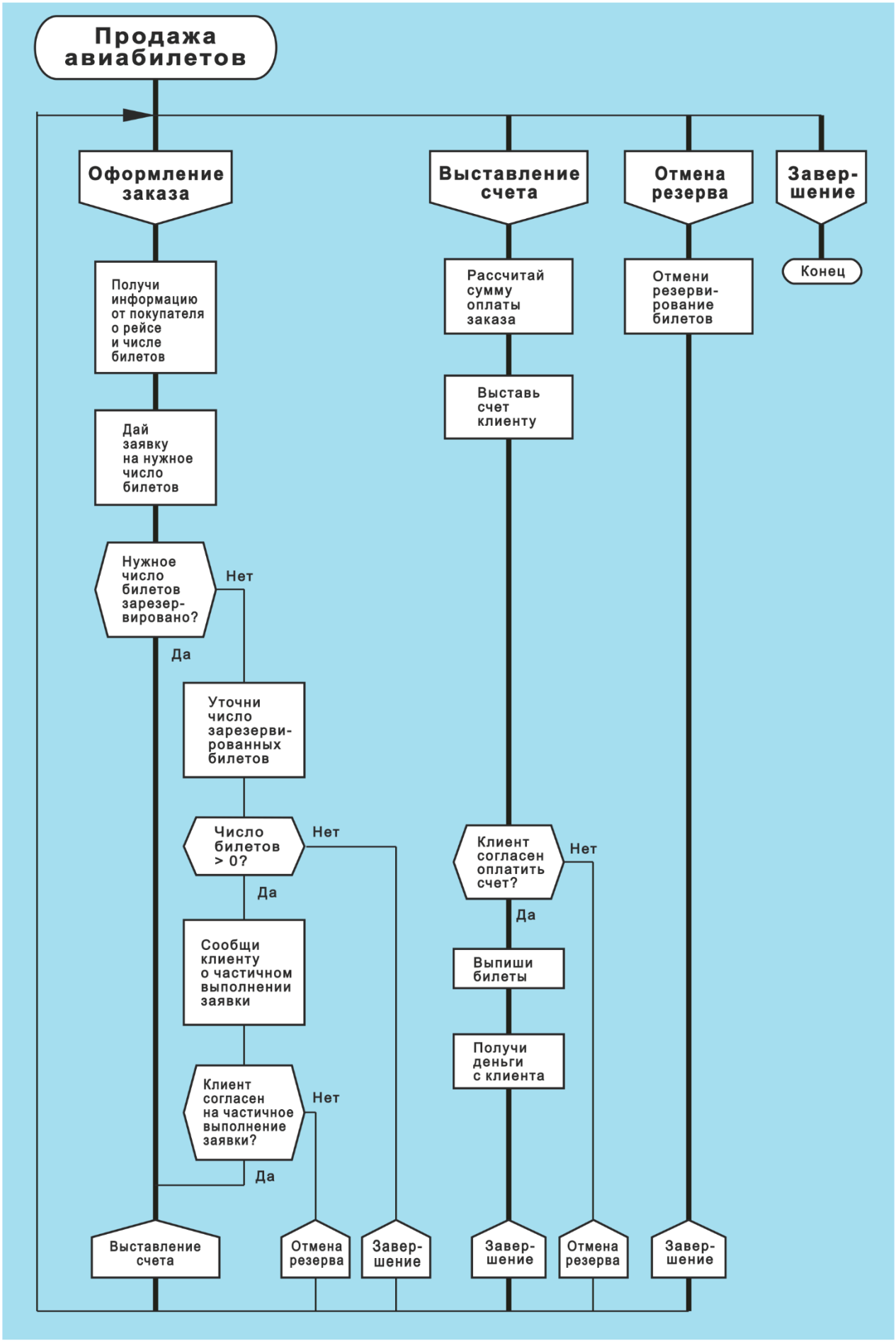


Рис. 138. Алгоритм «Продажа авиабилетов»

Данный алгоритм отчасти похож на предыдущий. Отличие в том, что продажа игрушек описана упрощенно (популярно), а продажа авиабилетов изложена профессиональным языком. Разница хорошо видна в таблице.

Продажа игрушек	Продажа авиабилетов
Дай заявку на склад на нужное число игрушек	Дай заявку на нужное число авиабилетов
Нужное число игрушек есть на складе?	Нужное число билетов зарезервировано?
На складе есть хоть сколько-то игрушек?	Число билетов > 0?
Сообщи клиенту, что игрушек очень мало	Сообщи клиенту о частичном выполнении заявки
Клиент согласен купить то, что есть?	Клиент согласен на частичное выполнение заявки?
Отменить заказ на игрушки	Отменить резервирование билетов

ВЫВОДЫ

1. Торговля, как человеческая деятельность, требует высокого уровня взаимопонимания между заказчиками, потребителями и разработчиками программных комплексов, обслуживающих торговые операции.
2. Взаимопонимание является труднодостижимой целью. Недостаточное взаимопонимание снижает производительность труда и влечет за собой упущенную выгоду.
3. Одна из причин недостатка состоит в том, что отсутствуют наглядные языковые средства:
   - пригодные для эффективного описания алгоритмов, используемых в торговой деятельности,
   - способные обеспечить быстрое взаимопонимание между заказчиками, потребителями и разработчиками программных комплексов торгового назначения.
4. Язык ДРАКОН позволяет устранить пробел и добиться повышения производительности труда в этой области.

Глава 24

АЛГОРИТМЫ В БИОЛОГИИ

РУКОТВОРНЫЕ И НЕРУКОТВОРНЫЕ АЛГОРИТМЫ

До сих пор мы рассуждали об алгоритмах, которые придуманы человеком. Но существуют и иные процессы, к созданию которых человек не имеет отношения. Имеются в виду нерукотворные (естественные) алгоритмы, порожденные эволюцией природных процессов.

Планета Земля — царство жизни. В каждом живом организме происходят процессы огромной сложности, необходимые для поддержания жизни. Рассматривая жизнь с точки зрения молекулярной биологии, генетики и других биологических наук, мы сталкиваемся со сложными процессами, которые можно и нужно назвать алгоритмами. Алгоритмы существуют всюду, где есть жизнь. Они ухитряются жить в каждом живом существе, в каждой живой клетке, от холерного вибриона до незабудки и слона.

Какова роль человека в естественных алгоритмических процессах? Человек не проектирует и не создает эти алгоритмы. Он всего лишь *изучает* их.

Человечество заинтересовано в том, чтобы биологические науки успешно развивались. От чего зависит скорость познания естественных алгоритмов? Во многом, от формы представления алгоритмических процессов.

К сожалению, представители биологических наук не владеют эффективными методами записи естественных алгоритмов. Язык ДРАКОН может оказать биологам существенную помощь.

ВИЗУАЛИЗАЦИЯ БИОЛОГИЧЕСКИХ АЛГОРИТМОВ

Чем глубже человеческий разум проникает в тайны живой материи, тем яснее становится, что живые существа во многих случаях ведут себя, как информационные биомашины, перерабатывающие информацию с помощью биоалгоритмов.

Опыт показывает, что биологические алгоритмы очень похожи на обычные алгоритмы, с которыми мы постоянно сталкиваемся в технике. А раз так, язык ДРАКОН может стать удобным средством для выражения и накопления знаний об алгоритмических процессах, протекающих в живых организмах.

Начнем с простого примера и приведем цитату из учебника.

Как работает сердце

«Наше сердце постоянно в работе… Оно работает непрерывно 70—80 лет и более. В чем секрет его неутомимости?...

Во многом это объясняется особенностями работы сердца. Оно последовательно сокращается и расслабляется с короткими промежутками для отдыха. В одном сердечном цикле можно выделить три фазы.

«Во время первой фазы, которая у взрослого человека длится 0,1с, сокращаются предсердия, а желудочки находятся в расслабленном состоянии. За ней следует вторая фаза (она более продолжительная — 0,3с): желудочки сокращаются, а предсердия расслаблены. После этого наступает третья, заключительная фаза — *пауза*, во время которой происходит общее расслабление сердца. Ее продолжительность 0,4с. Весь сердечный цикл занимает 0,8с.

«Вы видите, что в течение одного сердечного цикла предсердия тратят на работу 12,5% времени сердечного цикла, а желудочки 37,5%. Остальное время, а это 50%, сердце отдыхает.

«В этом секрет долголетия сердца, удивительной его работоспособности. Небольшие промежутки отдыха, следующие за каждым сокращением, дают возможность сердечной мышце отдохнуть и восстановить силы» [99].

Работа сердца очень сложна. Поэтому мы выбрали лишь малую часть работы нашего главного насоса. И представили ее на рис. 139 в виде простого алгоритма.

КАК И ПОЧЕМУ РАЗМНОЖАЮТСЯ ЖАБЫ

Рассмотрим еще один пример. Прочитаем отрывок из известной биологической книги, описывающий алгоритм размножения жаб.

«Рассмотрим изменения, происходящие в организме жабы в сезон размножения. Глаза жабы воспринимают свет и передают эту информацию в мозг, который определяет, что продолжительность светового дня увеличивается.

«Гипоталамус направляет в гипофиз соответствующие рилизинг-факторы. Гипофиз начинает выделять в кровь различные гормоны…

«Когда семенники и яичники «обнаруживают» их присутствие в крови, они начинают увеличиваться в размерах, продуцировать гаметы, а также выделять собственные гормоны и среди них — половые: тестостерон и эстроген. Реагируя на половые гормоны, мозг посылает нервные импульсы к мышцам — животное начинает поиск места для размножения и брачного партнера.

«Так, благодаря сложному взаимодействию органов чувств, нервов, мозга, мышц и эндокринных желез животное адекватно реагирует на смену сезона — наступление весны» [98].

Описание этого алгоритма на языке ДРАКОН показано на рис. 140. Чтобы не утомлять читателя громоздкими биологическими терминами, мы упростили алгоритм. А в комментариях дали необходимые пояснения.

Алгоритм содержит четыре ветки:

- Как жаба узнает, что пришла весна.
- Как мозг жабы передает приказ в половые железы.
- Как половые железы жабы передают в мозг ответ.
- Как жаба ищет жениха и невесту.

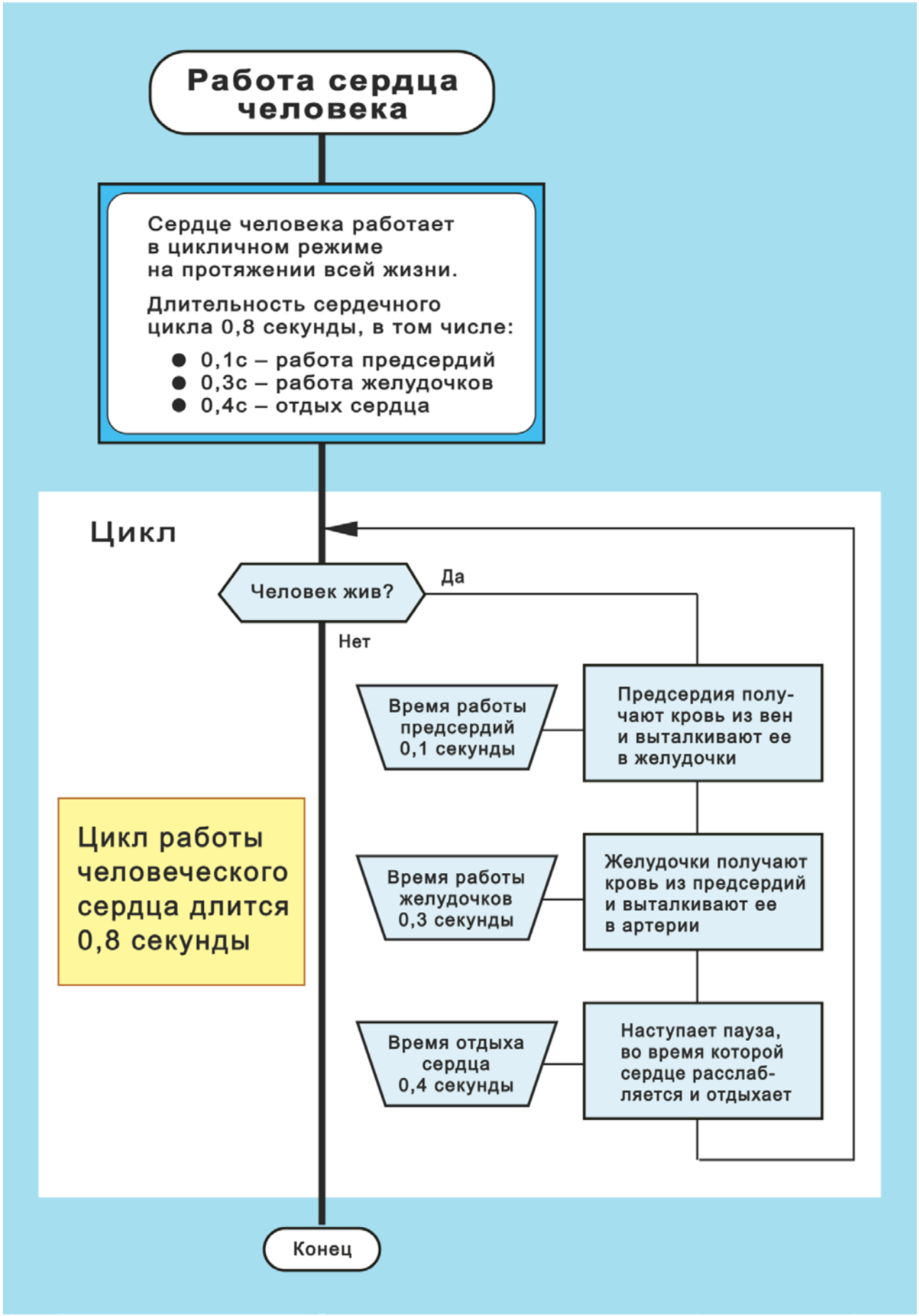


Рис. 139. Сердце человека всю жизнь работает в цикле (цикл ПОКА)

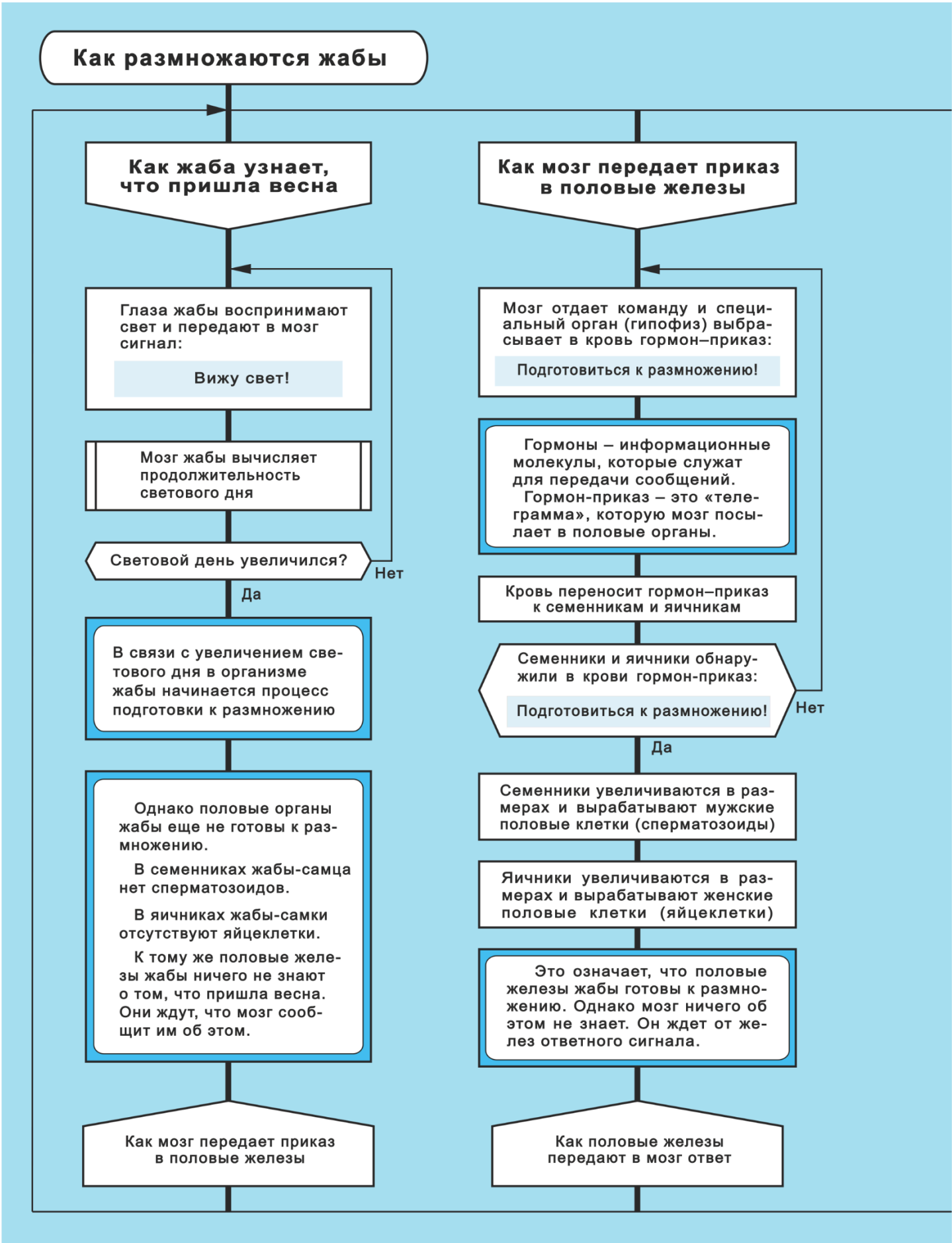
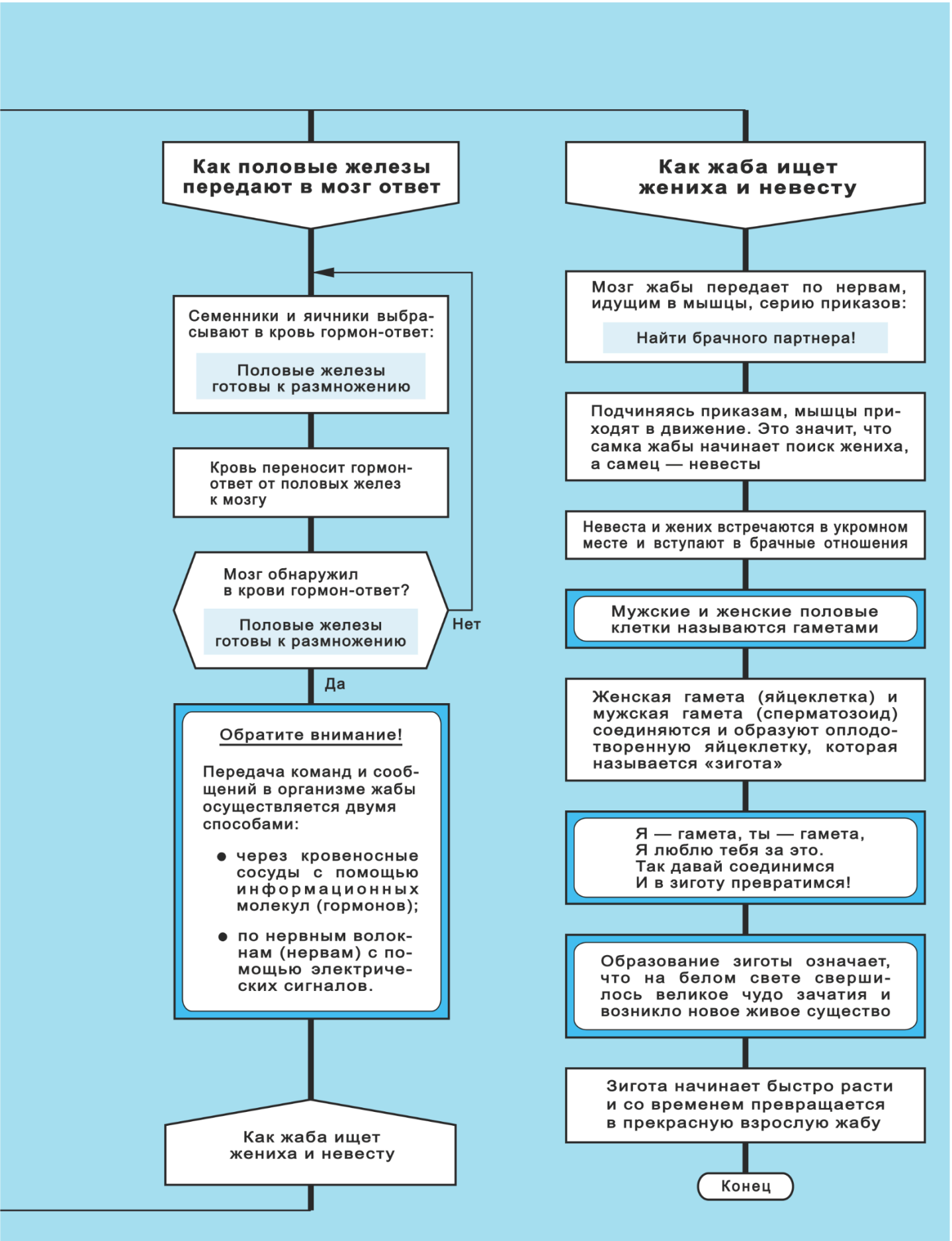


Рис. 140. Алгоритм «Как размножаются жабы»



Использование языка ДРАКОН для описания биологических процессов

НУЖНЫ НОВЫЕ СРЕДСТВА ДЛЯ ОПИСАНИЯ БИОЛОГИЧЕСКИХ АЛГОРИТМОВ

Существующие биоалгоритмы имеют низкое качество. Они изложены трудным, запутанным языком. Это вносит значительные трудности в работу биологов. Современная биология не умеет говорить на языке алгоритмов.

Чтобы биологическая наука интенсивно развивалась, она должна овладеть новым языком для описания биологических знаний. Биология должна освоить алгоритмический язык. Это создаст условия для разделения биологических знаний на процедурные и декларативные. Алгоритмы позволят глубже понять механизмы поведения живых клеток и живых организмов. Новый, алгоритмический язык создаст новые возможности и позволит использовать высококачественные биологические алгоритмы в научных исследованиях.

ЯЗЫК ДРАКОН И БИОЛОГИЯ

Язык ДРАКОН может оказать существенную помощь биологам. Визуализация биологических алгоритмов, создание бумажных альбомов и компьютерных библиотек биологических дракон-алгоритмов даст возможность улучшить форму представления биологических знаний.

С помощью ДРАКОНа можно облегчить решение ряда задач:

- выявить и устранить алгоритмические пробелы в биологических знаниях;
- укрепить позиции информационной биологии;
- облегчить дальнейшее исследование механизма функционирования живых организмов.

ВЫВОДЫ

1. Алгоритмы удобно разделить на две группы:
   - рукотворные (создаваемые человеком и выполняемые компьютером или вручную);
   - нерукотворные (биологические) алгоритмы.
2. В настоящее время отсутствуют эффективные способы описания биологических алгоритмов. Язык ДРАКОН позволяет успешно решить задачу.
3. До сих пор различные типы алгоритмов (рукотворные и нерукотворные) было принято описывать по-разному, с помощью различных и зачастую неудобных средств.
4. Язык ДРАКОН устраняет этот разнобой. Он позволяет описывать все алгоритмы *единообразно*, стандартным способом, с помощью одной и той же удобной системы чертежей (одной и той же эргономичной графической нотации).

Глава 25

АЛГОРИТМЫ В СЕЛЬСКОМ ХОЗЯЙСТВЕ

ПОМИДОРЫ НА САДОВОМ УЧАСТКЕ

Выращивание помидоров кажется простым делом. Так ли это? С позиций агрономической науки это, конечно, не так. Чтобы получить высокий урожай аппетитных плодов, необходимо тщательно соблюдать технологию посева и ухода за помидорами. А она отнюдь не проста.

Рассмотрим пример. На рис. 141 показана технология (алгоритм) выращивания помидоров на садовом участке. Алгоритм делится на три ветки, причем каждая имеет имя. Это не просто имена, расположенные в верхнем ряду схемы. Это эргономичные путеводители по алгоритму. Они дают содержательное название каждому шампуру и разбивают алгоритм на три смысловые части:

- Подготовка и посев семян.
- Уход за сеянцами и рассадой.
- Работа в теплице.

Но этого мало. Дробление алгоритма производится по двум основаниям:

- деление на ветки;
- деление на процедуры.

Алгоритм содержит девять процедур. Три из них раскрыты на рисунках 142-144.

На рис. 142 изображен алгоритм «Подготовка семян для проращивания».

На рис. 143 — алгоритм «Посев семян в ящики с грунтом».

На рис. 144 — алгоритм «Уход за сеянцами в ящике».

Алгоритмы заимствованы из книги [14]. На обложке сказано: «Точное, последовательное и наглядное описание каждого шага, ведущего к обильному урожаю». В этой книге все технологические операции показаны в виде развернутых дракон-схем.

ДРАКОН-СХЕМЫ ПОЛЕЗНО ДОПОЛНИТЬ РИСУНКАМИ И КАРТИНКАМИ

Чтобы увеличить наглядность, в книге [14] использован полезный прием. Каждая дракон-схема окружена рисунками. Причем рисунки расположены очень удобно — они прижаты к той дракон-иконе, содержание которой нужно объяснить.

Что это дает? Предположим, в иконе говорится о марлевом мешочке с семенами, к которому приделан ярлычок (рис. 142). Как только читатель прочитал эти слова, он тут же по соседству *видит* этот мешочек, из-под резинки которого торчит бумажный ярлычок.

Еще пример. На рис. 143 в первой ветке говорится о том, что «всходы имеют вид петелек». Рядом нарисованы эти самые петельки.

Рис. 141. Технология выращивания помидоров на садовом участке

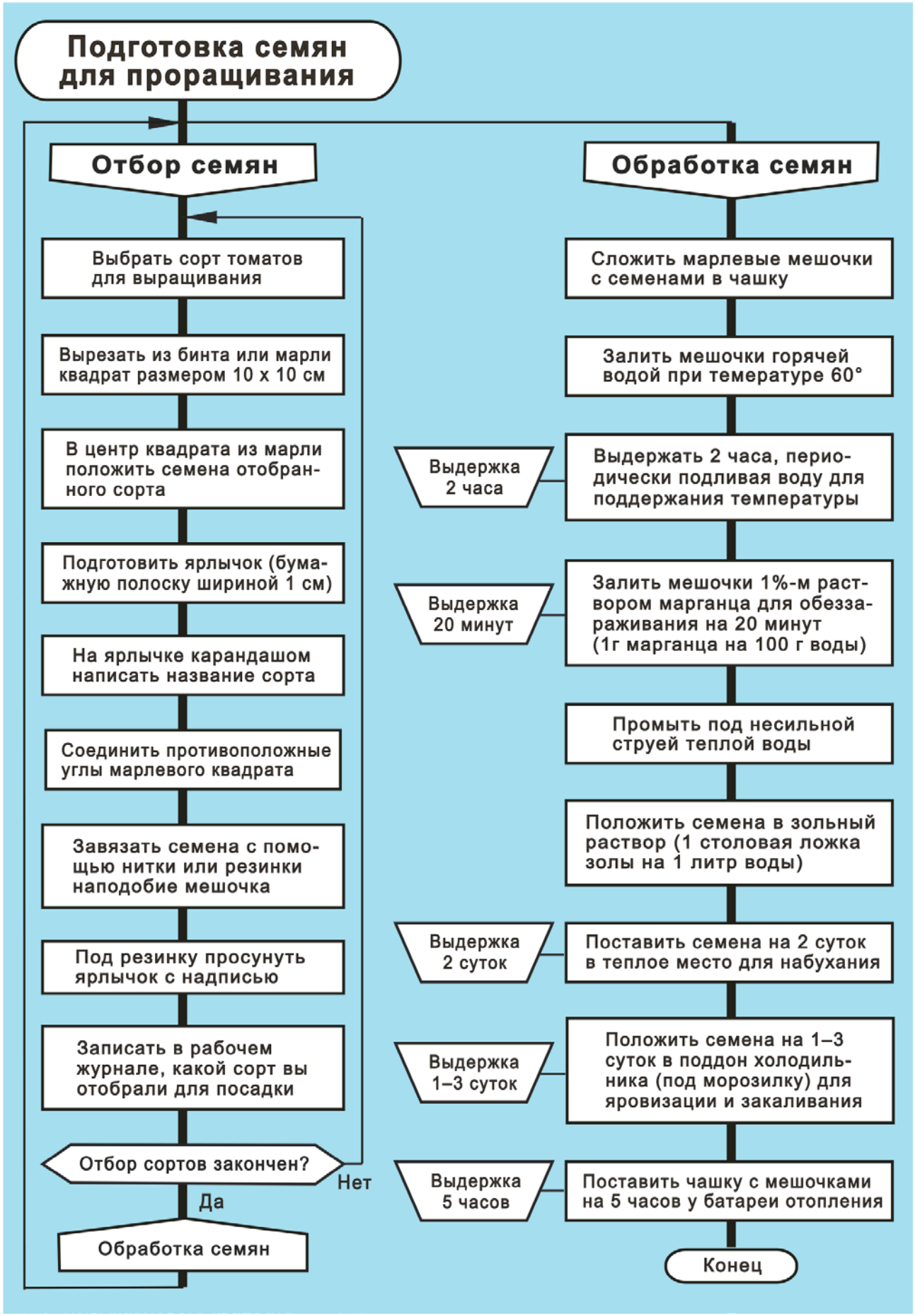


Рис. 142. Алгоритм «Подготовка семян для проращивания»

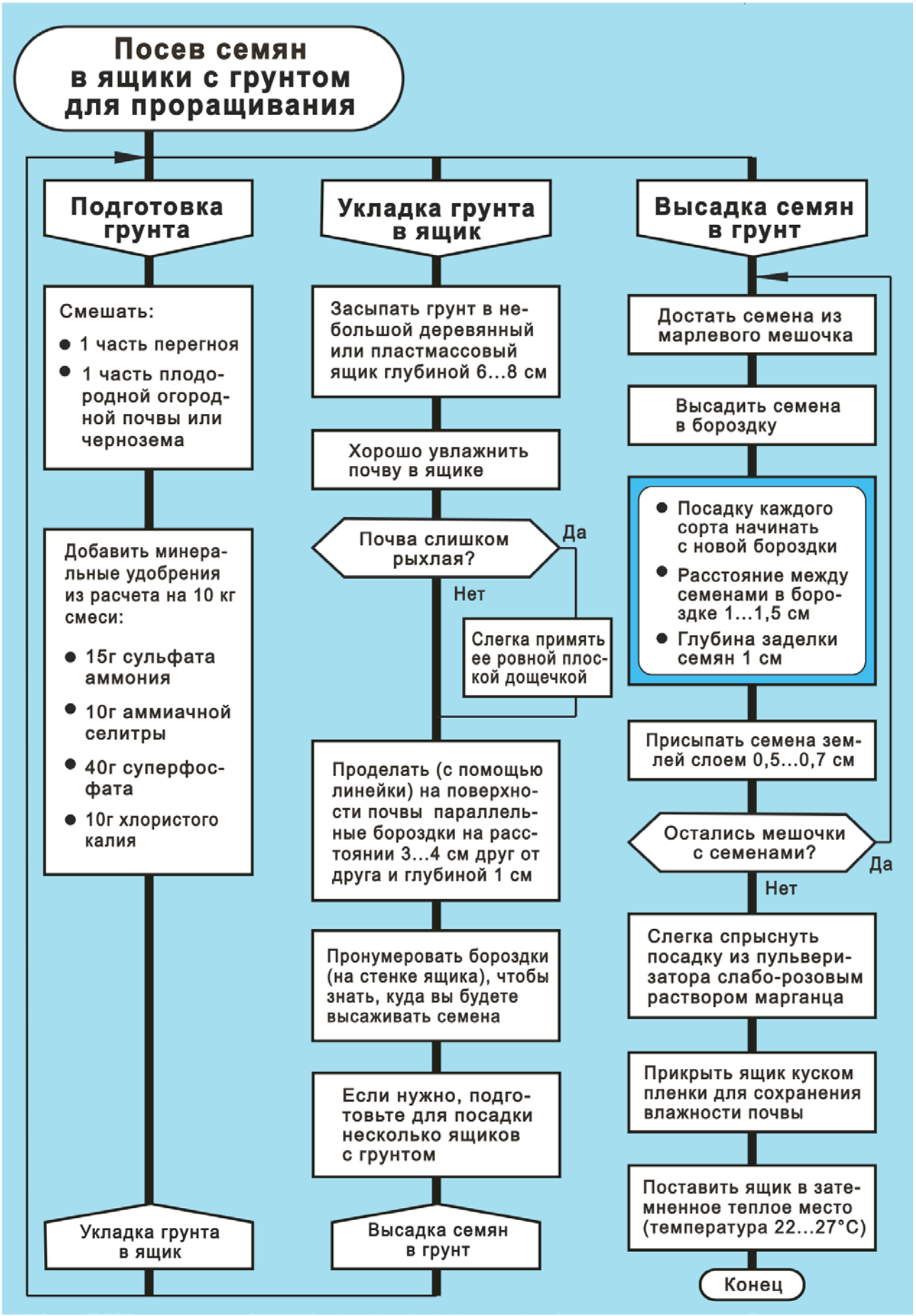


Рис. 143. Алгоритм «Посев семян в ящики с грунтом для проращивания»

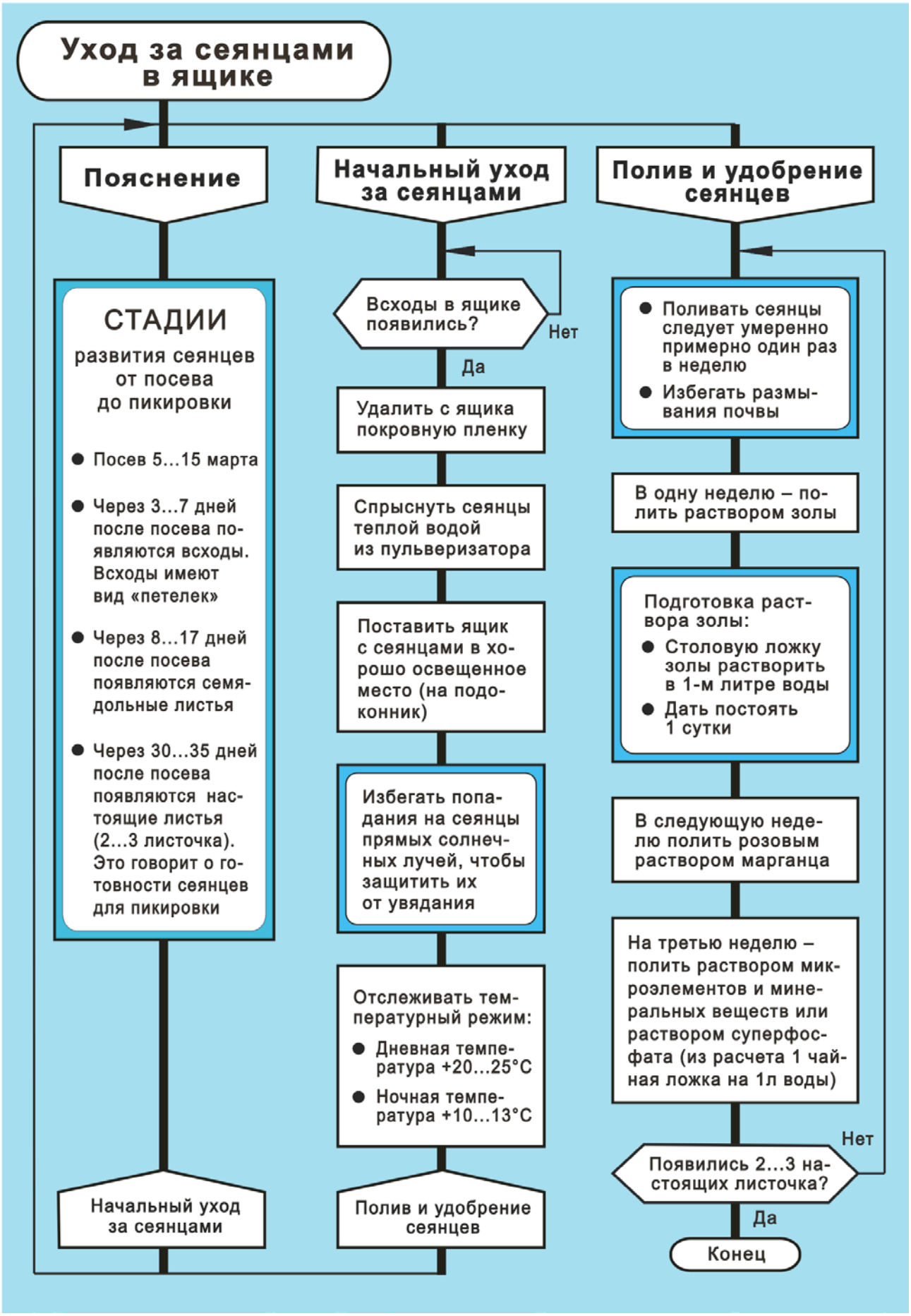


Рис. 144. Алгоритм «Уход за сеянцами в ящике»

Дальше речь идет о семядольных и настоящих листьях. Что это за листья?

У читателей такой вопрос не возникает. В книге прекрасно видны эти листья — на рисунках, дополняющих дракон-схему. Так что отличие между семядольными и настоящими листьями сразу бросается в глаза. Читатель не затрачивает никаких усилий, чтобы уяснить понятия.

ГРАФИЧЕСКИЙ КОММЕНТАРИЙ

Дракон-схема — это графический образ. Естественно, возникает вопрос: если речь зашла о графике, надо использовать ее не частично, а в полной мере.

Такую задачу можно решить с помощью иконы Комментарий. В ДРАКОНе возможен не только традиционный (текстовый), но и графический комментарий. Пример приведен в следующей главе на рис. 145.

Там же приведен и комментарий в виде математических формул (формульный комментарий).

ВЫВОДЫ

1. Язык ДРАКОН способен
   - наглядно изображать процессы в сельском хозяйстве,
   - описывать сельскохозяйственные и агропромышленные технологии.
2. Язык ДРАКОН можно использовать для описания алгоритмов, которые используются в агрохимических и агрофизических науках, а также в почвоведении.

Глава 26

АЛГОРИТМЫ В СРЕДНЕЙ ШКОЛЕ

ДРАКОН ПОМОГАЕТ ИЗУЧАТЬ ГЕОМЕТРИЮ

На рис. 145 представлена школьная задача по геометрии. Задача структурирована по правилам языка ДРАКОН.

В первой ветке дано условие задачи с чертежом.

Во второй — вычисление площади круга и треугольника.

В третьей — вычисление искомой площади *S.*

Чем отличается решение математической задачи с помощью дракон-схемы? Укажем три отличия.

- Обеспечена легкость восприятия: в одном визуальном поле находятся: 1) условие задачи; 2) решение; 3) ответ.
- Зрительная сцена имеет четкую структуру. Она упорядочена и по вертикали, и по горизонтали.
- Все этапы решения и формулы имеют разъясняющие словесные заголовки. Последние записываются не где угодно, а в специальных рамочках «Имя ветки», каждая из которых «знает свое место».

ВИЗУАЛЬНОЕ МЫШЛЕНИЕ

Чертеж на рис. 145 позволяет:

- рассматривать математическое решение задачи как зрительную сцену;
- правильно строить математическую зрительную сцену на плоскости чертежа;
- изображать решение математических задач, выявляя их визуальную структуру;
- структурировать задачу в пространстве чертежа по правилам языка ДРАКОН;

В основе языка ДРАКОН лежат правила визуального мышления.

Чтобы облегчить труд школьников, желательно разбивать ход решения задачи на отдельные этапы (ветки силуэта) и придумать для них краткие и точные надписи. Надписи следует записывать в шапке дракон-схемы: в иконах Заголовок и Имя ветки.

ИКОНА «КОММЕНТАРИЙ», СОДЕРЖАЩАЯ ЧЕРТЕЖ

Икона Комментарий обладает богатыми возможностями. Внутри ее можно размещать различные объекты. На рис. 145 показаны четыре иконы Комментарий, в которых находятся:

- чертеж (круг и треугольник);
- текст;
- формула.

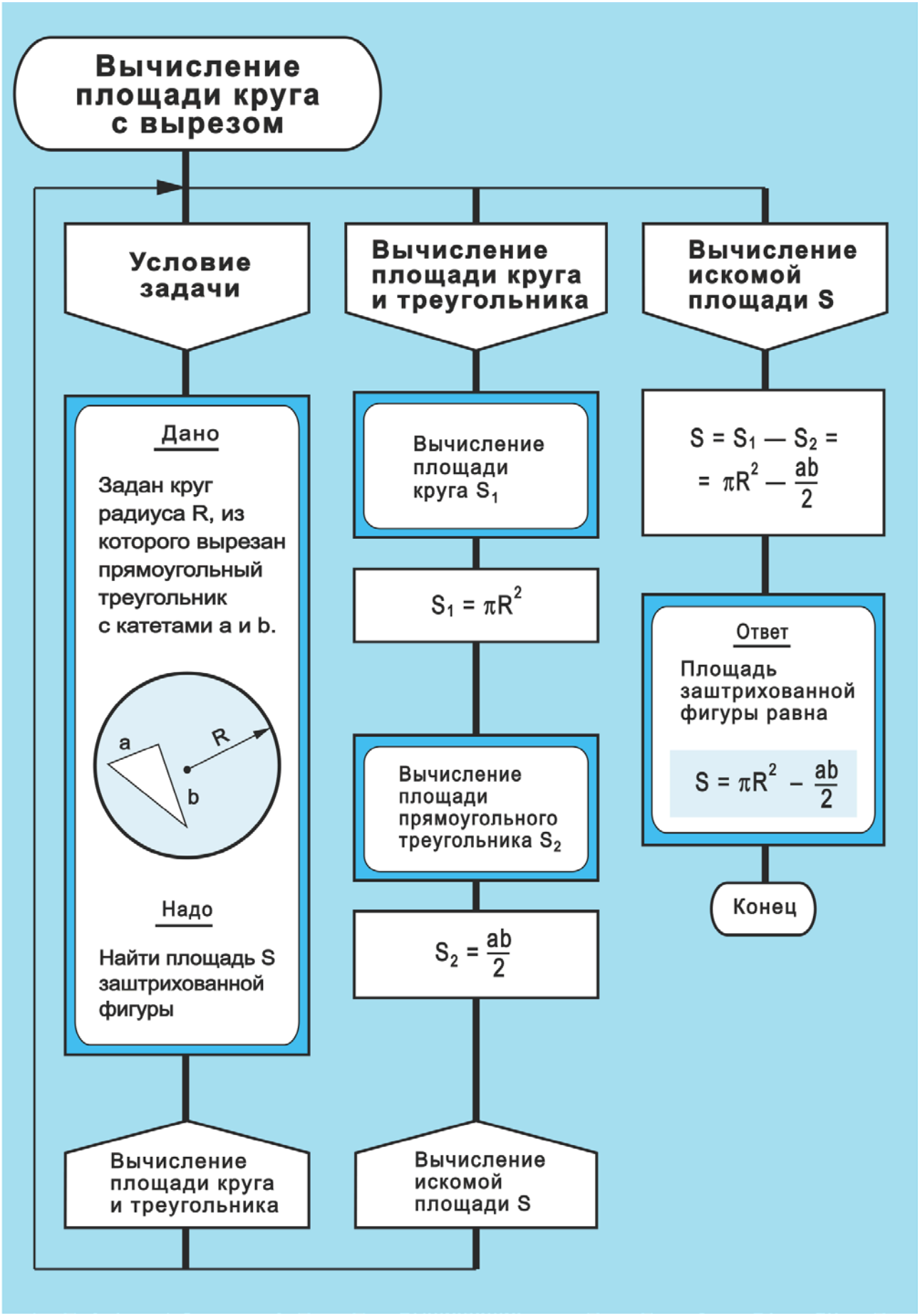


Рис. 145. Вычисление площади круга с вырезом

О ПОЛЬЗЕ ЭРГОНОМИЧНЫХ СТРЕЛОК В МАТЕМАТИКЕ

Рассмотрим алгоритм на рис. 146, который содержит «маленький секрет», связанный с числом 4. Действительно, в условии задачи мы видим 4 отдельных условия (см. икону Комментарий). В ветке «Решение задачи» имеются 4 вертикали, помеченные буквами *A, B, C, D.* Таким образом, в данной задаче действует

Правило	Число условий равно числу вертикалей. Причем каждому условию соответствует своя вертикаль.

Копнув глубже, можно заметить, что в задаче выполняется закономерность:

- каждому условию соответствует свое решение,
- каждое решение изображается отдельной вертикалью.

В обычных условиях указанные закономерности невидимы — они скрыты от читателя. Человек может правильно решить задачу, даже не заметив, что в ней затаился «маленький секрет». Это значит, что он решил задачу поверхностно, не проникнув в ее тайные глубины.

Чтобы устранить недостаток, надо сделать читателю небольшую, но умную эргономическую подсказку. Такой подсказкой служат 4 широкие светлые горизонтальные стрелки. Каждая из них устанавливает зрительную связь между условием и соответствующей вертикалью. Эти стрелки сразу бросаются в глаза и помогают читателю разгадать секрет.

Разумеется, можно обойтись и без стрелок. Но если мы хотим проявить заботу о читателе, облегчить его утомительный труд и оказать ему дополнительную помощь, то стрелки и другие эргономические мелочи, несомненно, будут полезны.

Эту мысль следует подчеркнуть особо. С логической точки зрения, указанные стрелки являются «архитектурным излишеством» и совершенно не нужны для решения задачи.

Но с эргономической точки зрения, дело обстоит иначе.

> Цель эргономики — облегчение умственного труда, минимизация интеллектуальных усилий, достижение «легкости и глубины мышления». Именно это и достигается. Указывая на нужную вертикаль, стрелки помогают читателю проникнуть в глубинную суть задачи.

Каждая вертикаль имеет два интересных конца: верхний и нижний. Верхний конец содержит алгоритмическую запись *условия*, нижний — *решения*. Это математическое богатство открывается ученику благодаря стрелкам.

Если от правого острия стрелки читатель поднимет глаза вверх, он увидит алгоритмическую реализацию условия. Если же посмотрит вниз — увидит решение. Поскольку стрелок четыре, подобную операцию придется проделать четыре раза. Но после этих упражнений структура алгоритма станет абсолютно прозрачной, ясной и наглядной (рис. 146).

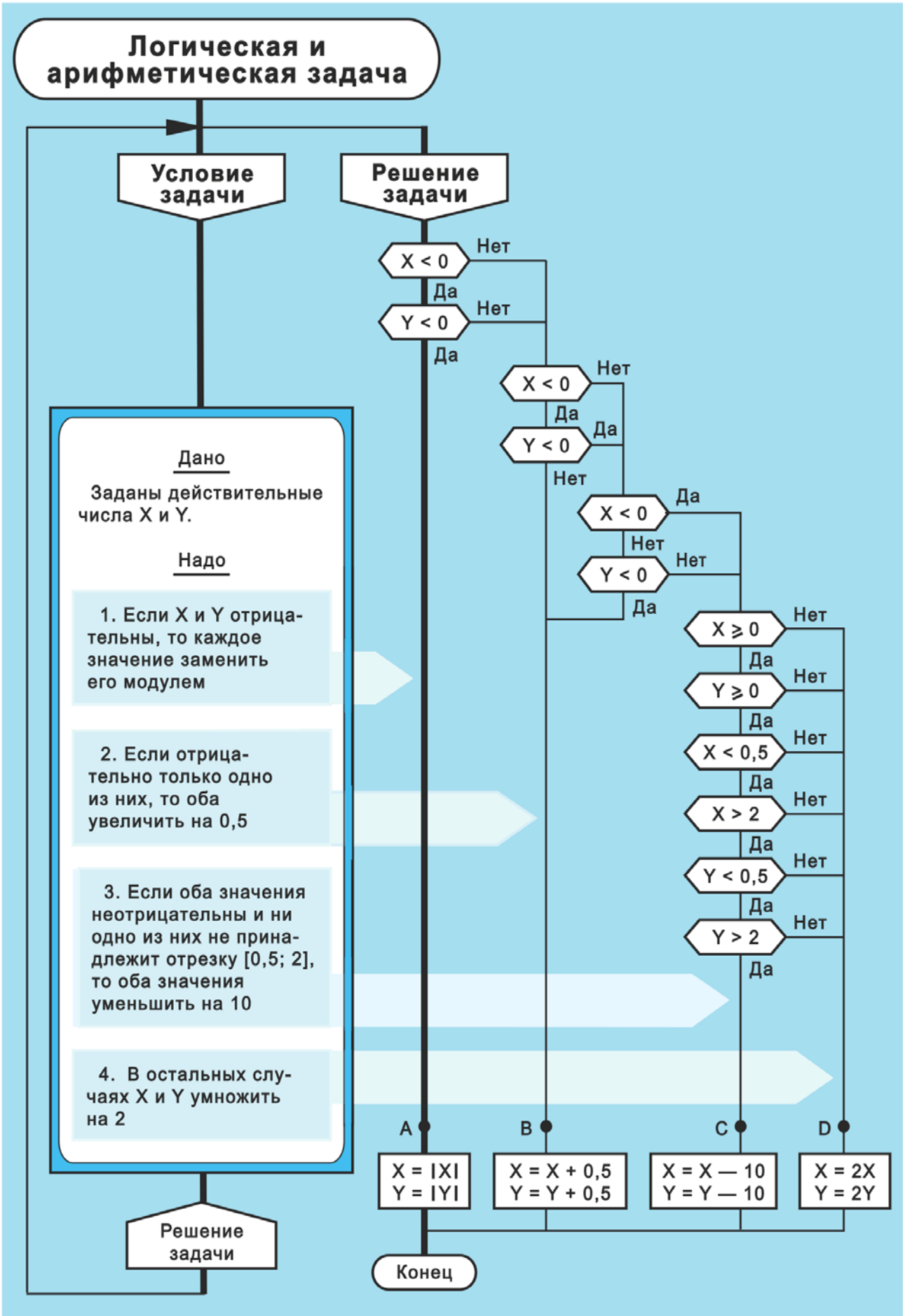


Рис. 146. Сложная логическая задача с простыми вычислениями

ЕЩЕ ОДИН ЭРГОНОМИЧНЫЙ АЛГОРИТМ

Разберем задачу на рис. 147. Это «провокационная» задача, так как она имеет не один, а шесть ответов. Чтобы выявить правильные ответы, нужно рассмотреть восемь (!) вариантов решения. И отбросить два из них как бессмысленные.

Типичная ошибка состоит в том, что учащиеся не видят эти восемь вариантов решения. И по этой причине упускают из виду один или несколько ответов.

На рис. 147 решение задачи изображено в виде чертежа (дракон-схемы). При этом используется ряд эргономических приемов, облегчающих понимание.

- Чертеж создает целостный графический образ решения, в котором иконы связаны между собой соединительными линиями.
- Зрительная сцена заставляет учащихся учесть все варианты, так как пропущенный вариант порождает оборванную линию. Последняя сразу же будет опознана как ошибка и исправлена.
- Схема упорядочена по вертикали таким образом, что бегунок, двигаясь по линиям сверху вниз, перемещается от условия задачи к одному из ответов. Значит, зрительная сцена организована не хаотично, а по правилу «Вверху — *условие задачи,* внизу — *ответы*».
- Схема упорядочена и по горизонтали. На одной горизонтали расположены однородные иконы, внутри которых находятся близкие по смыслу текстовые надписи. Это дисциплинирует чертеж, делает его зрительно предсказуемым, облегчает чтение и понимание.
- Чтобы облегчить зрительное различение хороших и плохих вариантов, нижняя горизонталь (горизонталь ответов) расщеплена на два уровня, слегка смещенных друг относительно друга.
- На нижнем уровне размещены шесть хороших вариантов, содержащих правильные ответы. Чуть выше находятся два плохих варианта, которые не содержат ответов.

Перечисленные эргономические приемы помогают читателю упорядочить свои мысли, привести их в стройную систему. В итоге сложная задача упрощается, становится наглядной и доходчивой.

Мы убедились, что с помощью языка ДРАКОН можно упростить алгебраическое выражение.

ПОДСКАЗКА

Чему равен корень из y^2? Двоечник ответит: $\sqrt{y^2} = y$. Это неполный и, следовательно, некорректный ответ.

Правильный ответ состоит из двух частей:

- $\sqrt{y^2} = y,$ если $y \geq 0.$
- $\sqrt{y^2} = -y,$ если $y < 0.$

Мы показали, что корень из y^2 имеет два ответа. Запомним это и вернемся к рисунку 147. В этой задаче имеется не один, а три корня из квадрата, а именно: $\sqrt{a^2}$, $\sqrt{b^2}$, $\sqrt{(a-b)^2}$. Следовательно, общее число вариантов (которые нужно рассмотреть) равно 2^3= 8. Мы так и сделали, и получили восемь ответов. При этом выяснилось, что два ответа из восьми негодные, так как условия содержат противоречие. Отбросив их, получим, шесть вариантов ответа, которые показаны на рис. 147 в нижнем ряду.

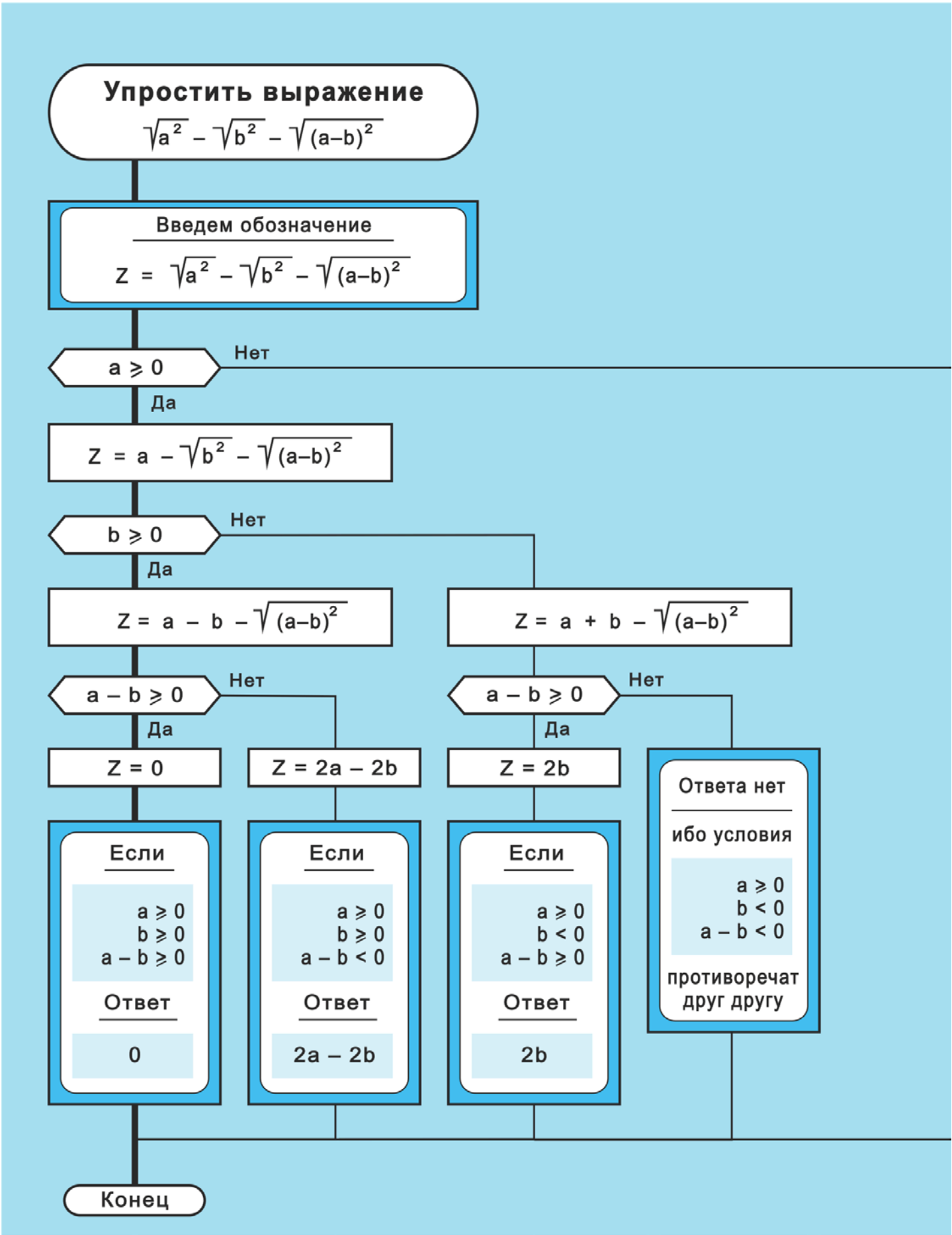
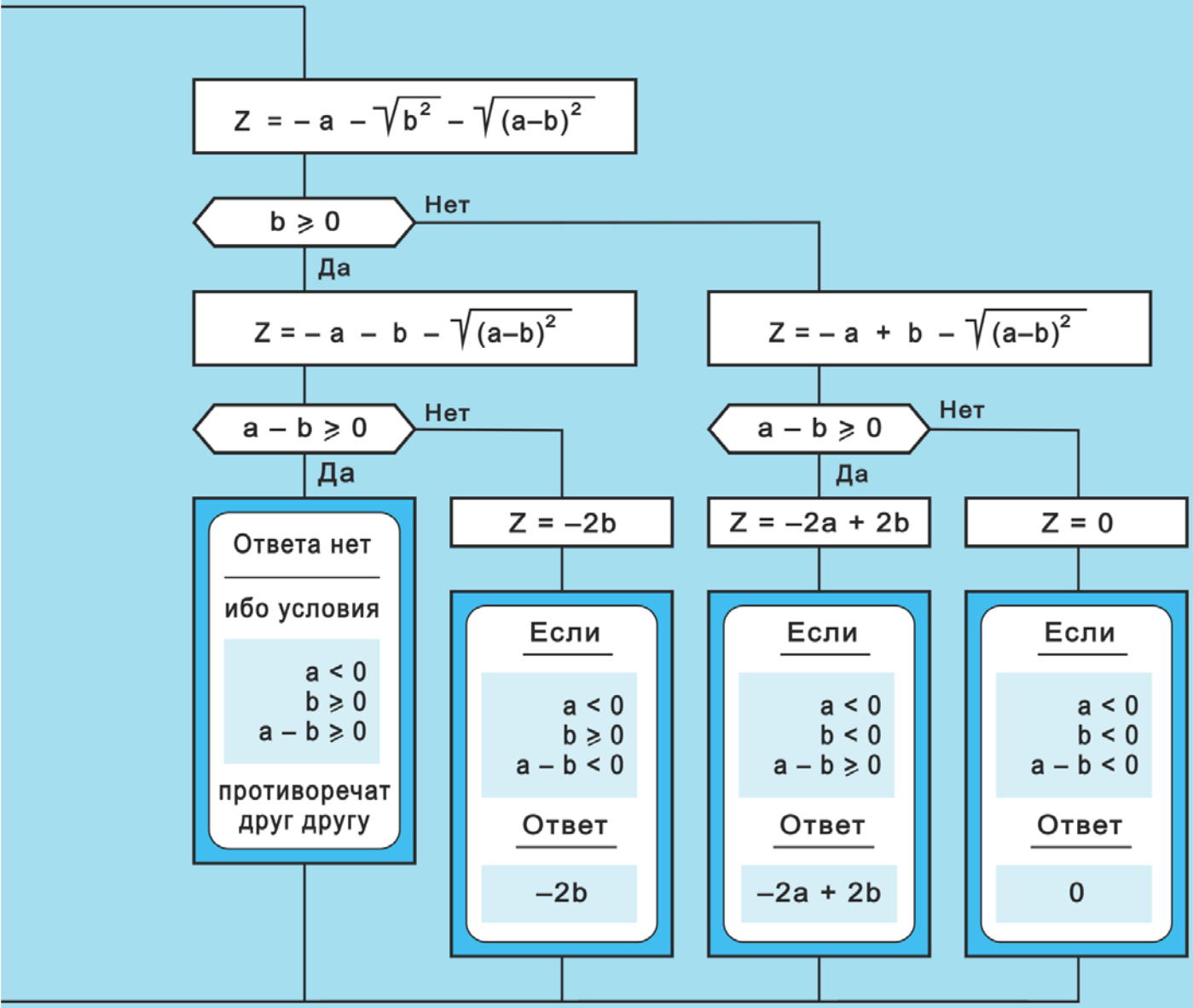


Рис. 147. Как упростить алгебраическое выражение

Знаете ли вы, что

... с помощью этой простенькой задачки два коварных экзаменатора «зарезали» сорок тысяч абитуриентов



ДРАКОН ПОМОГАЕТ ИЗУЧАТЬ ХИМИЮ

На рис. 148 показана схема простого химического опыта — определение кислотности раствора. Если лакмусовая бумажка, погруженная в раствор, краснеет, значит, раствор кислый. Если синеет — щелочной. Если цвет бумажки не изменился и остался фиолетовым — раствор нейтральный.

РАСПОЗНАВАНИЕ НЕИЗВЕСТНОГО ХИМИЧЕСКОГО ВЕЩЕСТВА

Рассмотрим сложный химический опыт (рис. 149). Учитель берет одно из шести химических удобрений и помещает его в колбу. Что в колбе — неизвестно. Это может быть аммиачная селитра, натриевая соль, сульфат аммония, суперфосфат, сильвинит или калийная соль.

Нужно выполнить эксперимент, позволяющий узнать, какое вещество находится в колбе.

Опыт делают в два этапа. Сначала идут подготовительные действия:

- Положить удобрение в три сосуда.
- В первый сосуд добавить серную кислоту H_2SO_4.
- Во второй сосуд добавить раствор хлорида бария BaCl.
- В третий сосуд добавить щелочь.

Указанные действия описаны на рис. 149 в первой ветке силуэта, которая называется «Проведение химических реакций».

ОПРЕДЕЛЕНИЕ НАЗВАНИЯ УДОБРЕНИЯ

После этого анализируются результаты и определяется неизвестное вещество на основании таблицы 1.

Таблица 1

Название вещества	Внешний вид вещества	Что мы видим или ощущаем, если в данное вещество добавить:		
		H_2SO_4	BaCl	Щелочь
Аммиачная селитра	Белые кристаллы	Бурый газ	—	Запах аммиака
Натриевая селитра	Бесцветные кристаллы	Бурый газ	Помут-нение	—
Сульфат аммония	Светло-серые кристаллы	—	Белый осадок	Запах аммиака
Супер-фосфат	Светло-серый порошок	—	Белый осадок	—
Сильвинит	Розовые кристаллы	—	—	—
Калийная соль	Бесцветные кристаллы	—	—	—

Будем считать, что дракон-схема на рис. 149 размещена на сенсорном экране, снабжена анимацией и работает в диалоговом режиме. Бегунок (рабочая точка процесса) движется от иконы Заголовок» к иконе Конец. По мере движения иконы и соединительные линии вспыхивают и горят на экране более ярким светом, выделяя пройденную часть пути.

Когда процесс дошел до иконы «При взаимодействии с H_2SO_4 выделяется бурый газ?», данная икона начинает мерцать, привлекая к себе внимание и требуя ответа. Реагируя на это событие, ученик касается рукой ответа: «Да» или «Нет».

Икона перестает мерцать и (при ответе «Да») загорается путь, ведущий к иконе «При взаимодействии со щелочью ощущается запах аммиака?», которая начинает мерцать. Далее события повторяются, пока на экране не загорится искомое название удобрения.

С точки зрения процесса познания, проблема распознавания удобрения состоит из трех частей:

- постановка задачи;
- описание действий с исследуемым веществом и реактивами;
- логический анализ результатов опыта.

В первой ветке даны постановка задачи (икона Комментарий) и описание последовательности ручных манипуляций (четыре иконы Действие). Во второй ветке демонстрируется логический анализ и получение ответа.

Важно, что все три части проблемы предъявляются зрителю в одном визуальном поле. Благодаря этому обеспечивается удобство восприятия и облегчается процесс познания.

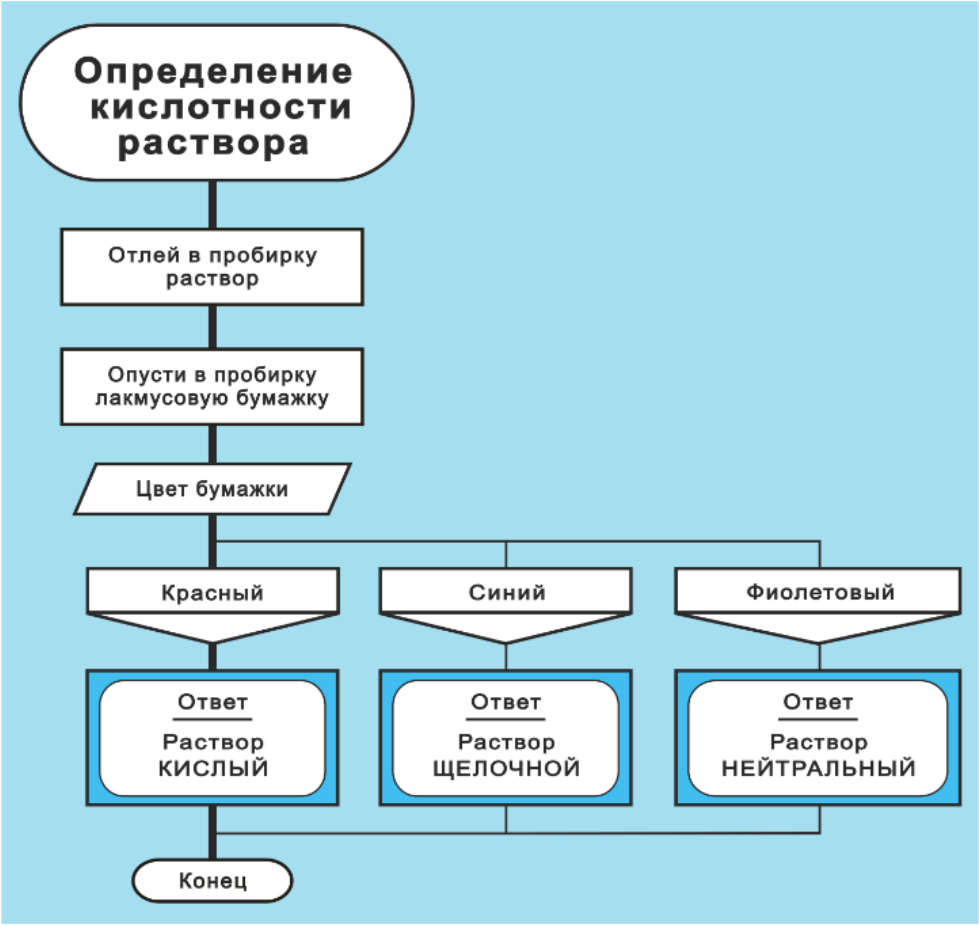


Рис. 148. Как определить кислотность раствора

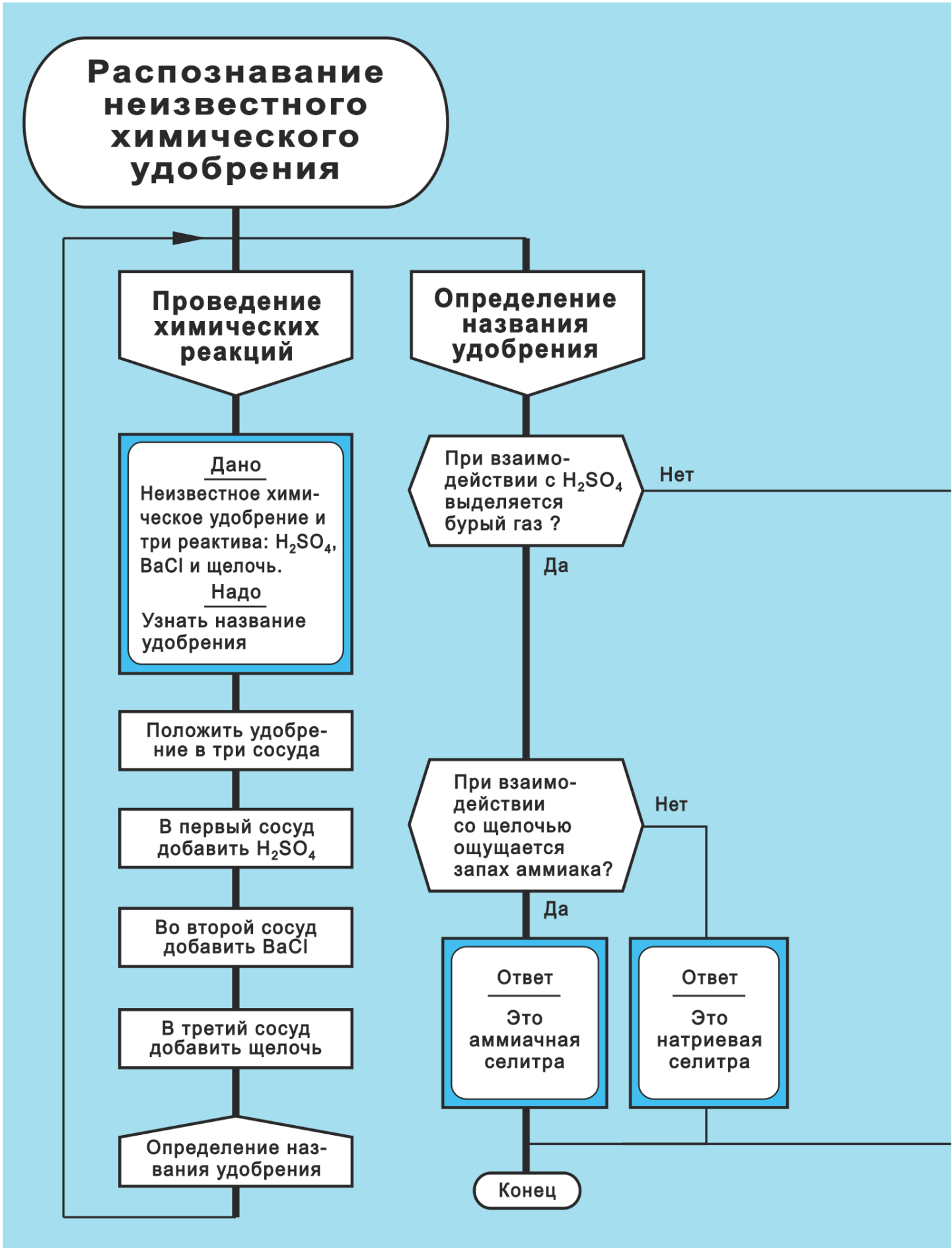
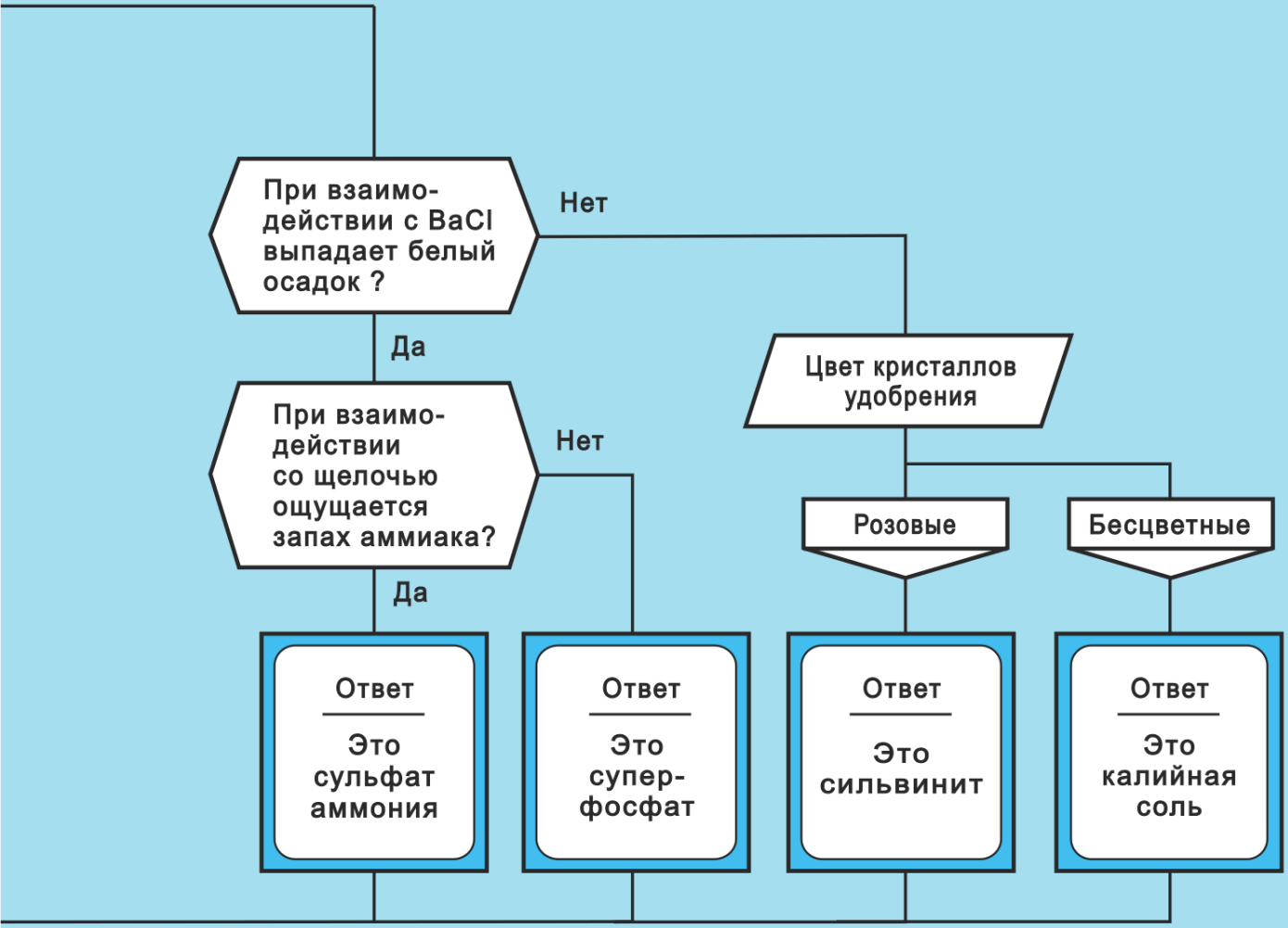


Рис. 149. Распознавание неизвестного химического удобрения

Язык ДРАКОН позволяет описывать:
1) физические действия (1-я ветка),
2) информационные действия (2-я ветка)



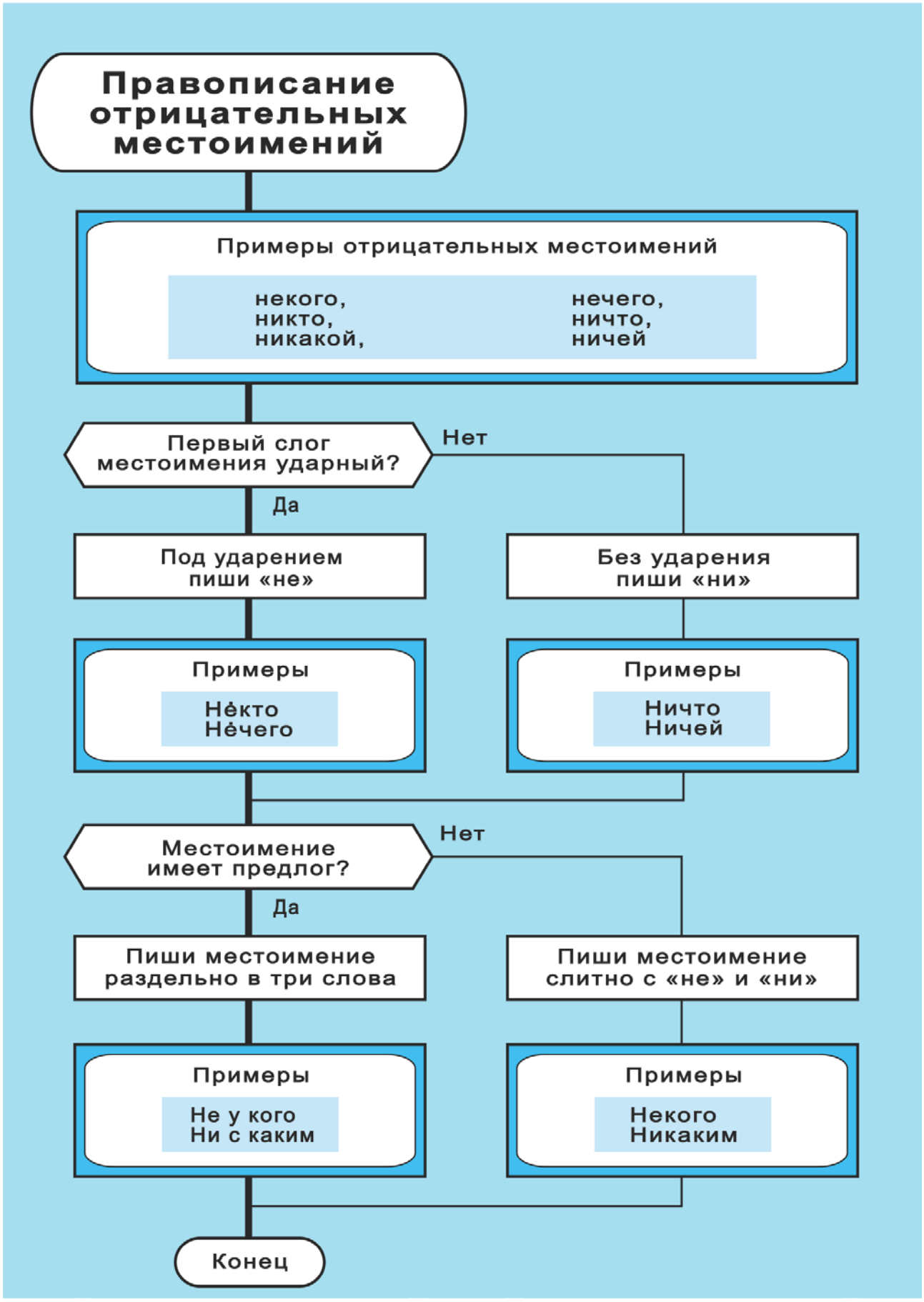


Рис. 150. Правила написания отрицательных местоимений

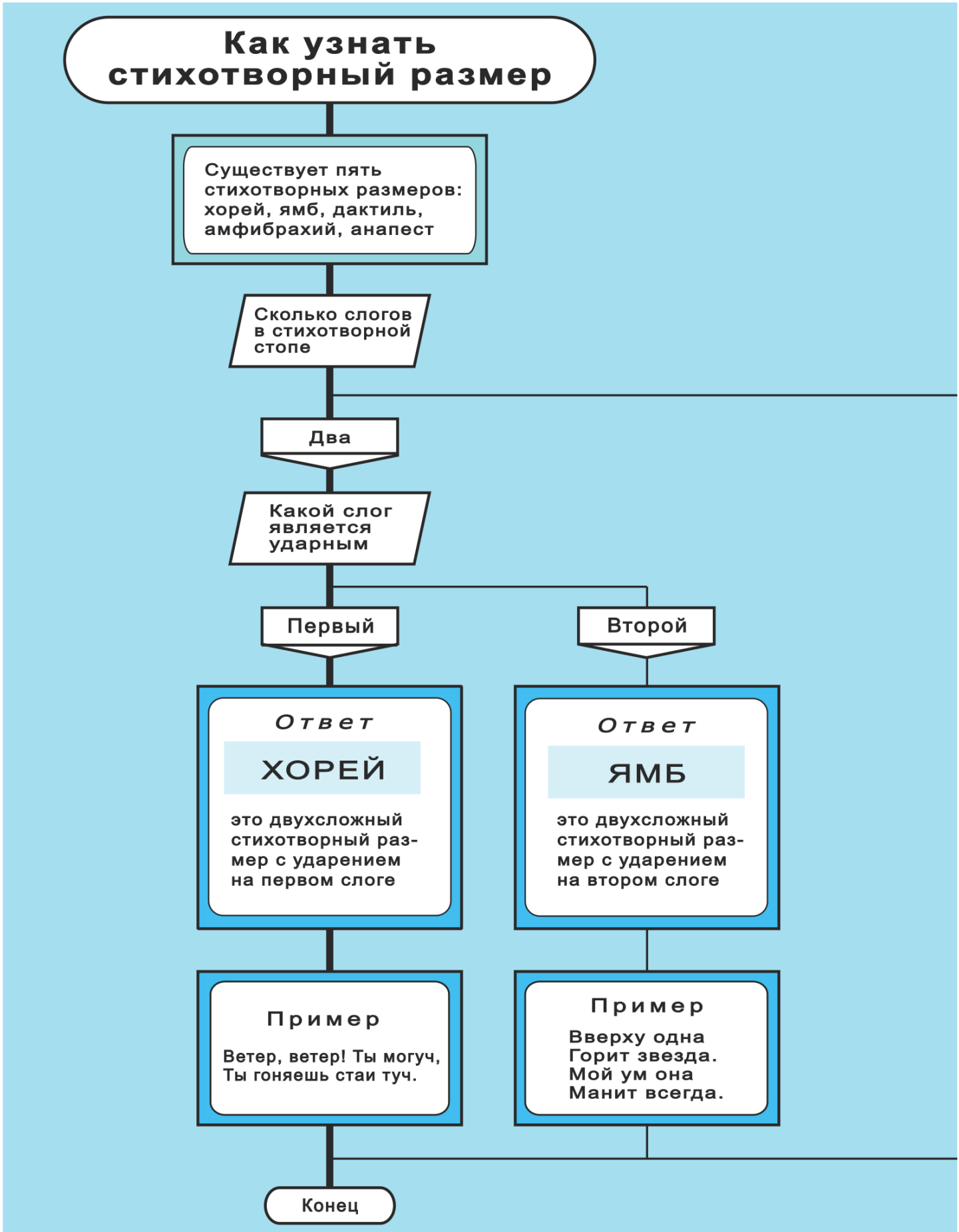


Рис. 151. Изучаем правила поэзии. Как определить стихотворный размер

Язык ДРАКОН позволяет описывать некоторые правила анализа произведений искусства, например стихотворений

ДРАКОН ПОМОГАЕТ ИЗУЧАТЬ ГУМАНИТАРНЫЕ ПРЕДМЕТЫ

Приведем еще пару примеров, подтверждающих достоинства языка ДРАКОН. Эти примеры демонстрируют возможность его применения в различных сферах человеческой деятельности.

Рисунок 150 иллюстрирует использование языка для изображения грамматических правил. На рис. 151 показан пример формализации простейших правил анализа стихотворений.

ВЫВОДЫ

1. Учась в школе, ученик получает знания из различных областей, или, как говорят, он изучает несколько предметов.
2. Между этими предметами существуют связи, но ученик, как правило, их не замечает.
3. Алгоритмы и дракон-схемы представляют собой удобное средство, облегчающее для школьника выявление и уяснение сути межпредметных связей.

Часть 6

АЛГОРИТМЫ И ЖИЗНЕРИТМЫ

Глава 27

ЧЕМ РАЗЛИЧАЮТСЯ АЛГОРИТМЫ И ЖИЗНЕРИТМЫ

ДВА ТЕРМИНА

Алгоритм — символ безукоризненной точности и строгой дисциплины. К сожалению, в научной и учебной литературе термин «алгоритм» зачастую используют для обозначения двух разных понятий. Это плохо. Подобная двусмысленность может привести к недоразумениям и неприятным последствиям.

Чтобы устранить неоднозначность, полезно различать два термина:

- алгоритм (для строгих алгоритмов),
- жизнеритм (для нестрогих).

Наша задача — разъяснить оба термина и устранить путаницу.

ЧЕМ РАЗЛИЧАЮТСЯ АЛГОРИТМ И ЖИЗНЕРИТМ

Ответ дан на рисунке 152. На шампуре видим вопрос: «Компьютер может решить задачу»? Если да, значит, метод решения является ***алгоритмом***.

Если нет, читаем второй вопрос: «Человек может решить задачу»? Предположим, нужно сходить в аптеку и купить лекарство. Компьютер такую задачу решить не может (у него нет ног), а человек выполнит ее без труда. Отсюда следует, что данный метод — это ***жизнеритм***.

На рис. 152 представлены оба варианта решения задачи. Там же показан третий вариант, где задача не имеет решения.

ЧТО СКАЗАНО В АВТОРИТЕТНОМ УЧЕБНИКЕ

Профессор А. Н. Степанов в «Курсе информатики для студентов информационно-математических специальностей» отмечает:

> «В 30–50-х годах XX века в связи с формированием теории алгоритмов кардинально изменилось понимание их роли и значения в научной сфере и бытовой жизни человека. В это время термин «алгоритм» приобрел четкое научное определение. Одновременно выяснилось, что огромное большинство своих действий в своей жизни и профессиональной деятельности: *умывание, приготовление различных блюд, переход улицы, выпечку хлеба, выплавку стали, выращивание винограда и т. д. — человек осуществляет по вполне определенным алгоритмам*» [15, p. 164].

Стоп! В это рассуждение вкралась неточность. Умывание — это не алгоритм, а жизнеритм. Точно так же приготовление блюд, переход улицы, выпечка хлеба, выращивание винограда — не алгоритмы, а жизнеритмы. Автор учебника, увы, не проводит различия между этими понятиями.

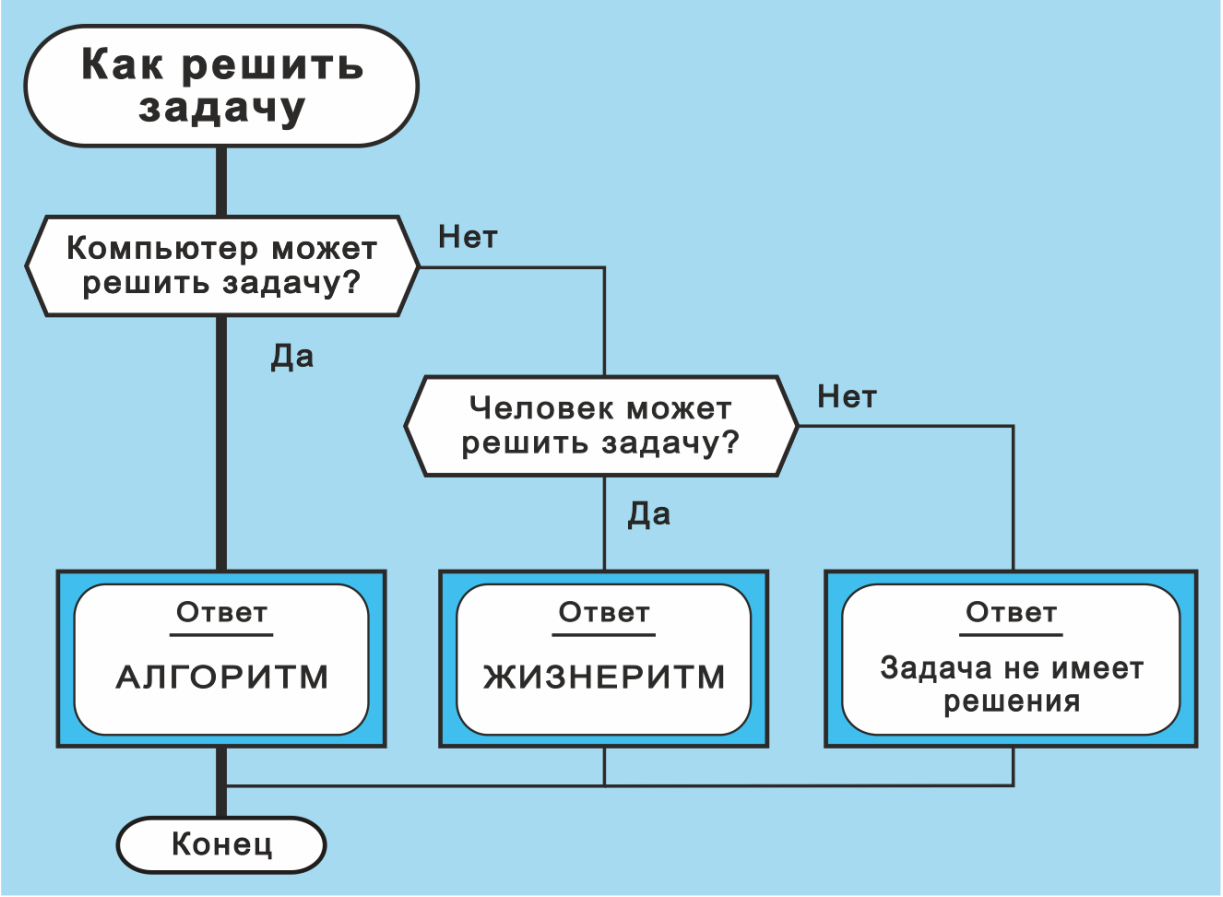


Рис. 152. Алгоритм «Как решить задачу»

ИНТУИТИВНОЕ ПОНЯТИЕ АЛГОРИТМА

Интуитивное понятие алгоритма обычно разъясняют следующим образом:

> Алгоритм — это «понятное и точное предписание (указание) исполнителю совершить последовательность действий, направленных на достижение указанной цели или на решение поставленной задачи» [16].
>
> «Алгоритм представляет собой совокупность правил, инструкций для исполнителя, выполняя которые, он за конечное число шагов добьется искомого результата» [15, p. 166].
>
> «Алгоритм представляет собой точный набор инструкций, описывающих порядок действий для достижения цели, получения решения задачи за конечное время» [15, p. 167].

Интуитивное понятие алгоритма (в такой формулировке) в равной степени относится как к алгоритмам, так и к жизнеритмам. Оно не позволяет провести разграничительную линию между ними.

РАЗЛИЧИЯ В ТРАКТОВКЕ ПОНЯТИЯ «АЛГОРИТМ» В ИНФОРМАТИКЕ И МЕДИЦИНЕ

В информатике (computer science) *алгоритм* — формальное понятие, заданное вычислительными моделями Тьюринга, Поста, Маркова, Колмогорова, RAM и др., образующее научный фундамент автоматической обработки данных на компьютерах. Хотя алгоритм можно выполнить и вручную (например, при сложении «столбиком» и

делении «уголком»), однако это не столь важно. Научная ценность понятия «алгоритм» состоит в доказательстве возможности *автоматической* обработки информации. Огромное значение для развития цивилизации имеет именно *автоматическое*, а отнюдь не ручное исполнение алгоритмов. Понятие алгоритма открыло дорогу для создания сотен миллионов сказочно быстрых компьютеров, которые усеяли земной шар и создали условия для улучшения жизни человечества.

Перейдем к медицинским алгоритмам. Что такое алгоритм в медицинской медицине? Это пошаговое описание действий врачей при решении лечебно-диагностических задач и вопросов врачебной практики. Автоматика здесь тоже применяется (например, автоматический дефибриллятор при реанимации), но ведущую роль в медицинских алгоритмах безусловно играет врач, а не автоматические устройства. Хотя роль последних медленно, но неуклонно возрастает.

Можно показать, что медицинский алгоритм — фундаментальное понятие для здравоохранения, он является способом фиксации и передачи знаний, добытых медицинской наукой [17, pp. 64, 289]. Вместе с тем у этого понятия есть недостаток, своего рода, «первородный грех»: медицинский алгоритм… не является алгоритмом.

Жизнеритм. В самом деле, с точки зрения информатики, *медицинский алгоритм* — это не алгоритм, а всего лишь жизнеритм. Последний имеет внешнюю форму алгоритма, но содержит не до конца определенные шаги.

Медицинский алгоритм внешне похож на алгоритм, но сходство обманчиво, ибо налицо принципиальный дефект. Шаги медицинского алгоритма могут быть не полностью определены или даже опущены. Поэтому медицинский алгоритм не пригоден для исполнения на компьютере, он выполняется человеком (врачом).

Настоящий алгоритм содержит в себе все указания, необходимые для его выполнения. Клинический алгоритм, напротив, содержит лишь часть указаний; недостающие сведения обязаны восполнить врачи, выполняющие алгоритм, опираясь на свои профессиональные знания.

Чем же отличается медицинский алгоритм от настоящего алгоритма? Отличия имеют фундаментальный характер.

Недостаток медицинского алгоритма – отсутствие свойства определенности (детерминированности), которое является обязательным для алгоритмов. На практике это часто приводит к отсутствию необходимой точности, приблизительности, пропуску и потере важных подробностей и условий. Все это вносит неоправданные трудности в мышление врачей и служит одной из причин врачебных ошибок.

ПРИМЕР МЕДИЦИНСКОГО АЛГОРИТМА

Петрович колол дрова, но поскользнулся и нечаянно отрубил себе палец. Что делать? Как спасти отрезанный палец (ампутат), который валяется на земле?

Ответ дает медицинский алгоритм на рис. 153. Прочитайте его. На каждом шаге выполняется одно медицинское действие. Действия выполняются последовательно, друг за другом и решают поставленную задачу.

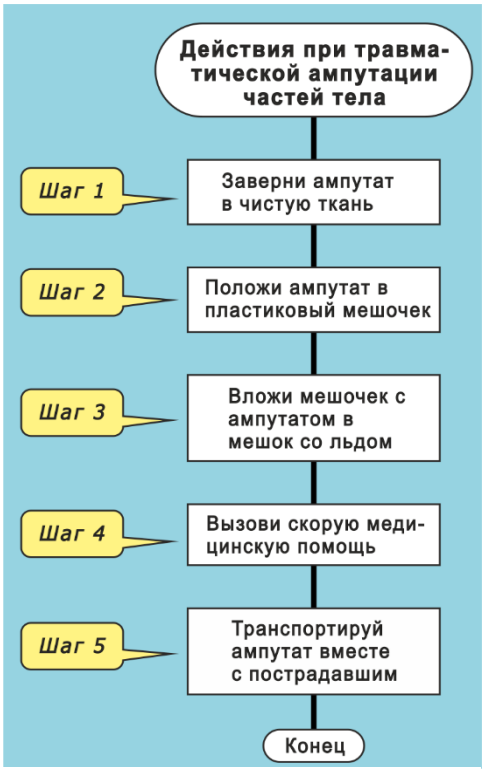


Рис. 153. Медицинский алгоритм «Действия при травматической ампутации частей тела»

Кто является исполнителем алгоритма? Кто будет выполнять команды? Компьютер? Вовсе нет. Данная графическая инструкция предназначена не для компьютера, а для человека. Выполнять ее будет врач или парамедик, т. е. человек, умеющий оказывать помощь и знающий, как это делать.

Из этого рассказа проистекает вывод. Компьютер не может выполнить алгоритм на рис. 153. Это может сделать только человек. Следовательно, на рис. 153 изображен не алгоритм, а жизнеритм.

НЕДОСТАТОК ПОНЯТИЯ «АЛГОРИТМ»

Мы показали, что термин «алгоритм» в научной и прикладной литературе используется неоднозначно. Необходимо различать:

- строгие алгоритмы, которые можно запрограммировать и автоматически исполнить на компьютере;
- нестрогие алгоритмы, выполнить которые способен лишь человек. А компьютер для этого не пригоден.

Читатель должен знать, что сегодня даже в очень хороших книгах имеет место неряшливость терминологии, когда разным вещам дают одинаковое имя. Подобная неаккуратность получила широкое распространение. В литературе термин «алгоритм» зачастую используют для жизнеритмов: кулинарных и иных рецептов, для различных инструкций и даже для простейших предписаний типа:

- Уходя, гасите свет.
- Идти слева, стоять справа [18].

Алгоритм про попугаев, описанный в начале книги, также является жизнеритмом.

АЛГОРИТМЫ И ЖИЗНЕРИТМЫ

Настоящий алгоритм можно воспроизводить (повторять) сколько угодно раз с абсолютной точностью и без малейших отклонений. Возьмем для примера алгоритм сложения столбиком.

$$\begin{array}{r} 37 \\ +\ 21 \\ \hline 58 \end{array}$$

Выучив правила сложения в начальной школе, человек приобретает умение складывать числа с нужной точностью. Интересно, почему?

Потому что алгоритм сложения содержит в себе все указания, необходимые для его выполнения. Такой алгоритм может выполнять кто угодно: человек, электронный робот, калькулятор или компьютер.

Алгоритмы второго типа (кулинарные рецепты, клинические алгоритмы, бизнес-процессы) устроены иначе. Они содержат в себе лишь часть указаний, а остальные пропускают, рассчитывая на знания и догадливость живого человека.

Поэтому *алгоритм ведения пациента при нормотензивной глаукоме* нельзя возложить на робота. Для этого непременно нужен врач, профессиональный офтальмолог.

Еще пример. Алгоритм хирургической операции на открытом сердце по замене митрального и аортального клапанов также нельзя поручить роботу. Нужны врачи, бригада кардиохирургов.

Чтобы назвать не полностью определенный алгоритм, выше предложен термин «жизнеритм» (liferithm).

АЛГОРИТМИЧЕСКОЕ ПРЕДПИСАНИЕ И ЖИЗНЕРИТМ

Математик Н. Н. Непейвода сформулировал важный для нас тезис:

> «Необходимо различать алгоритм и алгоритмическое предписание, имеющее внешнюю форму алгоритма, но включающее не до конца определенные шаги» [95].

На эту тему существует литература, восходящая к пионерской работе Льва Ланды «Алгоритмизация в обучении» [19]. Ланда первым выявил различие между алгоритмом и алгоритмическим предписанием, но мы предпочитаем более точную формулировку Непейводы[2].

Тезис Ланды-Непейводы создает научный фундамент для четкого разграничения алгоритмов и алгоритмических предписаний.

Легко заметить, что жизнеритм и алгоритмическое предписание — одно и то же. Интересующий нас тезис Ланды-Непейводы для жизнеритмов приобретает вид:

> Необходимо различать алгоритм и жизнеритм, имеющий внешнюю форму алгоритма, но включающий не до конца определенные шаги.

ФОРМАЛИЗАЦИЯ АЛГОРИТМИЧЕСКИХ ПРЕДПИСАНИЙ И ЯЗЫК ДРАКОН

Вернемся к понятию «алгоритмическое предписание». В настоящее время оно нередко плохо определено и имеет расплывчатый характер. Ниже приводится ряд примеров, характеризующих разнобой мнений специалистов и неблагополучное положение в этой области.

> «Алгоритмические предписания представляют собой указания учащимся последовательности выполнения операций познавательной деятельности… алгоритмическое предписание – это конструктивная модель необходимой деятельности» [20].

> «Алгоритмическое предписание указывает, ЧТО надо сделать, а вот КАК делать — обучаемый решает сам» [21].

> «Алгоритмическое предписание допускает большую свободу в характере использования его студентами, а действия, которыми описываются, являются вариативными и многозначными» [22].

> «Алгоритмическое предписание…– система указаний, определяющая лишь общие направления поиска плана решения задачи (задания) и оставляющая большие возможности для самостоятельного решения ряда вопросов» [23].

> «Под алгоритмическим предписанием понимается общепонятное предписание о выполнении в определенной последовательности элементарных операций для решения любой из задач, принадлежащих к

---

[2] Приведем исходную формулировку Льва Ланды: «В отличие от алгоритмов в строгом математическом смысле, алгоритмические предписания допускают правила, которые обращены не только к формальным, но и к содержательным операциям… Понятие предписания алгоритмического типа является менее точным (в математическом смысле), чем понятие алгоритма» [65].

> определенному типу или классу... Алгоритмическое предписание должно обладать всеми свойствами математического алгоритма, т.е. определенностью (детерминированностью), массовостью и результативностью» [24].

Приведенные суждения свидетельствуют о том, что понятие «алгоритмическое предписание» является некорректным, неудовлетворительным. Оно нуждается в уточнении на основе разъяснения математика Н.Н. Непейводы и тезиса Ланды-Непейводы.

Выше мы предложили заменить громоздкое и сомнительное выражение «алгоритмическое предписание» на четко определенное понятие «жизнеритм» (liferithm). Речь идет, разумеется, не о замене слов, а о формализации понятия «алгоритмическое предписание».

Предлагаемые изменения направлены на кардинальное повышение качества алгоритмических предписаний:

- существующие алгоритмические предписания должны претерпеть значительные изменения и превратиться в эргономичные жизнеритмы высокой точности;
- следует осуществить стандартизацию записи алгоритмических предписаний. Для этого они должны быть изображены на языке ДРАКОН и соответствовать правилам языка ДРАКОН.

Термин «жизнеритм» рекомендуется использовать только для тех алгоритмических предписаний, которые записаны на языке ДРАКОН в точном соответствии с правилами этого языка.

ЧТО ТАКОЕ ОПРЕДЕЛЕННОСТЬ АЛГОРИТМА

Определенность говорит о том, что каждое указание алгоритма должно быть точным, четким, однозначным и не оставлять места для случайных, произвольных толкований и действий. Как говорят специалисты, «неоднозначность толкования записи алгоритма недопустима» [25].

В тезисе Непейводы есть фраза «не до конца определенные шаги». Что это значит?

Смысл в том, что в *жизнеритме* (читай — в ущербном, неполноценном алгоритме) отсутствует надлежащий порядок. Некоторые правила описания и выполнения шагов алгоритма или не указаны, или неизвестны, или недоступны.

Повторим еще раз: в жизнеритме (в отличие от алгоритма) нет определенности, нет необходимой точности, нет однозначности. Поэтому компьютер не может его выполнить. Более того, некоторые читатели жизнеритма испытывают трудности и могут понять его смысл неправильно, неоднозначно и даже превратно.

ЧТО ТАКОЕ ВЫСОКАЯ ТОЧНОСТЬ

Итак, жизнеритм не обладает свойством определенности (однозначности), которое является обязательным для алгоритмов. Поэтому появляется задача: повысить качество жизнеритмов и в максимальной степени приблизить их свойства к образцу — к свойствам алгоритмов.

Эта цель достигается с помощью языка ДРАКОН. Он позволяет осуществить приближение к идеалу, то есть значительно повысить определенность (точность) жизнеритмов. Благодаря ДРАКОНу некачественные жизнеритмы превращаются в жизнеритмы высокой точности.

Оговоримся. Здесь есть принципиальное ограничение. Выше головы не прыгнешь. Жизнеритмы содержат ***не до конца определенные шаги.*** Исправить этот дефект невозможно.

Это значит, что жизнеритмы никогда не смогут превратиться в алгоритмы. Однако с помощью ДРАКОНа их можно улучшить, облагородить и сделать пригодными для решения сложных практических задач и предотвращения ошибок. В таком случае говорят, что жизнеритмы обрели высокую точность.

ВЫВОДЫ

1. Необходимо различать алгоритмы и жизнеритмы.
2. Все шаги алгоритма строго определены.
3. У жизнеритма некоторые шаги не полностью определены.
4. Алгоритм можно запрограммировать и автоматически исполнить на компьютере.
5. Жизнеритм нельзя запрограммировать и исполнить на компьютере.
6. Жизнеритм может исполнить только человек.
7. Вместо громоздкого и устаревшего термина «алгоритмическое предписание» рекомендуется использовать краткий термин «жизнеритм».
8. Жизнеритм не обладает свойством определенности, которое является обязательным для алгоритмов.
9. Создать жизнеритмы высокой точности — значит приблизить их свойства к свойствам алгоритмов.
10. Язык ДРАКОН позволяет превратить некачественные жизнеритмы в жизнеритмы высокой точности.

Глава 28

КЛИНИЧЕСКИЕ АЛГОРИТМЫ: АКТУАЛЬНАЯ, НО НЕРЕШЕННАЯ ПРОБЛЕМА

ВВЕДЕНИЕ

В истории алгоритмов можно выделить два этапа. На первом этапе алгоритмами стали пользоваться математики и программисты. На втором этапе, который начался сравнительно недавно, алгоритмы проникли в медицину и начали превращаться в рабочие инструменты врачей.

Второй этап застал многих врасплох. Система среднего и высшего образования, по-прежнему нацелена на подготовку математиков и программистов и слабо учитывает потребности врачей и среднего медперсонала, которым тоже нужны качественные алгоритмы. Дело осложняется двумя причинами:

- медицинские алгоритмы имеют специфику, требующую значительной перестройки системы здравоохранения;
- у большинства медиков отсутствует алгоритмическое мышление.

АЛГОРИТМИЧЕСКОЕ МЫШЛЕНИЕ

Вспомним историю. В 1980 году Сеймур Пейперт ввел понятие «вычислительное мышление» (computational thinking) [26]. Со временем от него отпочковалось понятие *алгоритмическое мышление* (algorithmic thinking), которое получило широкое распространение.

Преподаватели и учителя информатики заботятся о том, чтобы развить у школьников и студентов алгоритмическое мышление. С какой целью? Ответ известен — чтобы познакомить учащихся с алгоритмами и программированием. Существует устойчивое мнение: алгоритмы и алгоритмическое мышление нужны, в первую очередь, *программистам и математикам* (а отнюдь не врачам).

А.И. Газейкина пишет: «Для успешного обучения студентов программированию необходимо развивать у них… алгоритмический стиль мышления» [27]. Дж. Фучик полагает, что алгоритмическое мышление проявляется при создании новых алгоритмов, а оптимальной деятельностью для его развития являются занятия программированием [28].

Впрочем, есть и иное, более взвешенное суждение, отражающее более широкий взгляд на вещи:

> «Алгоритмический стиль мышления… позволяет решать задачи, возникающие в любой сфере человеческой деятельности, и не только в программировании или математике. Алгоритмическое мышление не обязательно связано только с вычислительной техникой» [29].

Последняя точка зрения не получила активной поддержки и осталась всего лишь благим пожеланием. Все учебники, посвященные алгоритмам, все методики развития

алгоритмического мышления, все педагогические и финансовые ресурсы направлены почти исключительно на подготовку кадров программистов, математиков и смежных специальностей. Алгоритмическое мышление у практикующих врачей по-прежнему не развито.

Подобная недальновидность привела к негативным последствиям. Наиболее остро данная проблема проявляется в медицине, где она приобретает неожиданную и драматическую форму. Форму врачебных ошибок, которые приводят к неоправданным и жестоким страданиям пациентов.

АХИЛЛЕСОВА ПЯТА МЕДИЦИНЫ

В медицине существуют подробные правила диагностики и лечения больных. На жаргоне медиков такие правила называются *клиническими алгоритмами*. Хотя фактически они являются не алгоритмами, а всего лишь *жизнеритмами.*

Проблема в том, что в подобных «алгоритмах» нет необходимой точности, нет однозначности. Поэтому студенты-медики и врачи нередко испытывают значительные трудности при изучении медицинских учебников и медицинской литературы. Они могут понять смысл клинического алгоритма неполностью, неправильно, неоднозначно, с искажениями, что чревато ошибками.

Это отнюдь не пустые слова.

Неопределенность в медицинской литературе может привести к непониманию и является глубинным источником многих врачебных ошибок. Неопределенность — ахиллесова пята медицинских публикаций, медицинских учебников, клинических рекомендаций, руководств и медицинских стандартов.

ЦЕЛЬ

Цель состоит в том, чтобы кардинально повысить качество клинических алгоритмов (жизнеритмов), уменьшить неопределенность, облегчить процесс мышления врачей, снизить вероятность врачебных ошибок и, в конечном итоге, обеспечить безопасность пациентов.

Повысить качество медицинских алгоритмов, создать клинические алгоритмы *высокой точности* — значит (по возможности) приблизить их свойства к идеалу – то есть к свойствам алгоритмов. Это позволит предотвратить врачебные ошибки и спасти многие жизни.

БОЛЕВАЯ ПРОБЛЕМА СОВРЕМЕННОЙ МЕДИЦИНЫ

Некоторые врачи настороженно и скептически относятся к алгоритмам; на их знамени написано: «Медицина относится к разряду наук неточных, железных алгоритмов в ней нет и быть не может» [30]. К счастью, подобные предрассудки начинают постепенно отмирать:

> «На протяжении многих лет медицину относили к неточным наукам. И это мнение во многом поддерживалось самими врачами, оправдывающими тем самым субъективные и объективные проблемы лечения пациентов. Представление о неточности медицины весьма серьезно пошатнулось в последние десятилетия» [31, p. 5].

Пагубную роль играет алгоритмическая неосведомленность врачей, которая серьезно тормозит алгоритмизацию медицины. Большинство клинических алгоритмов представлено в мировой медицинской литературе в виде текста на естественном

языке. Это недопустимо, потому что последний не пригоден для записи *безошибочных* алгоритмов. Применение профессионального медицинского языка для записи алгоритмов является некорректным (безграмотным) и подлежит исправлению.

Часть алгоритмов изображается графически, но обычно графическое представление алгоритмов выполняется неумело: страдает неполнотой и имеет многочисленные нарушения алгоритмических и эргономических правил. Например, в авторитетном руководстве по хирургии [32] показаны сокращенные графические алгоритмы (деревья решений), которые нельзя назвать алгоритмами, так как многие важные действия врачей (шаги алгоритма) пропущены, не упомянуты и не описаны.

Клинические алгоритмы, изображенные графически в виде блок-схем алгоритмов согласно стандартам ISO 5807:85 и ГОСТ 19.701-90, также не удовлетворительны, поскольку они не упорядочены, не эргономичны и не защищены от ошибок. При большой степени детализации блок-схемы становятся запутанными и неудобными для использования из-за существенно возрастающей громоздкости и быстрой потери наглядности [33]. То же самое можно сказать о схемах деятельности (activity diagrams) языка UML.

АЛГОРИТМИЧЕСКАЯ НЕРЯШЛИВОСТЬ И НЕКОМПЕТЕНТНОСТЬ

В целом напрашивается неутешительный вывод. Налицо серьезные системные дефекты в ныне существующей практике описания клинических алгоритмов.

Для медицинской литературы и системы медицинского образования характерны низкое качество клинических алгоритмов, что имеет далеко идущие последствия и неблагоприятно отражается на показателях здоровья населения. Доктор медицинских наук, профессор, член-корреспондент РАН Г.В. Порядин отмечает:

> «Алгоритмическая неряшливость и некомпетентность, невозможность обеспечить необходимую точность, неумение выявить при диагностике все точки разветвления алгоритма, низкая культура производства медицинских алгоритмов, систематическое нарушение правил алгоритмизации… – все это мешает делу. Подобные промахи, которые постоянно встречаются в медицинских учебниках, руководствах, клинических рекомендациях и протоколах, представляют собой болевую проблему современной медицины и имеют значимые негативные последствия. Корень всех этих недочетов состоит в том, что в мире до сих пор отсутствует единый стандарт медицинских алгоритмов» [34].

ВРАЧЕБНЫЕ ОШИБКИ И БЕЗОПАСНОСТЬ ПАЦИЕНТОВ

Врачебные ошибки опасны тем, что могут привести к смерти, стойкой инвалидности пациентов или причинить иной вред. До последнего времени статистика появления ошибок не была известна. Ситуация изменилась в 1999 году, когда был опубликован 300-страничный доклад Национальной академии медицины США (National Academy of Medicine, Institute of Medicine) под названием «Человеку свойственно ошибаться: создание более безопасной системы здравоохранения» [35].

Было установлено, что медицинские ошибки в больницах США приносят огромный вред: ежегодно они являются причиной смерти от 44000 до 98000 человек [35, pp. 1, 26, 31]. Это значит, что в американских больницах каждые полгода погибает больше американцев, чем за всю Вьетнамскую войну [36].

По вине врачей умирает больше людей, чем от дорожно-транспортных происшествий (43 458 жертв), от рака молочной железы (42 297 жертв), от СПИДа (16 516 жертв) [35, pp. 1, 26].

В докладе делается вывод, что больница гораздо опаснее самолета. Потому что риск смерти вследствие врачебной ошибки намного больше, чем риск гибели в авиационной аварии («risk of dying as a result of a medical error far surpasses the risk of dying in an airline accident») [35, p. 42].

Доклад вызвал большой общественный резонанс. По рекомендации Института медицины в конгрессе США были проведены слушания и принят закон о безопасности пациентов (Patient Safety and Quality Improvement Act of 2005), подписанный президентом Джорджем Бушем младшим 29 июля 2005 года. Таким образом, понятие безопасности пациентов приобрело не только научный, но и юридический статус. К работе активно подключилась Всемирная организация здравоохранения, используя глобальные механизмы агентства ООН и свои отделения во всех регионах мира[3].

Однако дело оказалось более сложным, а первоначальные цифры заниженными. Президент Национальной академии медицины Виктор Дзау в обобщающем докладе 2017 года [37] признал, что смертность в больницах США, которую можно предотвратить, по оценкам, превышает 200000 человек каждый год (number of preventable hospital associated deaths estimated to be over 200,000 each year in the US) [37, p. 13], что говорит о серьезности проблемы, неполном понимании глубины вопроса и явной недостаточности предпринимаемых мер и усилий.

Доклады Национальной академии медицины США впервые поставили в центр научного исследования исключительно трудную междисциплинарную проблему — проблему врачебных ошибок. В докладах предложена развернутая программа противодействия этому злу в интересах безопасности пациентов.

КРИТИКА

Указанная программа уязвима для критики. Врачебные ошибки зависят от многих причин, в том числе, от недостатков профессионального медицинского языка, который, будучи естественным языком, не приспособлен для точного и удобного описания клинических алгоритмов и не имеет необходимых для этого специальных средств.

Алгоритмы профилактики, диагностики, лечения, скорой помощи, реанимации, реабилитации, прогноза являются научной проблемой первостепенной важности, которая имеет прямое отношение к предотвращению врачебных ошибок и безопасности пациентов.

Однако проблема клинических алгоритмов (читай — жизнеритмов) как источник врачебных ошибок полностью выпала из поля зрения ученых. Она не рассматривается как фундаментальная проблема медицины, считается несущественной, недооценивается и не изучена в должной мере. Она отсутствует в программе безопасности пациентов.

СУЩЕСТВУЮЩИЕ МЕРЫ НЕДОСТАТОЧНЫ

Предпринимаемые сегодня меры являются недостаточными, поскольку не учитывают ошибки мышления врачей, вызванные процедурной и логической

[3] По оценкам Всемирной организации здравоохранения на март 2018 года, «число госпитализаций в мире достигает 421 миллиона в год. При этом примерно в 42,7 миллионах случаев у госпитализированных пациентов происходят неблагоприятные (adverse) эффекты. Согласно последним данным, причинение вреда пациенту занимает четырнадцатое место среди причин заболеваемости и смертности во всем мире» [100].

слабостью профессионального медицинского языка. Медицинский язык не формализован, не обеспечивает необходимую точность при решении сложных вопросов, вносит значительные трудности в мышление врачей, нуждается в доработке и совершенствовании.

Причина многих врачебных ошибок, приносящих вред пациентам — чрезмерная сложность клинического мышления, вызванная ограниченными возможностями медицинского языка, несовершенством или отсутствием качественных алгоритмов, а также сопряженных с ними недостатками медицинской литературы и медицинского образования.

Профессиональный медицинский язык не должен провоцировать врачебные ошибки, он обязан быть безопасным для пациентов, а для этого нужны специальные средства. Ключевым средством является алгоритмизация, т. е. расширение выразительных возможностей медицинского языка с помощью эргономичного алгоритмического языка высокой точности. В работе [17] автор попытался исследовать проблему безопасности, совершенствования и алгоритмизации медицинского языка и показать, что она является фундаментальной проблемой медицины, открывающей новый фронт исследований и требующей неотложного решения.

АЛГОРИТМИЧЕСКАЯ КЛИНИЧЕСКАЯ МЕДИЦИНА

Чтобы устранить слабое звено, необходимо реформировать клиническую медицину и преобразовать ее в *алгоритмическую* клиническую медицину. Научное обоснование, детальная проработка, экспериментальное подтверждение и возможность крупномасштабной реализации этого проекта представлены в монографии [17].

Ключевым элементом реформы служит медицинский алгоритмический язык. Имеется в виду язык ДРАКОН, специально доработанный для медицинских целей. Медицинский ДРАКОН имеет *высокую точность;* он помогает врачам исключать ошибки при выполнении клинических алгоритмов и обеспечить безопасность пациентов.

ДЕСЯТЬ ЦЕЛЕЙ АЛГОРИТМИЧЕСКОЙ КЛИНИЧЕСКОЙ МЕДИЦИНЫ

Алгоритмизация — это перевод медицины на алгоритмический путь развития, преследующий 10 целей:

1. сократить число врачебных ошибок в лечебно-профилактических учреждениях для повышения безопасности пациентов, повысить качество здравоохранения и показатели здоровья населения;
2. создать эргономичный медицинский алгоритмический язык высокой точности, удобный и комфортный для врачей, облегчающий их работу, исключающий умственное перенапряжение, стимулирующий безошибочную работу;
3. принять данный язык (медицинский язык ДРАКОН) в качестве медицинского стандарта для предотвращения врачебных ошибок и качественного изображения клинических алгоритмов;
4. включить указанный язык в систему медицинского образования наравне с латинским языком;
5. рекомендовать данный язык для представления алгоритмов в медицинской литературе: в учебниках, руководствах, клинических рекомендациях, протоколах диагностики и лечения, журнальных публикациях;

6. создать периодически обновляемую электронную базу данных сертифицированных эргономичных клинических алгоритмов высокой точности СЭМАВТ (в перспективе — для всех нозологических форм, предусмотренных в международной классификации болезней МКБ11);
7. устранить чрезмерную сложность клинического мышления, облегчить умственный труд врачей, создать благоприятные условия для мышления врачей, в частности, за счет мгновенного доступа к эргономичным алгоритмам, размещенным в электронной базе данных СЭМАВТ;
8. изменить методику преподавания медицинских дисциплин в медицинских учебных заведениях и системе непрерывного образования, опираясь на эргономичные алгоритмы высокой точности;
9. обеспечить высокую скорость разработки и сертификации эргономичных алгоритмов высокой точности;
10. подготовить учебные пособия и преподавателей по дисциплине «алгоритмическая клиническая медицина».

Уже говорилось, что клинические алгоритмы фактически являются не алгоритмами, а всего лишь *жизнеритмами.* Однако врачи привыкли к термину «алгоритм» и ни при каких условиях не станут от него отказываться. Это необходимо иметь в виду при общении с медиками.

МЕДИЦИНСКИЙ ЯЗЫК ДРАКОН

Алгоритмы, опубликованные в мировой медицинской литературе, как правило, не удовлетворяют требованию высокой точности. Они описывают лишь часть действий и решений врачей, а остальные опускают, что создает питательную среду для врачебных ошибок и негативно влияет на общественное здоровье.

Язык ДРАКОН позволяет устранить недостаток. ДРАКОН удовлетворяет медицинским требованиям и позволяет осуществить переход к алгоритмической клинической медицине. Эту идею активно поддержали литовские врачи. Ученые и преподаватели Литовского университета медицинских наук, опираясь на книги автора [38] [39] [17], изучили язык ДРАКОН и начали применять его на практике для разработки клинических алгоритмов, создания медицинских учебников и в медицинском образовании.

Британская вещательная корпорация BBC выпустила видеоролик и статью об успехах литовской медицины с применением языка ДРАКОН:

> «Ежегодно тысячи будущих врачей в Литве получают образование при помощи инновационной гибридной методики, которая совмещает дистанционное обучение с отработкой практических навыков в симуляционном классе. В отличие от обычных медицинских курсов, тут нет преподавателя или инструктора: его заменяет алгоритм, написанный на визуальном языке ДРАКОН — простом, понятном и исключающем возможность врачебной ошибки» [40].
>
> «Раньше в медицине алгоритмы писали, как придется, - объясняет профессор. — Просто делались какие-то блок-схемы, а какого-то единого принципа — как и что нужно писать — попросту не было. А когда мы начали работать с языком ДРАКОН, мы поняли, что он позволяет очень легко описать сложные процессы — так, чтобы они стали понятны для совершенно незнакомых с ними людей…
>
> «Гибридное обучение с использованием ДРАКОН-алгоритмов произвело настоящую революцию в медицинском образовании» [41].

ОТЗЫВЫ ЛИТОВСКИХ ВРАЧЕЙ О ЯЗЫКЕ ДРАКОН

Доктор медицинских наук А. Кудрявичене, неонатолог:

> «Язык ДРАКОН – отличный инструмент для обучения практическим навыкам и их стандартизации. Он позволяет выявить все, даже мельчайшие, но очень важные действия».

Доктор медицинских наук, профессор М. Ключинскас, акушер-гинеколог:

> «Язык ДРАКОН позволяет систематизировать процессы с минимальным применением текста – как при организации работы, так и при выполнении медицинских процедур. Он помогает всем одинаково понимать и выполнять конкретные действия… Позволяет ускорить запоминание действий».

Доктор медицинских наук, профессор Ж. Дамбраускас, абдоминальный хирург:

> «Огромным преимуществом языка ДРАКОН является то, что он позволяет конкретно выявить все этапы процедуры или процесса… Мысленно можешь повторить процесс этап за этапом, а затем каждый этап разделить на шаги… Процедуру или процесс можно выполнить мысленно, а затем и в реальности. ДРАКОН является инструментом мысленной тренировки».

Врач Б. Кумпайтене, анестезиолог-реаниматолог:

> «Польза языка ДРАКОН для разрабатывающего алгоритм автора состоит в том, что проявляется, кристаллизуется и стандартизируется каждый навык, каждая процедура. Польза для обучающегося – это ясный путь выполнения действий. ДРАКОН дает ответ на вопросы “что делать, если”».

А. Вилейките, координатор медицинского учебного Центра исследования кризисов:

> «Применение языка ДРАКОН позволяет стандартизировать и эргономично представить самую сложную процедуру… Если всё правильно описано на ДРАКОНе, значит, всё будет отлично выполнено».

Доктор медицинских наук, профессор Динас Вайткайтис, зав. кафедрой экстремальной медицины:

> «Язык ДРАКОН даёт ясность и чёткость процессам, применяемым в медицине. Он позволяет “автоматизировать” обучение студентов практическим навыкам. Может стать основой для технологии принятия клинических решений».

Доктор медицинских наук П. Добожинскас, исполнительный директор медицинского учебного Центра исследования кризисов:

> «Применение языка ДРАКОН действенно помогает в создании и описании сложных, динамичных решений медицинских проблем. Тем самым значительно облегчается проведение стандартизированного симуляционного обучения, внедряя культуру безопасности пациентов и принципы качественного оказания медицинских услуг в масштабах медицинского учреждения, региона или государства».

Доктор мед. наук, проф. Р.Й. Надишаускене, зав. клиники акушерства и гинекологии, главный специалист по акушерству и гинекологии Литовской республики:

> «Алгоритмизация медицины подразумевает значительную перестройку системы медицинского образования и перевод ее на алгоритмический путь… Накопленный в Литве практический положительный опыт

> использования языка ДРАКОН для представления сложных и разнообразных медицинских алгоритмов может послужить серьезной основой для принятия крупных структурных решений руководителями здравоохранения и системы медицинского образования в области алгоритмизации медицины».

ЗАКЛЮЧЕНИЕ ФУМО МИНЗДРАВА РФ «КЛИНИЧЕСКАЯ МЕДИЦИНА» О ЯЗЫКЕ ДРАКОН

Заключение подготовлено Федеральным учебно-методическим объединением в системе высшего образования по группе специальностей «Клиническая медицина». В документе, в частности, говорится:

> «Предложенный В.Д. Пароннджановым подход к представлению медицинских данных и знаний на языке… «Дракон» представляет несомненный интерес… Для обеспечения работы целесообразно разработать национальный стандарт представления медицинских данных и клинических документов в виде конструкций предложенного языка» [42].

В Красноярском медуниверситете доцент С.Д. Гусев разработал Методическое пособие по созданию клинических алгоритмов на языке ДРАКОН с помощью программы DrakonHub [31]. Координационный совет по области образования «Здравоохранение и медицинские науки» рекомендовал данную работу в качестве учебного пособия по группе специальностей «Клиническая медицина» [31].

ВЫВОДЫ

1. Многие клинические алгоритмы представлены в медицинской литературе в виде текста на естественном языке. Это недопустимо, потому что последний не пригоден для записи безошибочных алгоритмов и провоцирует врачебные ошибки.
2. Для медицинской литературы и системы медицинского образования характерны низкое качество клинических алгоритмов, что неблагоприятно отражается на показателях здоровья населения.
3. Врачебные ошибки опасны тем, что могут привести к смерти пациентов, стойкой инвалидности или причинить иной вред. Согласно последним данным, смертность в больницах США, которую можно предотвратить, превышает 200000 человек в год.
4. Предпринимаемые сегодня меры по безопасности пациентов недостаточны, ибо не учитывают ошибки мышления врачей, вызванные процедурной и логической слабостью профессионального медицинского языка.
5. Профессиональный медицинский язык не должен провоцировать врачебные ошибки, он обязан быть безопасным для пациентов, а для этого нужны специальные средства.
6. Ключевым средством является алгоритмизация, т. е. расширение выразительных возможностей медицинского языка с помощью эргономичного алгоритмического языка высокой точности ДРАКОН.
7. Необходимо реформировать клиническую медицину и преобразовать ее в алгоритмическую клиническую медицину.
8. Язык ДРАКОН помогает врачам исключать ошибки при выполнении клинических алгоритмов и обеспечить безопасность пациентов.

9. Накопленный в Литве практический положительный опыт использования языка ДРАКОН для представления сложных и разнообразных клинических алгоритмов может послужить основой для принятия крупных структурных решений руководителями здравоохранения и системы медицинского образования в области алгоритмизации медицины.

Глава 29

КАК УЛУЧШИТЬ ЖИЗНЕРИТМЫ

ВВЕДЕНИЕ

Жизнеритмы получили широкое распространение в различных областях и играют огромную роль в жизни человека и общества. Обойтись без них невозможно. Они проявляют себя всюду, где люди занимаются сложной работой. Однако их нельзя путать с алгоритмами. По сравнение с алгоритмом, жизнеритм — вещь неполноценная, второсортная.

Между тем подобная путаница происходит постоянно. Многие авторы, приводя примеры повсеместного использования алгоритмов, фактически говорят о жизнеритмах. Например, Н. А. Криницкий в книге «Алгоритмы вокруг нас» пишет:

> «Всюду алгоритмы. Они окружают нас, переплетаются, проникают друг в друга. Шага нельзя ступить, не наталкиваясь на них… Многие инструкции и приказы, определяющие наши действия на работе — это алгоритмы. Даже окончив работу и желая отдохнуть, мы постоянно сталкиваемся с ними» [43] [18, p. 9].

Недостатки жизнеритмов неблагоприятно отражаются на деловой и производственной деятельности. Подобные дефекты могут привести к печальным последствиям вплоть до смертельного исхода. К счастью, этого можно избежать. Язык ДРАКОН позволяет навести порядок в этой плохо исследованной области, то есть выявить недочеты жизнеритмов и во многих случаях ликвидировать их.

ДЛЯ АЛГОРИТМОВ ТЕОРИЯ ЕСТЬ, А ДЛЯ ЖИЗНЕРИТМОВ — НЕТ

Для алгоритмов существует детально разработанная теория и подробные правила для их записи. Это открыло дорогу ко многим важным достижениям: появились компьютеры, языки программирования и всемирная практика использования компьютеров, микропроцессоров, контроллеров, гаджетов для решения разнообразных задач и для управления различными объектами.

Ничего подобного для жизнеритмов нет. Нет ни теории, ни общепринятых правил записи. Единообразие отсутствует. Каждый записывает жизнеритмы на свой лад.

Имеющиеся стандарты на блок-схемы алгоритмов (ГОСТ 19.701-90 и ISO 5807-85) устарели и не соответствуют современным требованиям. Диаграммы деятельности языка UML (activity diagrams) также имеют изъяны.

Таким образом, возникает новая важная задача. Необходимо повысить качество жизнеритмов, предложить эффективные средства для их записи и стандартизации.

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ЖИЗНЕРИТМОВ

Ограничимся императивным и процедурным подходом к описанию алгоритмов и жизнеритмов. В рамках этого подхода в алгоритмах можно выделить следующие части:

- поток управления (control flow), позволяющий выполнять последовательности команд, разветвления и циклы;
- алгоритмическую логику;
- процессы реального времени;
- структуры данных (data structure);
- комментарии, пояснения, выноски.

Этот список имеет принципиальное значение. Он позволяет сопоставить алгоритмы и жизнеритмы, а также указать в чем они совпадают, а в чем расходятся. В жизнеритмах имеются все позиции списка за исключением структур данных.

Данные в жизнеритмах имеют имя, но описание данных либо отсутствует, либо является неформальным или неполным.

Для примера рассмотрим команду жизнеритма «Принеси пузырек с клеем». Пузырек и клей — это имена данных; описания данных здесь нет. Человек легко выполнит команду, так как прекрасно знает, что такое пузырек и что такое клей. Компьютер, напротив, не обладает такими знаниями и не сможет выполнить команду.

Жизнеритмы и алгоритмы изображаются в виде графических чертежей в точном соответствии с правилами эргономики и правилами языка ДРАКОН. Во избежание ошибок их следует рисовать с помощью программы «ДРАКОН-конструктор». Предыдущие главы содержат материал по жизнеритмам. Более подробно эти вопросы, включая теоретические основы жизнеритмов, изложены в книгах автора (перечислены в хронологическом порядке) [43] [44] [38] [39] [17].

ПРОЦЕДУРНЫЕ И ДЕКЛАРАТИВНЫЕ ЗНАНИЯ

Принято различать процедурные и декларативные знания (знания «как» и знания «что») [45].

Процедурные (императивные, алгоритмические, операторные, рецептурные) знания содержат сведения о последовательности информационных или физических действий, а также о выборе пути при разветвлении процессов. Примерами являются алгоритмы, компьютерные программы, бизнес-процессы, а также любые технологические процессы (промышленные, сельскохозяйственные, медицинские).

Декларативные (дескриптивные, атрибутивные, описательные) знания — это знания не о действиях, а об описаниях информационных и физических объектов, научные теории (знания «почему»), знания о фактах, о людях и т. д.

Процедурные знания играют особую роль в человеческой жизни. В самом деле, человек — деятельное существо. От рождения до смерти он непрерывно действует. Деятельность выражает сущность жизни. Бездеятельность — это смерть. Поэтому знания о действиях и структуре деятельности (процедурные знания) составляют важнейший компонент человеческих знаний, их основу. Можно предположить, что в системе человеческих знаний процедурные знания играют фундаментальную роль — роль несущей конструкции или каркаса, который скрепляет между собой (склеивает, цементирует) отдельные фрагменты декларативных знаний. Сказанное хорошо согласуется с известным мнением, что

> «большинство знаний об окружающем мире можно выразить в виде процедур или последовательности действий, направленных на достижение конкретных целей» [46].

УНИВЕРСАЛЬНЫЙ ЯЗЫК ДЛЯ ВЗАИМОПОНИМАНИЯ

Большой интерес представляет вопрос: можно ли создать единый универсальный язык представления профессиональных знаний, удобный для специалистов любой профессии и позволяющий улучшить взаимопонимание между людьми?

Для декларативных знаний ответ, очевидно, будет отрицательным. Потому что нельзя скрестить ужа с ежом и придумать разумный и полезный гибрид электрической схемы и географической карты (или конструкторского чертежа). Такой путь неизбежно ведет в тупик. Поэтому придется смириться с выводом, что специалисты разных профессий будут и впредь использовать множество самых разнообразных декларативных языков. Унификация здесь невозможна.

Иное дело процедурные знания. Знания специалистов любого профиля имеют в точности одинаковую процедурную структуру, которая нисколько не зависит от конкретной специальности и предметной области. Отсюда проистекает вывод: для отображения любых процедурных знаний можно использовать один и тот же язык, общий для всех научных и учебных дисциплин. Язык ДРАКОН предназначен именно для этой цели.

ДРАКОН призван сыграть роль межотраслевого и междисциплинарного языка, содействующего решению важнейшей проблемы — проблемы взаимопонимания между учеными и специалистами.

ИСТОРИЯ ПРОЦЕДУРНОГО ЗНАНИЯ

Знание играет ключевую роль в науке и технологиях. В течение длительного времени наука развивалась в форме декларативного знания. Со времен глубокой древности в старинных рукописях декларативное знание было доминирующим. Процедурное знание встречалось лишь изредка.

Произведения Платона и Аристотеля — типичный пример декларативного знания. Процедурное знание можно найти у математиков (алгоритм Евклида) и в трудах древних врачей, например, у Гиппократа.

Подавляющее большинство письменных научных источников содержат декларативное знание. История науки — это в основном история декларативного знания.

Противопоставление «процедурное – декларативное» появилось сравнительно недавно. В 1945 году английский философ Гилберт Райл ввел разграничение между знанием «как» (knowing-how) и знанием «что» (knowing-that):

- знание КАК завязать прямой (гераклов) узел на веревке;
- знание ЧТО королева Виктория умерла в 1901 году.

Со временем это различие привело к разграничению процедурного и декларативного знания в качестве моделей долговременной памяти человека.

Процедурная память — это память о том, как выполнять различные действия: как завязать шнурки, ездить на велосипеде, управлять автомобилем. Такая память является преимущественно неосознанной; указанные действия трудно или даже невозможно выразить словами.

Согласно когнитивной психологии, процедурные знания — это знания, для овладения которыми обязательно нужна практика, тренировка мышечной памяти. Например, чтобы научиться плавать, играть на музыкальных инструментах, управлять трактором. В таких случаях говорят, что руки и ноги сами знают, что нужно делать. Такие знания называют молчаливыми, или неявными (tacit knowledge).

Нас молчаливое знание не интересует. Для наших целей нужна иная трактовка процедурного знания.

ОПРЕДЕЛЕНИЕ ПОНЯТИЯ «ПРОЦЕДУРНОЕ ЗНАНИЕ»

Процедурное знание — общее понятие для алгоритмов и жизнеритмов. В свою очередь, понятие жизнеритм включает в себя поток работ (workflow), клинический алгоритм, бизнес-процесс, стандартную операционную процедуру (СОП) и т. п. Возможно, сюда же можно добавить хауту (how-to) — набор инструкций для выполнения каких-либо задач; см., например, сайт wikiHow, где представлены практические руководства с точными и актуальными инструкциями [47].

Язык ДРАКОН предлагается рассматривать как единую нотацию для любых видов процедурного знания.

КАК УЛУЧШИТЬ ЖИЗНЕРИТМЫ

Существующие жизнеритмы должны претерпеть значительные изменения и превратиться в эргономичные жизнеритмы высокой точности. Следует осуществить стандартизацию записи жизнеритмов. Для этого они должны быть изображены на языке ДРАКОН в соответствии с правилами языка ДРАКОН.

Задача

- Необходимо кардинально повысить качество жизнеритмов и в максимальной степени приблизить их свойства к образцу — к свойствам алгоритмов.
- Решить эту задачу позволяет язык ДРАКОН.

ВТОРАЯ АЛГОРИТМИЧЕСКАЯ ВСЕЛЕННАЯ

Алгоритмы известны всем или почти всем. В отличие от них жизнеритмы обычно остаются в тени и не купаются в лучах славы. Между тем они играют большую роль в науке, технике, образовании, культуре и других областях, но зачастую остаются незамеченными.

Настало время устранить подобное несоответствие, уравнять в правах два понятия: алгоритмы и жизнеритмы, разграничив области их применения. Поскольку эти понятия являются фундаментальными, надо уметь различать их на основании четких критериев, которые изложены выше.

Подобное размежевание играет важную роль — оно знаменует открытие второй алгоритмической вселенной. Первая вселенная — царство алгоритмов, вторая — территория жизнеритмов.

Жизнеритмы — это человеческие алгоритмы, их могут выполнять только живые люди. Жизнеритмы свидетельствуют о невозможности заменить человека машиной при выполнении многих операций (на данном историческом этапе). Как и

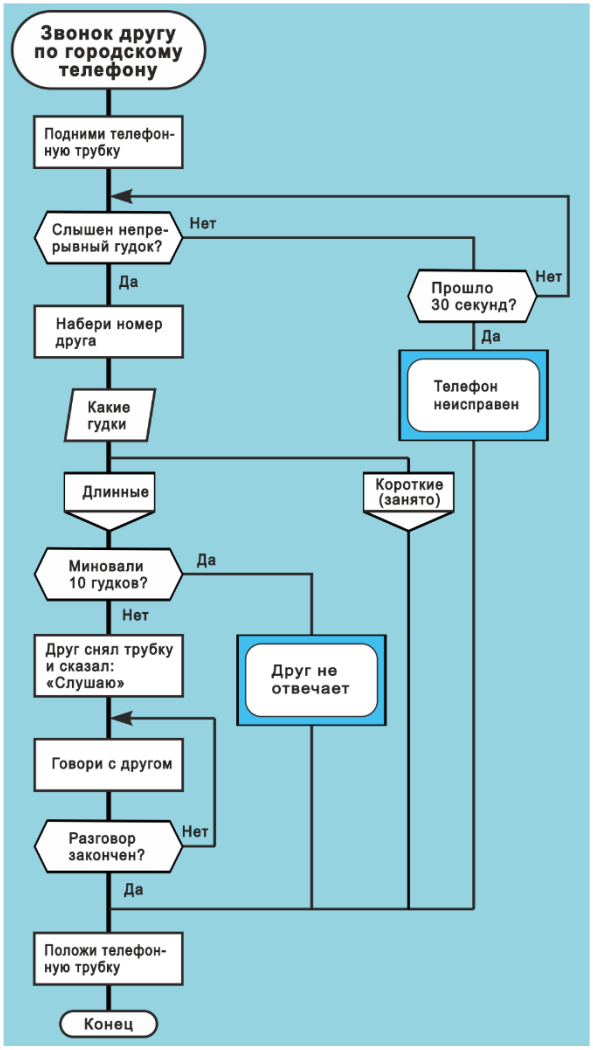


Рис. 154. Жизнеритм «Звонок другу по городскому телефону» [97]

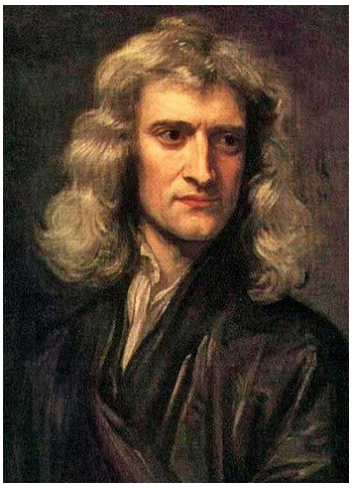
алгоритмы, они включают линейные, разветвленные и циклические последовательности действий, а также параллельные процессы, но, в отличие от алгоритмов, содержат не полностью определенные шаги (рис. 154).

Если бы все шаги были полностью определены, жизнеритм бы сразу исчез, превратившись в алгоритм. Именно этот дефект (наличие неприемлемых для компьютера шагов) требует человеческой смекалки, которая у компьютеров отсутствует.

ЖИЗНЕРИТМЫ ПОМОГАЮТ ИЗУЧАТЬ АЛГОРИТМЫ

Знания об алгоритмах становятся ясными и доступными, если подобрать простые и доходчивые примеры.

Великий Исаак Ньютон говорил: «При изучении наук примеры полезнее правил» [48]. Следуя мудрому совету, в данной книге мы вели рассказ об алгоритмах, используя простые и поучительные примеры. Такими примерами являются специально подобранные бытовые алгоритмы, известные каждому. Бытовые алгоритмы — это и есть жизнеритмы. Такой прием хорош тем, что позволяет упростить трудный материал. И сэкономить драгоценное время читателей.



Исаак Ньютон (1642 — 1727): «При изучении наук примеры полезнее правил»

ВЫВОДЫ

1. Жизнеритм — важное понятие, которое принципиально отличается от алгоритма.
2. В настоящее время многие жизнеритмы имеют низкое качество.
3. Язык ДРАКОН позволяет кардинально повысить качество жизнеритмов и в максимальной степени приблизить их свойства к свойствам алгоритмов.
4. Теория жизнеритмов включает поток управления, алгоритмическую логику и процессы реального времени.
5. В отличие от алгоритмов, жизнеритмы не содержат описание данных.
6. Следует осуществить стандартизацию записи жизнеритмов. Для этого они должны:
   - изображаться на языке ДРАКОН,
   - соответствовать правилам языка ДРАКОН,
   - разрабатываться с помощью программы «ДРАКОН-конструктор».
7. Процедурное знание — родовое понятие для алгоритмов и жизнеритмов.
8. Желательно иметь единый стандарт для удобного описания алгоритмов и жизнеритмов. В качестве такого стандарта предлагается алгоритмический язык ДРАКОН.
9. ДРАКОН призван сыграть роль межотраслевого и междисциплинарного языка, чтобы помочь решению проблемы взаимопонимания между учеными и специалистами.

Часть 7

БЕЗОШИБОЧНЫЕ АЛГОРИТМЫ И ПОДАВЛЕНИЕ ОШИБОК

Язык ДРАКОН использует оригинальный метод борьбы с ошибками в алгоритмах и программах. Метод называется «подавление ошибок». Данный метод отличается высокой эффективностью. Ошибки в алгоритмах и программах значительно уменьшаются или даже исчезают почти полностью.

Глава 30

БЕЗОШИБОЧНЫЕ АЛГОРИТМЫ. АНАЛИЗ КАТАСТРОФЫ САМОЛЕТА БОИНГ 737 MAX 8

ЦЕЛЬ — БЕЗОШИБОЧНОСТЬ

В этой главе мы откроем новую тему — тему безошибочных алгоритмов, которая будет продолжена в следующих четырех главах.

Ошибки в алгоритмах и программах — острая проблема. Исправление промахов и ошибок приводит к дополнительным затратам труда и времени, перерасходу средств. Отчасти это объясняется тем, что понятие алгоритма, его свойства и нотации разрабатывались без учета проблемы человеческих ошибок.

Эффективная и удобная нотация для записи алгоритмов до сих пор не создана. Задача данной книги — предложить такую нотацию и показать, что она содействует безошибочности.

Чтобы подчеркнуть важность понятия «безошибочный алгоритм», полезно рассмотреть инцидент с самолетом Боинг 737 Max 8, алгоритмы которого оказались ошибочными и привели к катастрофе.

МАКРОАЛГОРИТМЫ

Макроалгоритмы — это алгоритмы в широком смысле. Макроалгоритмы делятся на компьютерные алгоритмы (которые выполняются компьютерами) и жизнеритмы, выполняемые людьми.

Макроалгоритм большой человеко-машинной системы содержит взаимоувязанные и согласованные между собой алгоритмы и жизнеритмы.

Примером большой человеко-машинной системы служит фирма Боинг, огромная фирма, на которой трудятся 150000 человек. Это люди высочайшей квалификации, обладающие огромными знаниями. Что же их подвело?

ЧТО СЛУЧИЛОСЬ С БОИНГОМ 737 MAX 8

Через считанные минуты после взлета, когда надо было срочно набирать высоту, произошло невероятное — автоматическая система управления внезапно направила нос самолета к земле. Экипаж изо всех сил пытался выровнять лайнер и избежать катастрофы. Но все было тщетно. Автоматика была непреклонна и по крутой траектории вела воздушное судно вниз — навстречу неминуемой гибели.

Трагедия повторилось дважды: 29 октября 2018 года в Индонезии и спустя полгода, 10 марта в Эфиопии. Погибли 346 человек: все пассажиры и оба экипажа. Выживших не было.

В ТИСКАХ КОНКУРЕНЦИИ. КАК И ПОЧЕМУ ПОЯВИЛСЯ БОИНГ 737 MAX

Знаменитая американская корпорация Боинг была озабочена конкуренцией с мощной европейской фирмой Эйрбас.

Соперничество двух авиагигантов всегда было напряженным. Самолеты Боинг 737 и европейский А320 стали рабочими лошадками мира пассажирских перевозок, тысячами летавших на маршрутах короткой и средней дальности по всей планете. Их продажи были для обоих концернов надежным источником дохода. Рынок был поделен примерно поровну. Но обновленный лайнер А320 грозил нарушить равновесие и вывести Эйрбас далеко вперед.

Новые А320 обещали быть существенно дешевле в эксплуатации. В последние годы расходы на топливо составляли почти 25% операционных расходов авиакомпаний. Эйрбас обещал, что новые самолеты будут на 15% экономичнее прежних.

Заказы на новый самолет, названный A320neo, посыпались на европейский концерн. Спрос на лайнер 737 между тем заметно снизился. Боинг не мог оставить это без ответа.

Современные двигатели гораздо тише и экономичнее своих предшественников, но по технологическим причинам они гораздо больше в диаметре.

Боинг собирался установить на самолетах 737 Max наиболее экономичные двигатели LEAP производства фирмы CFM. Но Эйрбас его опередил.

Как только было объявлено, что Эйрбас будет использовать более бережливые двигатели на новом самолете А320, у Боинга не осталось выхода. Поступить иначе означало совершить коммерческое самоубийство.

И работа закипела! Однако под низко расположенное крыло лайнера 737 новый двигатель не влезал. Его пришлось выдвинуть на пилоне вперед и вверх. Это решило одну проблему, но создало другую.

Новое распределение веса и аэродинамика крыла с двигателем придали самолету необычные характеристики управляемости. У лайнера 737 Max обнаружилась тенденция сильно задирать нос, особенно если угол атаки (угол между осью фюзеляжа и землей) слишком велик.

Это очень неприятно. Излишне большой угол атаки может привести к сваливанию — то есть к тому, что набегающий поток воздуха перестанет создавать подъемную силу, которая поддерживает самолет в воздухе. И он начнет стремительно терять высоту. Летчики тщательно избегают подобных ситуаций.

Опытные пилоты-испытатели обнаружили, что новый самолет 737 Max управляется совсем не так, как предыдущие поколения 737 модели. Значит, прежние навыки пилотирования здесь не годятся.

Что делать? Создавать учебный симулятор нового лайнера и переучивать сотни пилотов? Но ведь это лишнее время и огромные расходы [49].

ЗАЧЕМ ПОНАДОБИЛАСЬ СИСТЕМА MCAS

Творческая мысль на выдумки хитра. Находчивые инженеры Боинга придумали палочку-выручалочку под названием «Улучшение характеристик системы маневрирования» — Maneuvering Characteristics Augmentation System (MCAS) [50]. Она должна была убрать головную боль и сделать лайнер 737 Max похожим в управлении на предыдущие самолеты.

Заодно программа MCAS устраняла недочеты в аэродинамике. Она автоматически предотвращала свойственные лайнеру 737 Max попытки

самостоятельно задрать нос, когда угол атаки чрезмерно велик. Тем самым устраняется опасность сваливания, вызванная задранным носом.

При этом разработчики подчеркивают, что система MCAS не является системой предотвращения сваливания, а нужна для того, чтобы «улучшить горизонтальную стабильность самолета, чтобы он походил в управлении на остальные 737».

Хотя между предыдущим поколением и лайнером 737 Max есть значительные отличия, но — благодаря замечательной системе MCAS — пилоты, летавшие на старых Боингах, могли чувствовать себя уверенно. От них требовалось совсем немного: пройти короткий онлайн-курс обучения на флешке — и можно спокойно сесть за штурвал нового самолета.

Так было задумано и допущено к полетам. Но, как говорится, гладко было на бумаге [49].

ЧТО ПОЛУЧИЛОСЬ НА САМОМ ДЕЛЕ. ИГРА СО СМЕРТЬЮ В КАБИНЕ БОИНГА

В кабине раздавались новые сигналы тревоги. Командир безуспешно пытался восстановить контроль над самолетом. Надо срочно набирать высоту! Но как? Когда капитан Яред Гетачу пытался поднять нос Боинга, электронные системы опускали его обратно.

Просто тянуть штурвал на себя оказалось недостаточно. Он щелкнул переключателем на штурвале — это должно было восстановить аэродинамический баланс самолета и направить его вверх. Увы, спустя несколько секунд переключатель автоматически вернулся в исходное положение.

Штурвал затрясся — механическое предупреждение о том, что ситуация становится угрожающей: самолет опасно близок к сваливанию. Звуковая сигнализация продолжала трезвонить о потере высоты.

Пилоты отключили часть электронных систем и стали управлять самолетом вручную. Но контролировать самолет становилось все труднее. К этому моменту он набрал скорость, аэродинамические силы, удерживающие его в воздухе, стали настолько велики, что противиться им с помощью ручного управления стало невозможно.

Пилоты снова включили электронику, и капитан вновь попытался поднять нос "Боинга" с помощью переключателя на штурвале. На какое-то время это помогло: самолет вновь начал набирать высоту.

Но затем набор высоты прекратился — снова вмешались

Рис. 155. Командир самолета Боинг 737 Max Яред Гетачу (справа), управлявший рейсом 302 авиакомпании Ephiopian Airlines, вылетевшим из Аддис-Аббебы 10 марта 2019 года в 8:38 по местному времени

компьютеры, которые упрямо гнули свое. Новое предупреждение извещало о том, что самолет движется со слишком большой скоростью — он начал пикировать к земле.

В отчаянии капитан Гетачу обратился к помощнику, и они вдвоем налегли на штурвалы в надежде выровнять самолет силой собственных мускулов.

Но и эта последняя попытка не удалась. Самолет продолжал набирать скорость, пикируя все круче, пока не врезался в землю на скорости более 800 километров в час. Всего через шесть минут после взлета.

Пятью месяцами ранее такой же Боинг 737 Max индонезийской компании Lion Air вылетел обычным рейсом из Джакарты.

Полет в город на западе Индонезии должен был занять около часа. Но спустя несколько минут после взлета пилоты оказались в похожей ситуации.

Самолет, казалось, жил своей жизнью, норовя снизиться, несмотря на усилия пилотов заставить его набрать высоту.

Они не понимали, что происходит. Каждый раз, когда они поднимали нос самолета вверх, спустя несколько секунд автоматика упорно толкала его вниз.

Это произошло более 20 раз подряд. На земле диспетчеры забеспокоились, глядя на экраны радаров — Боинг явно терял высоту. Один из пилотов сообщил, что у них возникли проблемы с управлением.

Пытаясь разобраться, экипаж растерялся еще больше — приборы командира и второго пилота показывали разную высоту полета. Пилоты не знали точно, на какой высоте они летят и с какой скоростью.

В конце концов, они полностью потеряли контроль над самолетом. Он вошел в крутое пике и на большой скорости врезался в воды Яванского моря [49].

СЕРЬЕЗНЫЙ ПРОСЧЕТ РУКОВОДСТВА ФИРМЫ БОИНГ И FAA В ОБЛАСТИ БЕЗОПАСНОСТИ ПОЛЕТОВ

Потеря двух новых самолетов, трагическая гибель свыше трехсот человек, запрет на продолжение полетов и отказ ряда авиакомпаний от закупок новых самолетов 737 Max явились тяжелым ударом для Боинга.

Одновременно это выявило упущения в работе Федеральной авиационной администрации США — Federal Aviation Administration (FAA) [51], которая отвечает за безопасность и сертификацию самолетов и летчиков.

Почему вопиющие ошибки в области безопасности самолетов и подготовки пилотов остались незамеченными? Было начато расследование в Конгрессе США, ФБР и других инстанциях [52]. Здесь мы должны прервать рассказ и вернуться к теме книги.

Нас интересует проблема алгоритмов. Какую роль сыграли алгоритмы в трагедии лайнера 737 Max?

Мы предполагаем, что причина авиационного инцидента связана с низким качеством и неполнотой алгоритмов и жизнеритмов.

АЛГОРИТМЫ НА СКАМЬЕ ПОДСУДИМЫХ

Применительно к лайнеру 737 Max слово «жизнеритм» можно трактовать как бизнес-процесс. Имеются в виду бизнес-процессы в широком смысле слова, описывающие функционирование не только коммерческих фирм, но и государственных учреждений (FAA), а также взаимодействие между ними.

При традиционном подходе алгоритмическая часть бизнес-процесса зачастую описывается не строго и эргономически не удовлетворительно. Можно предположить,

что — как в случае с лайнером 737 Max — анализ алгоритмов и жизнеритмов не позволил выявить погрешности проектирования и своевременно устранить их. Не позволил предотвратить беду и спасти людей.

Алгоритмы и жизнеритмы — абстрактные понятия. Однако они должны быть четко описаны и представлены в удобочитаемой форме. Они должны быть пригодны для быстрого чтения и понимания. Если это не так, появляются разночтения и взаимное непонимание — предвестники беды.

Иными словами, все алгоритмические подробности и оттенки жизнеритмов должны быть тщательно проработаны и аккуратно изложены. Потому что от точности и эргономичности описаний и моделей может зависеть жизнь людей.

КРИЗИС САМОЛЕТА 737 MAX ИЛИ КРИЗИС ПОНЯТИЯ «АЛГОРИТМ»?

Алгоритмы находятся внутри компьютеров и выполняются автоматически. Жизнеритмы описывают поведение людей в разнообразных бизнес-процессах. Нужно иметь и то, и другое. Необходимо тщательно сопрягать алгоритмы и жизнеритмы.

При создании и эксплуатации самолета 737 Max понятие «жизнеритмы» охватывает следующую группу процессов:

- внутренние бизнес-процессы фирмы Боинг, описывающие взаимодействие работников различных отделов и подразделений фирмы друг с другом;
- бизнес-процессы, регламентирующие взаимодействие сотрудников фирмы Боинг и Федеральной авиационной администрации FAA;
- бизнес-процессы, описывающие взаимодействие сотрудников Боинга с авиакомпаниями и другими контрагентами;
- бизнес-процессы, регламентирующие обучение и сертификацию пилотов самолета 737 Max, в том числе, в критических и нештатных ситуациях полета.

Мы исходим из того, что анализ алгоритмов и жизнеритмов должен показывать реальное состояние дел и демонстрировать упущения, ошибки и слабые места.

Это очень жесткое требование. Двойная авария лайнера 737 Max показала, что данное требование не выполняется. Алгоритмы и жизнеритмы не позволили заблаговременно выявить и устранить недостатки, допущенные при разработке и эксплуатации самолета 737 Max.

Почему? Может быть, понятие алгоритма нуждается в уточнении?

АНАЛИЗ ПОНЯТИЯ «АЛГОРИТМ»

Уже говорилось, что понятие алгоритма, его свойства и нотации создавалось без оглядки на проблему ошибок. Между тем ошибки, упущения и слабые места во многих случаях являются постоянным спутником сложных алгоритмов и сложных программных комплексов. Однако увидеть, обнаружить и заблаговременно исправить подобные недостатки трудно, что ведет к лишним издержкам.

Ошибочные алгоритмы никому не нужны. Алгоритм с ошибкой — это не алгоритм. Актуальной задачей является уточнение понятия «алгоритм» исходя из требования безошибочности. Это требование относится не только к одиночным алгоритмам, но и к алгоритмическим системам, содержащим сотни и тысячи взаимодействующих алгоритмов.

Алгоритм — сложное понятие, в котором можно выделить:

- определение алгоритма (интуитивное и формальное);
- свойства алгоритма;
- нотации, т. е. способы записи алгоритма.

До сих пор свойства и нотации алгоритма рассматривались и изучались в отрыве от потребностей практики, которая заинтересована в сокращении числа ошибок. Это имело негативные последствия. Недооценка проблемы безопасности открыла путь к использованию нежелательных методов разработки алгоритмов — небезопасных методов, порождающих ошибки. Такое положение следует признать неприемлемым.

КОМПЛЕКСНАЯ ПРОГРАММА УМЕНЬШЕНИЯ ЧИСЛА ОШИБОК

Наряду с устоявшимся понятием «алгоритм» целесообразно ввести понятие «безошибочный алгоритм». Речь идет не просто о термине, а о нацеленной в будущее комплексной программе значительного сокращения ошибок в алгоритмах за счет использования новых (когнитивно-эргономических и формальных) методов. Программа называется «подавление ошибок» и будет описана в главах 31-34.

Свойства и нотация безошибочных алгоритмов должны:

- оцениваться с учетом требования безошибочности, которое следует считать приоритетным;
- тщательно выбирать и рекомендовать такой способ записи алгоритма, который предотвращает ошибки;
- запрещать способы записи, содействующие появлению ошибок (или присваивать им низкую оценку).

Создание безошибочных алгоритмов следует начинать с разработки эргономичной нотации, которая способна предотвращать ошибки.

ПОНЯТНОСТЬ И ПОНИМАЕМОСТЬ БЕЗОШИБОЧНОГО АЛГОРИТМА

Следует различать два свойства алгоритма: понятность и понимаемость.

Свойство «понятность» говорит о том, что алгоритм должен быть понятен компьютеру, т. е. состоять из инструкций, входящих в его систему команд. Это разумное и очевидное свойство алгоритма.

К сожалению, при этом полностью выпадает из поля зрения не менее важное свойство «понимаемость алгоритма» для человека. Имеется в виду человек, который читает или изучает алгоритм с целью выявления ошибок, упущений, слабых мест и иных недостатков. Такая проверка может проводиться в любое время в процессе жизненного цикла алгоритма. Проверку могут проводить разные люди: автор алгоритма, его руководитель, заказчик, специалисты по сопровождению (эксплуатации) алгоритма и др.

В главе 3 уже говорилось, что понимаемость алгоритма для человека (в отличие от понятности для компьютера) означает удобочитаемость, то есть ясность, доходчивость, пригодность алгоритма для быстрого и легкого понимания.

Понимаемость есть совокупность свойств алгоритма, характеризующая затраты усилий пользователя на понимание логической концепции этого алгоритма[4]. Чтобы

---

[4] Термин «понимаемость» (understandability) определен в стандарте «ГОСТ 28806-90 Качество программных средств. Термины и определения» [96].

.

облегчить выявление ошибок при зрительной проверке за столом, указанные затраты должны быть минимальными. Отсюда следует, что понимаемость есть свойство алгоритма минимизировать интеллектуальные усилия, необходимые для его понимания при зрительном восприятии алгоритма человеком.

Зачем нужна понимаемость? Она необходима при создании безошибочных алгоритмов.

Если бы алгоритмы и жизнеритмы (бизнес-процессы) фирмы Боинг и FAA обладали надлежащим свойством «понимаемость», то ошибки, упущения и слабые места в самолете 737 Max были бы своевременно выявлены и устранены, катастрофа не произошла бы, а пассажиры и экипажи остались бы живы.

ДИСКУССИЯ О ПОНИМАНИИ АЛГОРИТМОВ

— Можно ли читать алгоритмы, как увлекательную повесть, быстро и с удовольствием?

— Нет, нельзя.

— Как сделать алгоритмы легкими и удобными для изучения?

— Увы, это невозможно.

— Почему?

— Потому что алгоритмы очень трудны и предназначены для вдумчивого, серьезного, медленного чтения, обеспечивающего глубокое понимание.

— Все это так, но жизнь идет вперед и ставит новые задачи. То, что вчера было невозможно, завтра станет возможным. Жизнь требует, чтобы сложные алгоритмы стали удобными для людей — дружелюбными, понятными, доходчивыми. Алгоритмы должны быть легкими для быстрого восприятия и усвоения, пригодными для быстрого выявления ошибок.

ПОЧЕМУ АЛГОРИТМЫ ТРУДНЫ ДЛЯ ПОНИМАНИЯ

Низкая понимаемость алгоритмов и программ — важный недостаток современного программирования. Джозеф Фокс, руководитель отделения федеральных систем IBM, у которого в подчинении было 4400 человек, объясняет, в чем причина:

> «Некий превосходный программист спроектировал и написал программу определения орбитальных характеристик искусственного спутника Земли. Он первым закончил программирование, все работало правильно, память попусту не тратилась. Программа была написана на языке Фортран и занимала около четырех страниц плотного фортрановского текста. Он знал свою программу вдоль и поперек.
>
> «Через три месяца его попросили добавить к программе несколько новых функций. Он достал документацию (описание программы) и принялся ее изучать. Три или четыре дня он пытался понять, что же происходит в его программе. А ведь он ее сам написал! Сколько бы сил он потратил, если бы это была чужая программа!» [53].

Эта удивительная история подтверждает простую истину: алгоритм вещь сложная и понять его непросто. Оказывается, даже сам автор программы спустя три месяца не сумел расшифровать свой собственный алгоритм. Чтобы прочитать всего четыре страницы и докопаться до истины, превосходному программисту пришлось изрядно помучиться. И потратить три или четыре дня.

Отсюда следует неутешительный вывод. Глядя на исходный текст программы, понять сложный алгоритм очень трудно. А быстро понять — невозможно.

Разве это хорошо? Разве можно с этим мириться?

МЕТОД ПРОБ И ОШИБОК: ЧЕМУ УЧИТ ИСТОРИЯ АВИАЦИИ

История авиации — это история замечательных достижений и одновременно история катастроф и человеческих трагедий.

Анализируя неудачу лайнера 737 Max, можно указать два варианта развития событий:

- традиционный подход к исследованию причин катастроф и аварий и исправлению недостатков;
- принципиально новый подход к безошибочности, опирающийся на Комплексную программу уменьшения числа ошибок.

Первый вариант ведет к успеху методом проб и ошибок, где проба — это очередная катастрофа, причем платить приходится жизнями людей.

Второй вариант нацелен на поиск более рационального решения. При этом важную роль играет новая нотация, специально предназначенная для создания безошибочных алгоритмов.

НОТАЦИЯ БЕЗОШИБОЧНОГО АЛГОРИТМА

Нотация алгоритма — система обозначений, позволяющая записать, прочитать и понять алгоритм.

При обычном подходе к выбору нотации требование об отсутствии ошибок не ставится и не рассматривается. Такой подход уязвим для критики, ибо позволяет использовать для записи алгоритмов любую подходящую нотацию, например:

- естественный язык (включая словесно-формульную запись);
- псевдокод;
- язык программирования;
- блок-схему согласно ГОСТ 19.701-90 (ISO 5807-85);
- схему деятельности (activity diagram) языка UML;
- диаграмму Несси-Шнейдермана и т. д.

По нашему мнению, все нотации из этого списка имеют существенные недостатки. Они не обладают свойством эргономичности и не могут использоваться для безошибочных алгоритмов.

Новый способ записи должен быть эргономичным и значительно более эффективным. Новая нотация способна обеспечить почти полное отсутствие ошибок. При этом используется батарея новых методов подавления ошибок, изложенная в следующих главах.

МОГЛИ ЛИ БЕЗОШИБОЧНЫЕ АЛГОРИТМЫ И ЖИЗНЕРИТМЫ СПАСТИ САМОЛЕТ 737 MAX?

Мы склонны дать положительный ответ: да, безошибочные алгоритмы могли предотвратить катастрофу и спасти от гибели 346 человек.

МОЖНО ЛИ СОЗДАТЬ АЛГОРИТМИЧЕСКИЙ ЯЗЫК, СПОСОБНЫЙ ПРЕДОТВРАЩАТЬ ОШИБКИ

Наша цель — ***облегчить и ускорить понимание алгоритмов и выявление ошибок.***

Для этого необходимо устранить или ослабить когнитивные затруднения, то есть трудности понимания алгоритмов. Когнитивно-эргономический подход — это рабочий метод, дающий полезные плоды — улучшение понимаемости алгоритмов и программ, повышение производительности сложного интеллектуального труда. Мы постарались обосновать этот тезис, постепенно раскрывая особенности языка, способного предотвращать появление ошибок. В данной книге показано, что указанным требованиям удовлетворяет язык ДРАКОН.

Как и все прочие языки, ДРАКОН опирается на математику и логику. Однако сверх того, он самым тщательным образом учитывает когнитивные вопросы. Благодаря систематическому использованию когнитивно-эргономических методов ДРАКОН приобрел уникальные эргономические характеристики. Можно предположить, что в будущем он сможет претендовать на звание чемпиона по критерию «понимаемость алгоритмов и программ» (в классе императивных языков).

ДРАКОН должен удовлетворять требованию безошибочности алгоритмов. Такое требование предъявляется к языку впервые. Оно символизирует принципиально новый подход к разработке алгоритмического языка.

ДРАКОН можно определить как общедоступный визуальный язык, предназначенный для описания структуры деятельности, для систематизации, структуризации, наглядного представления и формализации императивных знаний, а также для проектирования, программирования, моделирования и обучения. Это универсальный межотраслевой язык делового мира, служащий для описания научно-технических, медицинских, биологических, экономических, социальных, учебных и иных задач. ДРАКОН позволяет упорядочить и представить решение любой, сколь угодно сложной императивной (процедурной, деятельностной, технологической, рецептурной, алгоритмической) проблемы в виде наглядных чертежей, выполненных по принципу «Взглянул — и сразу понял!»

Человечность языка ДРАКОН, стремление создать максимальный комфорт для работы человеческого мозга, всемерная забота о повышении творческой продуктивности персонала позволяет надеяться, что со временем он получит широкое применение в народном хозяйстве, бизнесе, обороне, науке и системе образования.

Используя не просто наглядные, а предельно наглядные формы представления знаний, облегчая работу мозга, ДРАКОН обеспечивает заметный рост производительности интеллектуального труда.

В основе языка ДРАКОН лежит идея когнитивной формализации знаний, позволяющая сочетать строгость логико-математической формализации с точным учетом когнитивных (познавательных) характеристик человека. Это дает возможность значительно упростить и облегчить процедуру описания структуры деятельности, формализацию профессиональных знаний специалистов, стандартизовать ее и сделать пригодной для массового практического использования. Это в равной степени касается как компьютерной, так и «бескомпьютерной» интеллектуальной деятельности людей.

Предполагается, что безошибочность языка ДРАКОН со временем позволит обеспечить качественный скачок в повышении продуктивности сложного интеллектуального труда, выявление и более полное использование резервов человеческого интеллекта, создания когнитивных предпосылок для существенного повышения эффективности информационных технологий [44, pp. 31, 32].

ТРУДНОСТИ И СРЕДА РАЗРАБОТКИ (IDE)

Принято различать и противопоставлять текстовое (textual) и визуальное (visual) программирование. Первое является доминирующим, оно безраздельно господствует на рынке программных продуктов. Второе носит в основном экспериментальный характер и занимает ничтожно малую долю рынка. Однако визуальное направление имеет многообещающий потенциал и в будущем может занять лидирующие позиции.

В этой области наблюдается путаница с терминологией. Некоторые языки именуются визуальными (например, Visual Basic), но на самом деле предназначены для текстового программирования.

Трудности на пути визуального программирования связаны со средой разработки программ. Интегрированная среда разработки (Integrated development environment — IDE) является важным инструментом программистов. Среды IDE и смежные инструменты хорошо развиты для текстового программирования и слабо представлены для визуального.

Причины понятны, так как текстовое программирование появилось намного раньше визуального. На начальном этапе программы были исключительно текстовыми, что позволило накопить богатый опыт текстового кодирования. Поскольку разработка IDE для текстовых программ ведется давно, удалось создать разнообразные и удобные средства и инструменты. В разработку «текстовых» IDE крупнейшие компании и лидеры рынка вложили огромный труд и огромные деньги. Для визуального программирования дело обстоит намного хуже.

ДРАКОН — алгоритмический язык визуального программирования и моделирования. Указанные выше объективные трудности «визуального развития» в полной мере относятся и к ДРАКОНу.

ВЫВОДЫ

1. Низкая понимаемость алгоритмов и программ — важный недостаток современного программирования, способствующий появлению и распространению ошибок.
2. Глядя на исходный текст программы, понять сложный алгоритм трудно. А быстро понять — невозможно.
3. Понимаемость есть свойство удобочитаемости алгоритма, позволяющее минимизировать интеллектуальные усилия, необходимые:
    - для понимания алгоритма при зрительном восприятии человеком;
    - для легкого и быстрого выявления ошибок.
4. Актуальной задачей является разработка теории безошибочных алгоритмов.
5. Нотация алгоритма — система обозначений, позволяющая записать, прочитать и понять алгоритм.
6. Существующие нотации не пригодны для создания безошибочных алгоритмов, ибо не обладают свойством эргономичности.
7. Для массового производства безошибочных алгоритмов нужна эргономичная нотация, обеспечивающая высокую понимаемость алгоритмов и удовлетворяющая требованиям когнитивной эргономики.
8. Макроалгоритмы делятся на компьютерные алгоритмы и жизнеритмы, выполняемые людьми.
9. Жизнеритмы — это бизнес-процессы, описывающие функционирование коммерческих фирм и государственных учреждений, а также взаимодействие между ними.
10. Системный подход к анализу макроалгоритмов требует описать алгоритмы и жизнеритмы как единой взаимосвязанной системы.

11. Совместное использование эргономичных и понятных алгоритмов и жизнеритмов позволяет выявить ошибки и упущения в макроалгоритмах.
12. Недооценка важности эргономичности и понимаемости макроалгоритмов может привести к негативным последствиям и к катастрофе, как показала трагедия самолета Боинг 737 Max.

Глава 31

ЯЗЫК ДРАКОН ПОМОГАЕТ ПРОГРАММИРОВАТЬ БЕЗ ОШИБОК

ЯЗЫК ДРАКОН И ПОДАВЛЕНИЕ ОШИБОК

В языке ДРАКОН используется оригинальный метод под названием «подавление ошибок». В него входят четыре элемента (рис. 156):

- эргономичная нотация;
- алгоритмическая логика;
- исчисление икон;
- интеллектуальный «ДРАКОН-конструктор».

Рис. 156. Структурная схема обеспечения безошибочности языка ДРАКОН

ЭРГОНОМИЧНАЯ НОТАЦИЯ

Разработка эргономичной нотации для записи безошибочных алгоритмов ведется по двум направлениям: формальному и неформальному (рис. 157).

Формальное направление связано с созданием языка ДРАКОН и охватывает:

- графический и текстовый алфавит;
- графический и текстовый синтаксис;
- семейство дракон-языков.

Неформальное направление затрагивает вопросы:

- когнитивно-эргономические правила;
- правила шампура;
- правила семейного сходства;
- правила выравнивания.

Сюда же можно добавить картографический принцип силуэта и примитива.

Общая цель формального и неформального подходов — обеспечить безошибочность дракон-алгоритмов за счет строгого порядка в визуальной картине алгоритма.

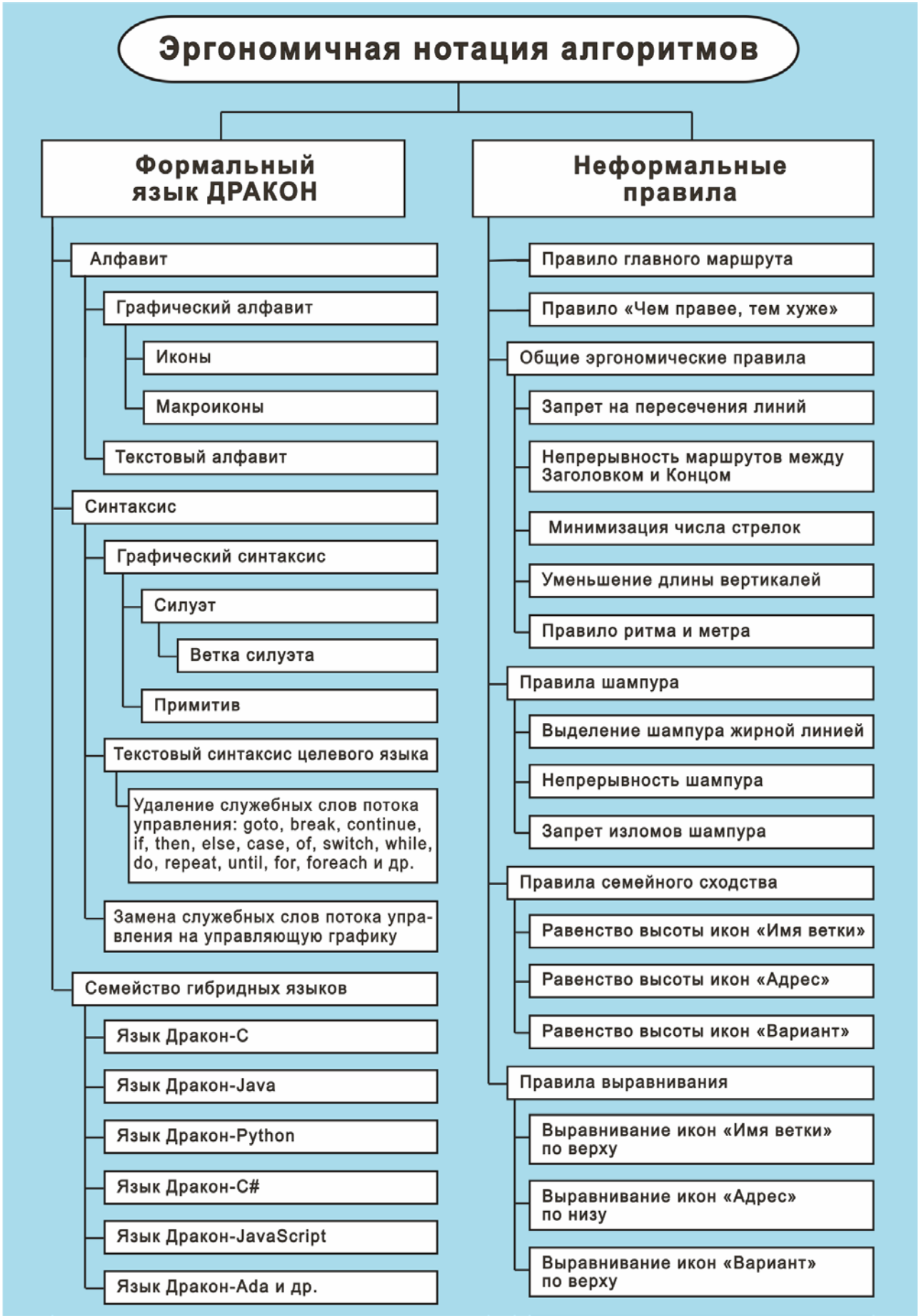


Рис. 157. Структурная схема эргономичной нотации языка ДРАКОН

ГРАФИЧЕСКИЙ И ТЕКСТОВЫЙ СИНТАКСИС ЯЗЫКА ДРАКОН

ДРАКОН — визуальный язык, в котором используются два типа элементов:

- графические фигуры (иконы и макроиконы);
- текстовые надписи, расположенные внутри и снаружи икон.

По этой причине язык ДРАКОН имеет не один, а два синтаксиса: графический и текстовый.

Графический синтаксис охватывает алфавит икон, правила их размещения на чертеже и правила связи икон с помощью соединительных линий.

Текстовый синтаксис задает алфавит символов, правила их комбинирования и привязку к иконам. Привязка необходима потому, что внутри разных икон используются различные типы выражений.

СЕМЕЙСТВО ДРАКОН-ЯЗЫКОВ

ДРАКОН — не один язык, а целое семейство, которое может включать неограниченное число ДРАКОН-языков. В состав семейства входит универсальный визуальный алгоритмический язык (являющийся языком моделирования, а не программирования), а также гибридные языки программирования.

Императивную (процедурную) часть языка ДРАКОН можно присоединить к некоторым языкам программирования и получить гибридные языки, например:

язык Дракон + язык C	=	гибридный язык Дракон-C
язык Дракон + язык Java	=	гибридный язык Дракон-Java
язык Дракон + язык C#	=	гибридный язык Дракон-C#
язык Дракон + язык Python	=	гибридный язык Дракон-Python
язык Дракон + язык JavaScript	=	гибридный язык Дракон JavaScript
язык Дракон + язык Lua	=	гибридный язык Дракон-Lua
язык Дракон + язык Erlang	=	гибридный язык Дракон-Erlang
язык Дракон + язык Ada	=	гибридный язык Дракон-Ada
язык Дракон + язык Oberon	=	гибридный язык Дракон-Oberon
язык Дракон + язык Tcl	=	гибридный язык Дракон-Tcl
и т. д.		

КАК ПОСТРОИТЬ ГИБРИДНЫЙ ЯЗЫК ДРАКОН-СИ

Чтобы построить язык Дракон-Си, надо по определенным правилам соединить графический синтаксис ДРАКОНа с текстовым синтаксисом языка Си. При этом Си рассматривается как целевой язык (target language). Нужно сделать следующее:

- использовать синтаксис целевого языка (синтаксис языка Си) в качестве текстового синтаксиса гибридного языка Дракон-Си;
- удалить из текстового синтаксиса гибридного языка Дракон-Си все элементы, которые заменяются управляющей графикой ДРАКОНа;
- создать транслятор из дракон-схемы в исходный код языка Си.

Любой гибридный язык (например, Дракон-Си) почти полностью сохраняет концепцию, структуру, типы данных и другие особенности целевого языка (Си). При этом в строго определенном числе случаев текстовая нотация целевого языка заменяется на визуальную. Такой прием позволяет существенно улучшить эргономический облик языка и сократить число ошибок.

При использовании гибридных языков исходным текстом программы считается дракон-схема и только она [43, pp. 263-266].

ЕДИНСТВО И РАЗНООБРАЗИЕ

Все языки ДРАКОН-семейства имеют единый графический синтаксис. Это обеспечивает зрительное сходство дракон-схем различных дракон-языков.

Каждый язык семейства отличается тем, что имеет свой собственный текстовый синтаксис. Строгое разграничение графического и текстового синтаксиса позволяет в максимальной степени расширить сферу применения языков семейства, обеспечивая гибкость и универсальность выразительных средств языка.

При этом единообразие правил графического синтаксиса дракон-языков обеспечивает их концептуальное единство. Разнообразие текстовых правил (то есть возможность выбора любого текстового синтаксиса) определяет гибкость языка и богатство выразительных средств [43, p. 263].

БЕЗОШИБОЧНОСТЬ ПРИ ОПИСАНИИ ПОТОКА УПРАВЛЕНИЯ

На рис. 157 имеется пункт: «Замена служебных слов потока управления на управляющую графику». Это важный пункт, который нуждается в подробном разъяснении.

Покажем, что потенциально опасные *текстовые* средства потока управления можно заменить на безопасные *визуальные* средства управления.

Начнем с истории. В 1968 году Эдсгер Дейкстра в журнале «Communications of the ACM» указал, что оператор *goto,* используемый во многих языках программирования высокого уровня, является основным источником ошибок и потому должен быть запрещен [54]. Затем Бертран Мейер выявил еще два опасных элемента — операторы *break* и *continue,* который также следует запретить как замаскированные *goto.* По словам Мейера, это те же старые «*goto* в овечьей шкуре» [55, p. 210].

Еще дальше идет И. В. Вельбицкий, который считает, что из программирования следует исключить

> «ключевые слова-паразиты и соответствующие им конструкции языков типа: *goto, if, for, while, break*, *begin-end, {-}* и т.д… Эти конструкции являются основным источником ошибок и проблем в современном программировании» [56].

Язык ДРАКОН продолжает и развивает традицию, начатую Дейкстрой и продолженную Мейером и Вельбицким. Традицию, направленную на выявление и изгнание потенциально опасных операторов управления, которые могут послужить причиной ошибок.

В ЧЕМ ИДЕЯ

Идея проста и сводится к двум положениям:

- надо выявить опасные служебные слова и знаки, используемые в операторах управления, и запретить их;
- вместо них следует использовать безопасную графику.

Приведем список исключенных из языка ДРАКОН служебных слов, обеспечивающих управление вычислительным процессом: *goto, break, continue, if, then, else, case, of, switch, while, do, repeat, until, for, foreach, loop, exit, when, last* и их аналоги. Все они подлежат замене на математически строгую графику управления. Графика реализует ту же самую функцию, что и забракованные нами операторы и служебные слова.

Замена текстовых операторов на их точные графические эквиваленты означает, что язык ДРАКОН использует двумерное (2d) структурное программирование [39, pp. 425-472]. Последнее можно рассматривать как дальнейшее развитие одномерного (1d) структурного программирования, которое создали Эдсгер Дейкстра, Тони Хоар, Оле-Йохан Дал [57] и др.

ОПАСНЫЙ КАТАЛИЗАТОР ОШИБОК

Отрицательную роль перечисленных текстовых операторов управления можно охарактеризовать как «опасный катализатор ошибок», ибо они:

- отвлекают внимание программиста, затрудняют понимание смысла программы, делают работу программиста трудной;
- засоряют визуальную картину программы, создают визуальные помехи, фактически являясь ненужными, паразитными элементами, без которых можно обойтись;
- провоцируют появление ошибок.

Цель заключается в том, чтобы улучшить способность языков проектирования и программирования ***предупреждать появление ошибок.*** Нужно существенно увеличить надежность программно-математического обеспечения. Ниже показано, что операторы, провоцирующие ошибки, можно обезопасить.

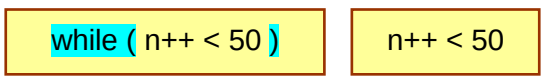
ОПАСНЫЙ КАТАЛИЗАТОР ОШИБОК В ЦИКЛЕ *WHILE*

На рис. 158 слева показана си-программа с циклом *while,* справа — эквивалентная дракон-программа.

В си-программе имеются избыточные элементы, которые можно изъять с целью улучшения исходного кода программы. Вот они:

- *while*;
- две круглые скобки;
- две фигурные скобки.

Сравним два оператора (рис. 158):



Си-оператор в левой рамке содержит загромождающие паразитные элементы, отсутствующие в дракон-операторе справа. Следствием является неоправданное усложнение си-оператора, которое засоряет программу и отвлекает внимание программиста.

В дракон-программе избыточные (паразитные) знаки не нужны. Вместо них используются линии формального чертежа. Эргономичный чертеж гораздо лучше, чем текст, показывает маршруты алгоритма, а также петлю обратной связи и тело цикла.

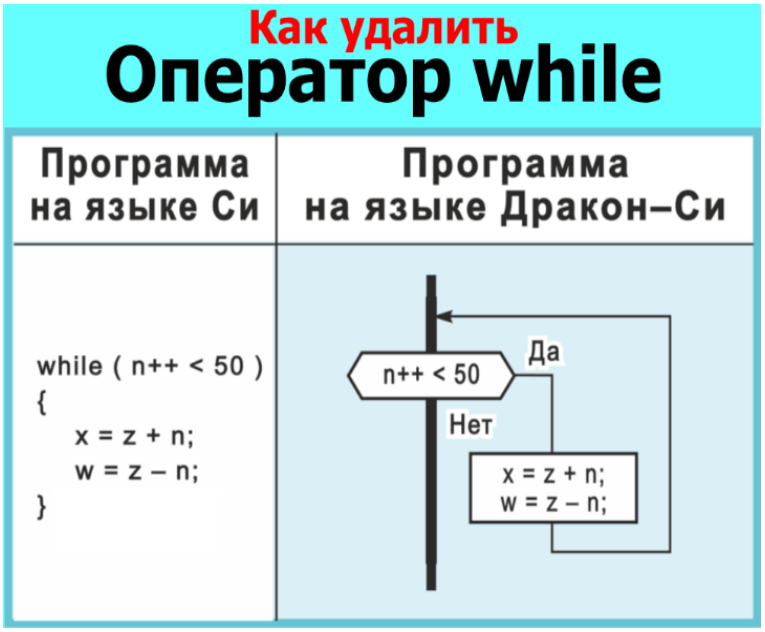


Рис. 158. Оператор *while* можно удалить и заменить на управляющую графику языка ДРАКОН

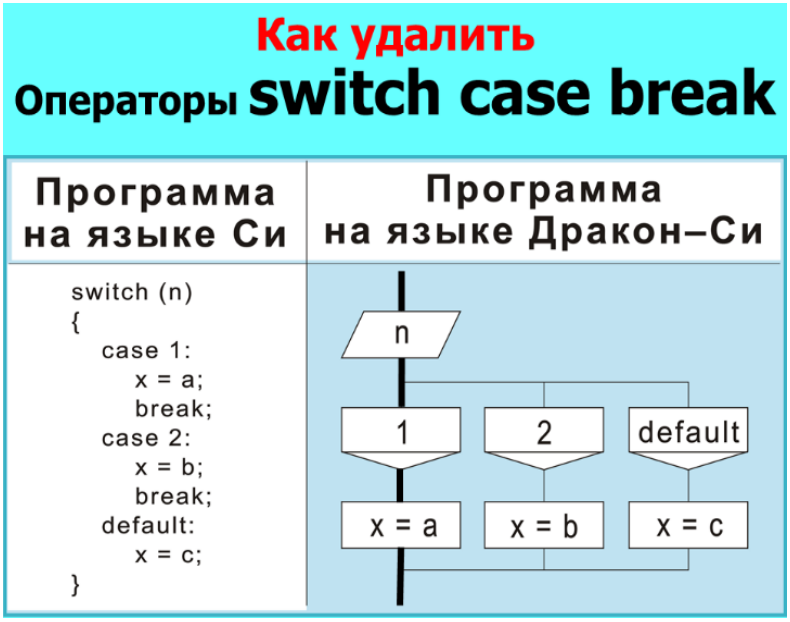


Рис. 159. Операторы *switch, case, break* можно удалить и заменить на управляющую графику языка ДРАКОН

СРАВНЕНИЕ ТЕКСТА И ГРАФИКИ ДЛЯ ОПЕРАТОРОВ *SWITCH, CASE, BREAK*

На рис. 159 слева показана си-программа с операторами *switch, case, break,* а справа — эквивалентная дракон-программа. В си-программе используется большое количество ненужных ключевых слов и знаков пунктуации:

- *switch,*
- *case* (два раза),
- *break* (два раза),
- две фигурные скобки,
- две круглые скобки,
- три двоеточия,
- пять точек с запятой.

Указанные слова и знаки для языка Си являются обязательными элементами исходного текста программы. Обойтись без них невозможно, они должны соответствовать формальным правилам синтаксиса языка Си, которые проверяются при компиляции в ходе лексического и синтаксического анализа.

С точки зрения языка ДРАКОН, дело обстоит принципиально по-другому. То, что важно и необходимо для Си, язык ДРАКОН рассматривает как лексический и синтаксический мусор, как слова-пустышки и знаки-паразиты, как бессмысленные, ненужные и вредные элементы, которые подлежат изъятию и удалению. Потому что они являются катализаторами ошибок.

Приведем количественный расчет. В си-программе на рис. 159 использовано 55 символов (characters) без пробелов, а в дракон-программе всего 19, т. е. в 2,9 раза меньше. Благодаря графике дракон-программа легче для восприятия, в ней проще разобраться и заметить ошибку. Дополнительное удобство для чтения программного кода связано с тем, что размещение символов в пространстве подчиняется строгим правилам эргономичной декомпозиции и правилам отделения фигуры от фона.

Результат сравнения

- В дракон-программе на рис. 159 количество символов сокращается почти в три раза, по сравнению с си-программой.
- Лишние символы рассматриваются как мусор и исчезают. Они превращаются в приятный для глаза чертеж, пригодный для быстрого симультанного (панорамного) восприятия.

ВИТАЛИЙ КАУФМАН И КРИТЕРИЙ ДЕЙКСТРЫ

В книге «Языки программирования. Концепции и принципы» В. Ш. Кауфман сформулировал «критерий Дейкстры». Кауфман выявил негативные признаки, затрудняющие программирование:

- провоцирует ошибки,
- засоряет программу,
- отвлекает внимание программиста от существенно более важных проблем правильности и надежности [58].

Нетрудно заметить, что негативные признаки Кауфмана в точности совпадают с описанным выше «опасным катализатором ошибок».

СРАВНЕНИЕ ТЕКСТА И ГРАФИКИ ДЛЯ ОПЕРАТОРА *DO-WHILE*

На рис. 160 рассмотрен цикл *do-while* (цикл ДО). Слева мы видим си-программу, справа дракон-программу.

Как и в предыдущих случаях, в си-программе имеются лишние, избыточные, загромождающие элементы:

- *do;*
- *while;*
- две круглые скобки;
- две фигурные скобки;
- две косые скобки;
- две звездочки.

В дракон-программе эти значки рассматриваются как вредоносные и исчезают за ненадобностью. На чертеже они полностью лишены смысла. Чертеж наглядно демонстрирует структуру цикла, причем делает это намного лучше текста.

Рис. 160. Оператор *do while* можно заменить на управляющую графику

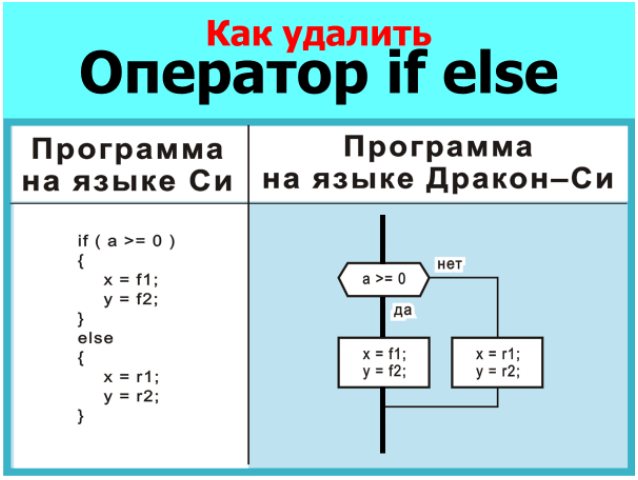


Рис. 161. Оператор *if else* можно заменить на управляющую графику

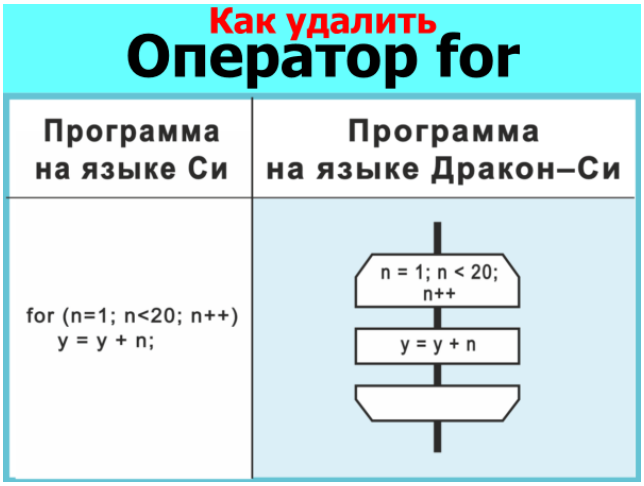


Рис. 162. Оператор *for* можно заменить на управляющую графику

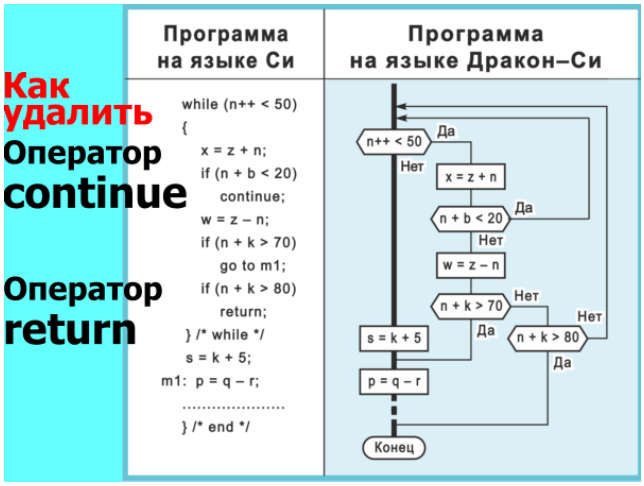


Рис. 163. Операторы *continue* и *return* можно заменить на управляющую графику

ДРУГИЕ ОПЕРАТОРЫ УПРАВЛЕНИЯ

На рис. 161-163 показан еще три примера, подтверждающих возможность удалить управляющие текстовые операторы. И заменить их управляющей графикой языка ДРАКОН. Имеются в виду операторы:

- *if else,*
- *for u foreach,*
- *continue,*
- *return.*

ТЕОРЕМА О СТРУКТУРНОМ ПРОГРАММИРОВАНИИ

Являются ли наши примеры исчерпывающими? Обратимся к теореме Бёма-Якопини [59], которую часто называют теоремой о структурном программировании.

Коррадо Бём и Джузеппе Якопини не употребляли термин «структурное программирование» *(structured programming).* Тем не менее, доказанную ими теорему (и её последующие вариации у разных авторов) впоследствии стали называть именно так — теоремой о структурном программировании *(structured program theorem).*

Представляет интерес доказательство Харлана Миллса, который первым ввел термин «структурная теорема» *(structure theorem)* [60].

Сопоставляя рис. 158-163 и указанную теорему, можно утверждать, что возможность замены управляющих текстовых операторов эквивалентными визуальными аналогами доказана [39, pp. 255-262].

ОБСУЖДЕНИЕ

Обобщая материалы, представленные на рис. 158-163, можно сделать следующие замечания.

- Показанные на рисунках фрагменты программ на языках Си и Дракон-Си математически эквивалентны.
- Хотя с точки зрения математики программы эквивалентны, с точки зрения эргономики они существенно отличаются. Графический образ имеет значительные преимущества перед текстовым.
- Программы на языке Си представляют собой эргономически неудачную попытку описать словами чертеж дракон-схемы. И наоборот, визуальный образ программы на языке Дракон-Си удовлетворяет критерию сверхвысокого понимания, сокращает число ошибок и повышает производительность труда.
- Процедурный текст на языке программирования *X* можно преобразовать в математически эквивалентный графический образ — дракон-программу на гибридном языке программирования Дракон-*X*.

С эргономической точки зрения, служебные слова и многие символы, присутствующие в текстовых языках, есть не что иное как визуальный шум (визуальные помехи). Они назойливо притягивают к себе внимание читателя, отвлекая его внимание от содержательной стороны дела, мешая ему продуктивно мыслить.

Эргономическое преимущество ДРАКОНа состоит в том, что вместо служебных слов используется визуальный образ, который воспринимается читателем бессознательно, на интуитивном уровне. За счет этого канал сознательного внимания действует более продуктивно — для восприятия наиболее важных, содержательных аспектов задачи.

БЕЗОШИБОЧНОСТЬ В ОПЕРАТОРАХ УПРАВЛЕНИЯ ВЫЧИСЛИТЕЛЬНЫМ ПРОЦЕССОМ

Ошибки в операторах управления очень неприятны. Предположим, что нам удалось избавиться от них. Что это даст?

Все циклы и разветвления станут безошибочными. Ошибки могут сохраниться только на линейных участках программы, где их можно легко выявить и устранить.

Язык ДРАКОН стремится (в пределах возможного) приблизиться к этому идеалу. Чтобы предотвратить ошибки в циклах и разветвлениях, в языке ДРАКОН предусмотрен ряд способов, указанных на рис. 156. Эргономичную нотацию мы уже рассмотрели. Перейдем к следующему способу — алгоритмической логике.

АЛГОРИТМИЧЕСКАЯ ЛОГИКА И БЕЗОШИБОЧНОСТЬ

Вспомним еще раз мудрые предостережения Эдварда Йодана и Гленфорда Майерса:

> «Если это возможно, избегайте отрицаний в булевых выражениях. Представляется, что их понимание представляет трудность для многих программистов… Избегайте без нужды использования сложных булевых выражений… Даже если нет неоднозначностей, такие выражения обычно с трудом понимают все, за исключением их автора» [10]. «Распространенным источником ошибок является использование логических операций И и ИЛИ» [11].

Чтобы обеспечить логическую безошибочность, надо отказаться от логических связок, как предписано в главе 15.

Краткий путеводитель по алгоритмической логике, описанной в Части 3, представлен на рис. 164. Рекомендуется обратить внимание на следующие пункты:

- алгоритмическая схема «И»;
- алгоритмическая схема «ИЛИ»;
- алгоритмическая схема «НЕ»;
- удаление пропозициональных связок;
- приведение к каноническому виду;
- равносильные преобразования визуальных алгоритмов.

Алгоритмическая логика позволяет существенно улучшить безошибочность, используя удаление пропозициональных связок и приведение к каноническому виду.

ПОСТУЛАТ ДРАКОНА И ТЕОРЕМА БОСУЭЛЛА И ФАУЧЕРА

Чем больше сложность, тем больше путаницы, тем больше ошибок. По мнению экспертов, «поведение – самая сложная часть программы и поэтому источник самых запутанных ошибок» [61].

Постулат языка ДРАКОН	Дракон-алгоритм должен быть эргономичным, то есть нарисован так, чтобы можно было максимально быстро понять, как он работает

Сходную мысль высказывают Дастин Босуэлл и Тревор Фаучер в книге «Читаемый код» [62].

Босуэлл и Фаучер считают удобочитаемость программного кода важным параметром. Они дали своей идее образное название «фундаментальная теорема читаемости». Вот ее текст:

> «Код должен быть написан так, чтобы можно было максимально быстро понять, как он работает» [62, p. 21].

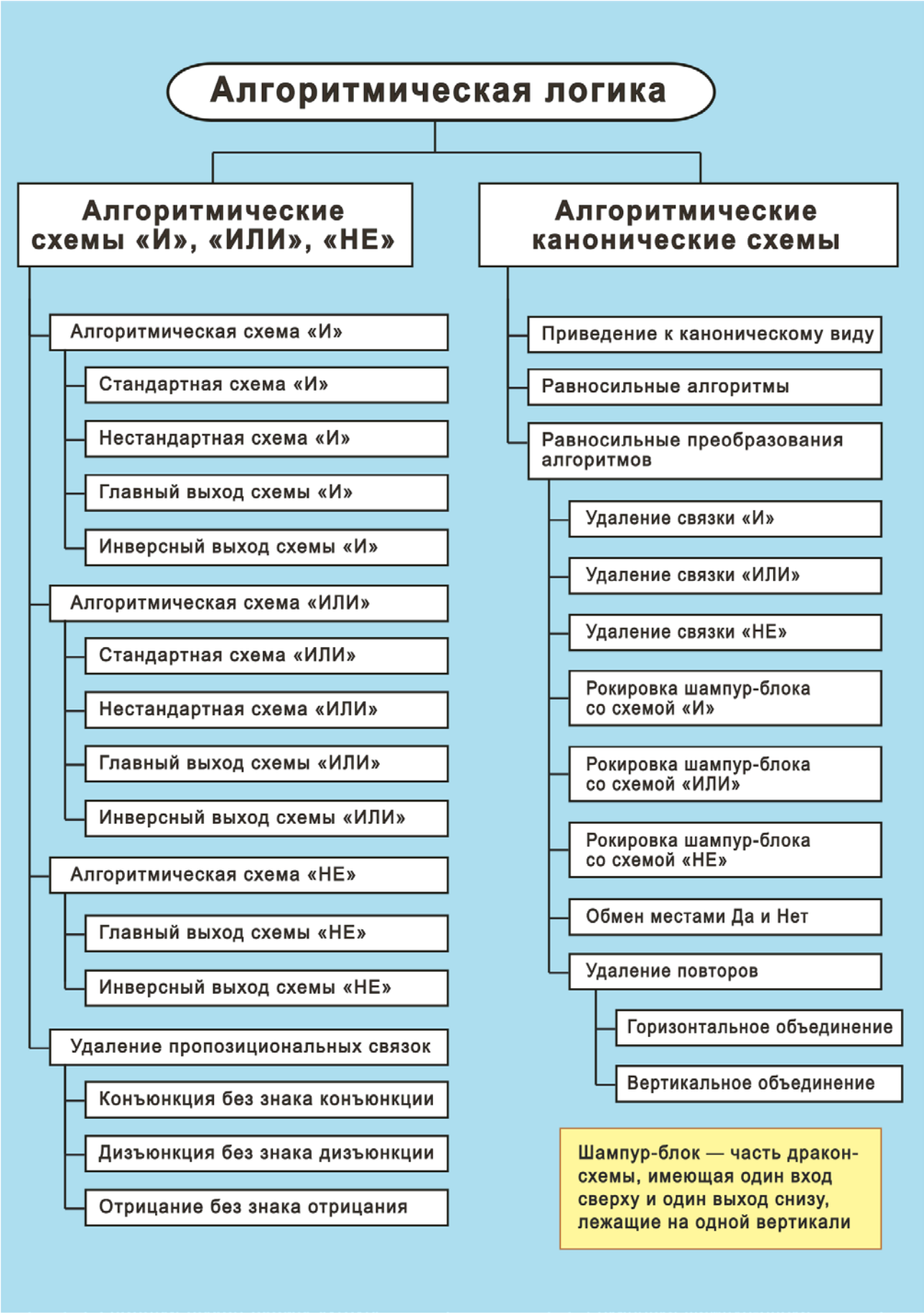


Рис. 164. Структурная схема алгоритмической логики языка ДРАКОН

Речь идет о глубоком и быстром понимании. Акцент делается на словах «максимально быстро понять». Авторы поясняют:

> «Время-для-понимания — это теоретический параметр, который вам нужно уменьшить. Обратите внимание, что, говоря «понимание», мы устанавливаем планку очень высоко. *Полное понимание* кода означает возможность вносить в него изменения, находить ошибки, а также понимать, как именно он взаимодействует с остальными фрагментами кода» [62, p. 21].

Чем интересна книга «Читаемый код»? Она устанавливает закономерную связь между удобочитаемостью, пониманием и ошибками. Чем лучше читаемость программного кода, тем глубже понимание кода, тем меньше ошибок. Это значит, что удобочитаемость есть средство защиты от ошибок. Босуэлл и Фаучер утверждают: тот, кто научится писать код, обеспечивающий быстрое понимание, «будет допускать меньше ошибок» [62, p. 22].

Легко заметить, что постулат языка ДРАКОН и теорема Босуэлла и Фаучера почти полностью совпадают.

ДРАКОН ИГРАЕТ РОЛЬ ЗАЩИТНОГО ФИЛЬТРА

Применяя гибридный язык, например, Дракон-Си, пользователь начинает работу с дракон-схемой и получает все преимущества языка ДРАКОН, связанные с безошибочностью.

Известно, что Си — небезопасный язык. По мнению экспертов, «не вызывает сомнений, что Си – изобилующий потенциальными опасностями и не вполне прозрачный для восприятия человеком язык» [63]. Суть в том, что ДРАКОН сразу исключает многие ошибки, связанные с потоком управления, логикой и условными операторами. ДРАКОН (в составе гибридного языка) служит «входной дверью» в язык Си. Такая «дверь» устраняет часть ошибок, присущих Си и тем самым играет роль защитного фильтра, не пропускающего ошибки. Свойство защиты относится не только к Си, но и к любому целевому гибридному языку, входящему в ДРАКОН-семейство.

Как это выглядит на практике? Послушаем рассказ Сергея Ефанова.

«Некоторое время назад мне на глаза попалось упоминание о языке ДРАКОН. Я немного почитал, ничего не понял, закрыл, забыл.

Но почему-то забылось не совсем.

Тут подвернулась поездка. Снова нашёл, закачал файл в электронную книжку, взял с собой. В поезде всё равно делать нечего.

Медленно и со вкусом прочитал.

И — понял! Это просто клад!

По возвращении уже думал только об одном: где бы найти инструмент для работы?

К счастью, такой инструмент нашелся [69].

Попробовал небольшие примерчики — вроде какой-то код генерируется. Переписал на ДРАКОНе довольно запутанную функцию из реального проекта.

Функция заработала сразу! Более того, при переносе алгоритма в дракон-схему, я обнаружил, что у меня в ней была ошибка! Эта функция работала уже довольно давно, не в одной сотне изделий. Ошибка не была фатальной, она возникала редко, и компенсировалась переподключением к серверу. Но она была!

В тексте на языке Си она была незаметной. А при попытке перенести алгоритм на дракон-схему, ошибка стала не просто заметной — алгоритм в этом месте «не вырисовывался»! [101].

ВЫВОДЫ

1. Одна из целей языка ДРАКОН — обеспечить безошибочность, сократить число ошибок в алгоритмах и программах.
2. Для этого используется метод «подавление ошибок», куда входят:
   - эргономичная нотация;
   - алгоритмическая логика;
   - исчисление икон;
   - интеллектуальный «ДРАКОН-конструктор».
3. В эргономичной нотации текстовые операторы управления вычислительным процессом заменяются на управляющую графику.
4. Вероятность ошибок в визуальных циклах и разветвлениях значительно снижается. Ошибки могут сохраниться только на линейных участках программы.
5. Язык ДРАКОН можно присоединить к некоторым языкам программирования и получить гибридные языки, например: Дракон-Си, Дракон-Java и др.
6. Чтобы построить язык Дракон-Си, надо соединить графический синтаксис Дракона с текстовым синтаксисом языка Си. Из последнего нужно убрать все операторы управления, функции которых выполняет управляющая графика Дракона.
7. Операторы присваивания, логические выражения и т. д. переносятся в визуальную программу и размещаются внутри ее икон. Другие элементы языка Си (которые можно назвать удаляемыми) становятся ненужными. Они превращаются в графические линии и служебные слова «Да» и «Нет».
8. Гибридный язык почти полностью сохраняет концепцию, структуру, типы данных и другие особенности целевого языка.
9. Показанные на рисунках фрагменты программ на языках Си и Дракон-Си эквивалентны.
10. Хотя с точки зрения математики программы эквивалентны, но с точки зрения эргономики они существенно отличаются. Графический образ легче воспринимается, чем текст.
11. В си-программе имеются служебные слова потока управления, различные скобки и другие знаки. В дракон-программе они отсутствуют, превращаясь в чертеж. Чертеж отчетливо показывает пространственную структуру программы. В итоге зрительное восприятие улучшается, что способствует выявлению ошибок.
12. Императивный текст на языке программирования X можно преобразовать в математически эквивалентный графический образ — дракон-программу на гибридном языке программирования Дракон-X.

Глава 32

ИСЧИСЛЕНИЕ ИКОН — НОВЫЙ МЕТОД ПРЕДОТВРАЩЕНИЯ ОШИБОК

ВВЕДЕНИЕ

В прошлой главе мы показали, что — в целях борьбы с ошибками — текстовые операторы управления можно заменить на управляющую графику. Но ведь графика тоже может содержать ошибки! Чтобы этого не случилось, необходимо придумать способ, позволяющий создавать безошибочную (или почти безошибочную) графику.

Таким способом является визуальное логическое исчисление, которое называется «исчисление икон» [39, pp. 427-435].

А.Н. Степанов в «Курсе информатики для студентов информационно-математических специальностей» (2018) отмечает:

> «Обсуждаемый подход… был развит в процессе создания отечественного графического языка программирования ДРАКОН… Чтобы получить полноценный язык программирования, необходимо было создать математически строгий метод формализации… Для решения этой задачи был разработан специальный математический аппарат — визуальное логическое исчисление иконок, который стал теоретической основой языка ДРАКОН» [15, pp. 1017-1019].

СВЯЗЬ С МАТЕМАТИЧЕСКОЙ ЛОГИКОЙ

Эта глава посвящена связи между языком ДРАКОН и математической логикой:

- графический синтаксис языка ДРАКОН опирается на идеи математической логики;
- исчисление икон — это раздел математической логики;
- внутренние алгоритмы «ДРАКОН-конструктора» реализуют исчисление икон, то есть опираются на положения математической логики;
- математическую логику можно рассматривать как одно из теоретических оснований языка ДРАКОН.

ОБЩЕИЗВЕСТНЫЕ СВЕДЕНИЯ О МАТЕМАТИЧЕСКОЙ ЛОГИКЕ

Принципиальным достижением математической логики является разработка аксиоматического метода, который характеризуется тремя чертами:

- явной формулировкой исходных положений (аксиом) развиваемой теории (формальной системы);
- явной формулировкой правил логического вывода, с помощью которых из аксиом выводятся теоремы теории;
- использованием формальных языков для изложения теорем рассматриваемой теории [64].

Основным объектом изучения в математической логике являются логические исчисления. В понятие исчисления входят:

а) формальный язык, который задается с помощью алфавита и синтаксиса,
б) аксиомы,
в) правила вывода [64].

Таким образом, исчисление позволяет, зная аксиомы и правила вывода, получить (то есть вывести, доказать) все теоремы теории. Причем теоремы, как и аксиомы, записываются только на формальном языке.

Напомним, что в рамках математической логики следующие термины можно рассматривать как синонимы:

- логическое исчисление,
- формальная система
- теория.

Следовательно, теоремы исчисления, теоремы формальной системы и теоремы теории — одно и то же.

ШАМПУР-СХЕМА

Для дальнейших рассуждений нам понадобится понятие «шампур-схема».

Представим себе дракон-схему:

1) из которой полностью удален текст;
2) в которой описана последовательность выполнения веток с помощью надписей в иконах «Имя ветки» и «Адрес», как показано на рис. 165.

Шампур-схема — это дракон-схема, свободная от текста, в которую добавлены надписи «Имя ветки» и «Адрес».

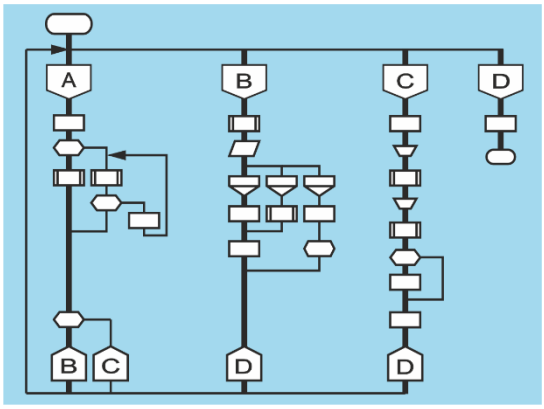


Рис. 165. Шампур-схема

ВИЗУАЛИЗАЦИЯ ПОНЯТИЙ МАТЕМАТИЧЕСКОЙ ЛОГИКИ

Определим два понятия:

- *визуальный логический вывод* (для краткости — видеовывод);
- *визуальное логическое исчисление* (для краткости — видеоисчисление).

Чтобы облегчить изучение материала, используем метод «очной ставки». Поместим в левой графе таблицы 1 хорошо известное «текстовое» понятие. А в правой — определение нового, визуального понятия.

Нетрудно заметить, что новое определение (справа) почти дословно совпадает с классическим (слева). Разница состоит лишь в добавлении приставки «видео».

Таблица 1

Определение понятия «логический вывод»	**Определение понятия «видеовывод» (визуальный логический вывод)**
Вывод в исчислении V есть последовательность C_1, ... , C_n формул, такая, что для любого i формула C_i есть ● либо аксиома исчисления V; ● либо непосредственное следствие предыдущих формул по одному из правил вывода. Формула C_n называется теоремой исчисления V, если существует вывод в V, в котором последней формулой является C_n [8, pp. 101, 285].	Видеовывод в видеоисчислении V есть последовательность C_1, ... C_n видеоформул, такая, что для любого i видеоформула C_i есть ● либо аксиома видеоисчисления V; ● либо непосредственное следствие предыдущих видеоформул по одному из правил видеовывода. Видеоформула C_n называется теоремой видеоисчисления V, если существует видеовывод в V, в котором последней видеоформулой является C_n

Развивая этот подход и опираясь на «текстовое» определение логического исчисления, можно по аналогии ввести понятие «видеоисчисление» (табл. 2).

Таблица 2

Определение понятия «логическое исчисление»	**Определение понятия «видеоисчисление» (визуальное логическое исчисление)**
Логическое исчисление может быть представлено как формальная система в виде четверки $V = < И, S_0, A, F >$ ● $И$ — множество базовых элементов (букв алфавита); ● S_0 — множество синтаксических правил, на основе которых из букв строятся правильно построенные формулы; ● A — множество правильно построенных формул, элементы которого называются аксиомами; ● F — правила вывода, которые из множества A позволяют получать новые правильно построенные формулы — теоремы [65].	Видеоисчисление может быть представлено как формальная система в виде четверки $V = < И, S_0, A, F >$ ● $И$ — множество икон (букв графического алфавита); ● S_0 — множество правил графического синтаксиса, на основе которых из икон строятся правильно построенные видеоформулы; ● A — множество правильно построенных видеоформул, элементы которого называются аксиомами; ● F — правила видеовывода, которые из множества A позволяют получать новые правильно построенные видеоформулы — теоремы. (Множество теорем обозначим через T).

ИСЧИСЛЕНИЕ ИКОН

Мы определили нужные понятия визуальной математической логики. С их помощью можно построить исчисление икон.

- Множество икон *И* (букв графического алфавита) показано на рис. 19, 20.
- Множество S_0 правил графического синтаксиса описано в работе [39] (глава 33, тезисы 2—37).
- Множество *A* графических аксиом включает всего два элемента, которые называются *аксиома-силуэт* и *аксиома-примитив* (рис. 166).
- Множество *T*, охватывающее все графические теоремы исчисления *V*, есть не что иное как множество шампур-схем (абстрактных дракон-схем). Заметим, что множество *T* не включает аксиомы, так как последние содержат незаполненные критические валентные точки и, следовательно, эквивалентны пустым операторам. Множество *T* распадается на две части: множество силуэтов T_1 и множество примитивов T_2.
- Множество *F* правил видеовывода также делится на две части F_1 и F_2. Множество F_1 позволяет выводить все теоремы-силуэты, принадлежащие множеству T_1, из единственной аксиомы-силуэт. Оно содержит семь правил вывода: ввод атома, добавление варианта, добавление ветки, пересадка лианы, заземление лианы, боковое присоединение, удаление последней ветки. Правила вывода для силуэта описаны в работе [39] (глава 33).
- Множество F_2 дает возможность выводить все теоремы-примитивы множества T_2 из единственной аксиомы-примитив. Оно содержит пять правил вывода: ввод атома, добавление варианта, пересадка лианы, боковое присоединение, удаление конца примитива. Эти правила описаны в работе [39] (глава 33).

На этом построение исчисления икон заканчивается.

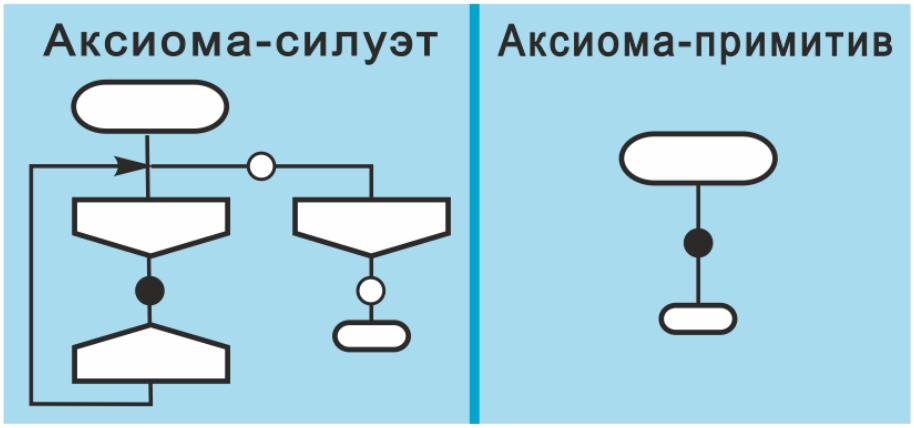


Рис. 166. Аксиома-силуэт и аксиома-примитив.
Черными и белыми кружками отмечены валентные точки

АТОМ

Атом есть графическая фигура, имеющая один вход сверху и один выход снизу, лежащие на одной вертикали. Атомы показаны на рис. 167. Атом можно вставить в валентную точку с помощью операции «ввод атома». Почти все макроиконы (рис. 21, 22) являются атомами.

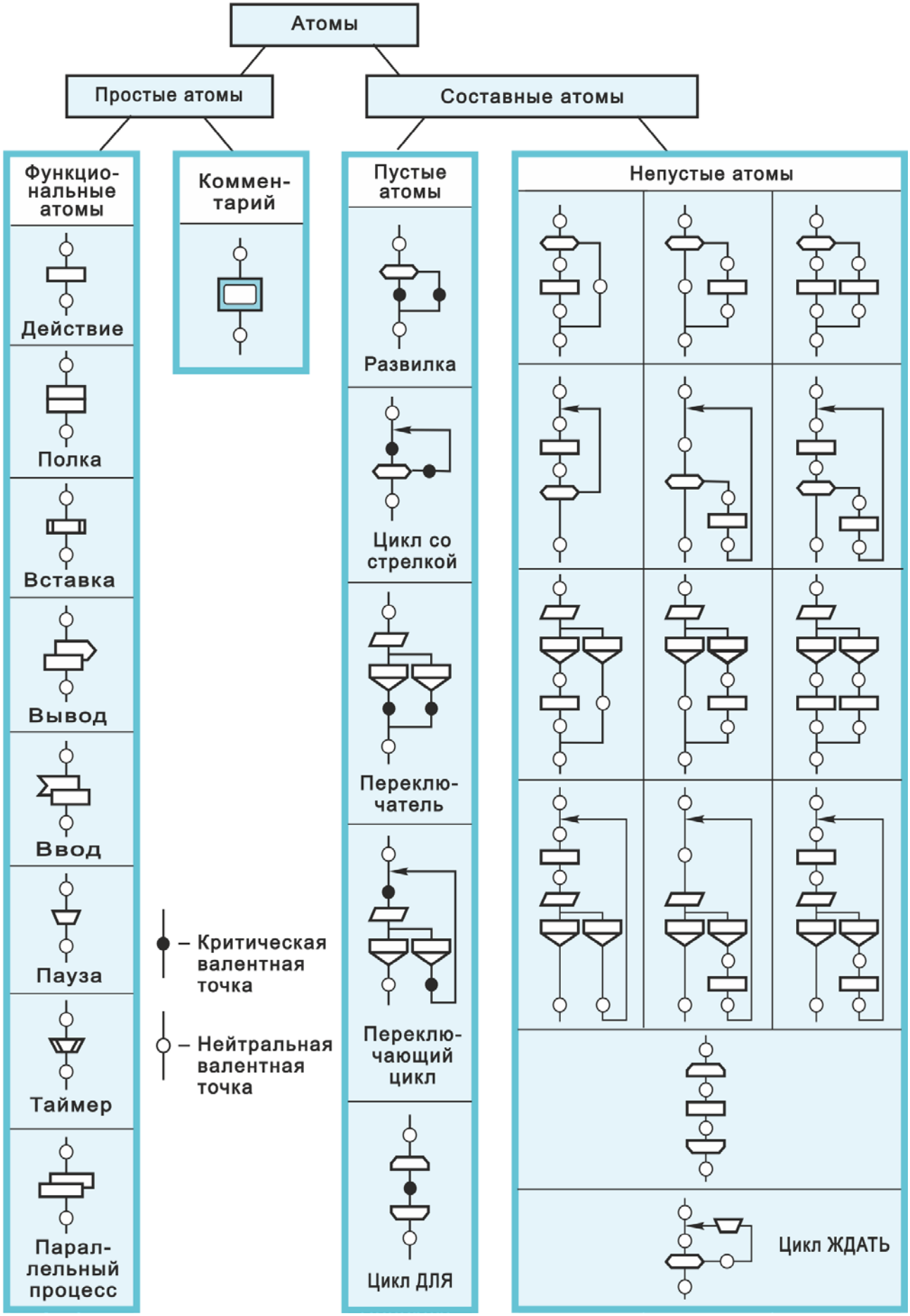


Рис. 167. Атомы и валентные точки

КРИТИЧЕСКИЕ И НЕЙТРАЛЬНЫЕ ВАЛЕНТНЫЕ ТОЧКИ

Валентные точки делятся на нейтральные и критические. Точка называется нейтральной, если операция «ввод атома» в данную точку является возможной, но не обязательной. В отличие от нее критическая точка требует обязательного ввода атома.

Валентные точки находятся в аксиомах и атомах. Они показаны на рис. 166 и 167, где нейтральные точки обозначены белыми кружками, а критические – черными.

Если в фигуре (аксиоме или атоме) всего одна критическая точка, ввод атома обязательно производится именно в нее; при этом критическая точка уничтожается. Если фигура имеет две критические точки, обязательный ввод атома делается только в одну из них; критическая точка, в которую произведен ввод, уничтожается, а другая критическая точка нейтрализуется, т. е. становится нейтральной.

Полная совокупность критических точек охватывает:

- критические точки в пустых атомах;
- одну критическую точку в аксиоме-силуэт;
- одну критическую точку в аксиоме-примитив.

Полная совокупность нейтральных точек охватывает:

- входные и выходные точки атомов;
- две внутренние точки в атоме «цикл ЖДАТЬ»;
- две точки в аксиоме-силуэт;
- точки, полученные в результате нейтрализации критических точек.

Правило критической точки

- В законченной дракон-схеме не должно быть критических точек.
- Наличие критической точки говорит о наличии пустого оператора и является признаком ошибки.

СЕМАНТИКА ШАМПУР-СХЕМ

Известно, что изучение исчислений составляет синтаксическую часть математической логики. Кроме того, последняя занимается семантическим изучением формальных языков, причем основным понятием семантики служит понятие истины [64].

В исчислении икон семантика тривиальна. Различные видеоформулы (шампур-схемы) могут быть истинными или ложными.

Шампур-схема называется истинной, если она выводится из аксиом с помощью правил вывода и не содержит критических точек. И ложной в противном случае.

Таким образом, все правильно построенные шампур-схемы (теоремы) с уничтоженными критическими точками истинны. И наоборот, неправильно построенные схемы, не удовлетворяющие визуальным правилам языка ДРАКОН, считаются ложными.

Примеры ложных схем (содержащих критические точки) показаны на рис. 166.

АЛГОРИТМИЧЕСКИЕ ОШИБКИ

Существуют алгоритмические ошибки, не имеющие видимых признаков, например:

Рис. 168. Структурная схема исчисления икон

- в алгоритме должна быть развилка, но она отсутствует;
- в алгоритме должен быть цикл, но его нет;
- в алгоритме должно быть действие, но оно отсутствует.

Поскольку явные признаки таких ошибок не представлены, постольку исчисление икон не может их выявить. Это означает, что исчисление икон реализует не полную, а частичную проверку графики дракон-алгоритмов.

ВАЛЕНТНЫЕ ТОЧКИ И МАКРОИКОНЫ КАК СРЕДСТВО ПРЕДОТВРАЩЕНИЯ ОШИБОК

Графический синтаксис языка ДРАКОН содержит правила, специально предназначенные для обеспечения безошибочности. Сюда, в частности, относятся правила, касающиеся валентных точек и макроикон; назовем их ВТМ-правила.

Наличие ВТМ-правил создает принципиальное отличие графического синтаксиса языка ДРАКОН от графического синтаксиса других языков.

Вспомним, что ДРАКОН имеет 23 макроиконы (рис. 21, 22). Макроиконы и валентные точки служат для предотвращения ошибок при проведении соединительных линий.

Сравним с блок-схемами алгоритмов и программ по ГОСТ 19.701-90 и ISO 5807:85. В блок-схемах формализованы только фигуры. За пределами формализации остаются:

- точки размещения фигур, или точки ввода фигур;
- связи между фигурами.

Точки и связи остаются произвольными, что вызывает справедливые нарекания пользователей.

В отличие от блок-схем, в дракон-схемах проведена полная формализация. Строго определены не только иконы, но и соединительные линии между ними, а также точки ввода фигур. Формализацию соединительных линий обеспечивают валентные точки и макроиконы.

Правила работы с валентными точками и макроиконами описаны в работе [39, pp. 393-472].

ЧАСТИЧНОЕ ДОКАЗАТЕЛЬСТВО ПРАВИЛЬНОСТИ АЛГОРИТМОВ

Роберт Андерсон подчеркивает: «целью многих исследований в области доказательства правильности программ является... механизация таких доказательств» [66]. Дэвид Грис указывает, что «доказательство должно опережать построение программы» [67].

Объединив оба требования, получим, что автоматическое доказательство правильности должно опережать построение алгоритма. Нетрудно убедиться, что метод исчисления икон обеспечивает частичное выполнение этого требования.

Из математической логики известно, что логический вывод позволяет применить к аксиомам правила вывода и получить строго доказанные теоремы. Визуальный логический вывод, следуя этой схеме, берет за основу две визуальных аксиомы языка ДРАКОН (аксиому-силуэт и аксиому-примитив). Применяя к ним визуальные правила вывода, получим шампур-схему, т. е. графический каркас дракон-алгоритма.

Мы выяснили, что любая правильно построенная шампур-схема является строго доказанной теоремой. В алгоритмах ДРАКОН-конструктора закодировано исчисление икон. Поэтому любая шампур-схема, построенная с его помощью и не содержащая

критических точек, является истинной, то есть правильно построенной. Этот результат означает, что:

> ДРАКОН-конструктор осуществляет частичное автоматическое доказательство правильности шампур-схем

К сожалению, данный метод позволяет доказать правильность шампур-схемы и только (без учета алгоритмических ошибок). Это составляет лишь часть от общего объема работы, которую нужно выполнить, чтобы доказать правильность алгоритма на 100%.

Здесь необходима оговорка. Частичное доказательство правильности алгоритма с помощью «ДРАКОН-конструктора» осуществляется автоматически и достигается совершенно бесплатно, так как дополнительные затраты труда, времени и ресурсов не требуются. Так что полученный результат (почти безошибочное автоматическое проектирование графики дракон-схем) следует признать значительным достижением.

ПРОГРАММНО-АЛГОРИТМИЧЕСКИЕ ОШИБКИ И СРЕДСТВА БОРЬБЫ С НИМИ

Ошибки в алгоритмах и программах (software bugs, logic errors) доставляют много неудобств специалистам и пользователям. Для предотвращения, поиска и исправления ошибок используют разнообразные средства:

- требования к программному обеспечению (software requirements);
- спецификация требований программного обеспечения (software requirements specification);
- просмотр кода (code review);
- статический анализ кода (static code analysis);
- тестирование (software testing);
- тестирование на основе модели (model-based testing);
- отладка (debugging);
- защитное программирование (defensive and secure programming);
- рефакторинг (refactoring);
- система отслеживания ошибок (bug tracking system);
- формальная верификация (formal verification);
- контрактное программирование (design by contract);
- проверка моделей (model checking);
- стандарт оформления кода, стиль программирования (coding standard, coding convention, programming style).

Указанные средства обладают несомненными достоинствами, однако они не снимают проблему. Сохраняется потребность в разработке новых средств противодействия ошибкам. Метод исчисления икон может дополнить этот список для случая визуального программирования.

ВЫВОДЫ

1. Для предотвращения ошибок разработан специальный математический аппарат — визуальное логическое исчисление под названием «исчисление икон».
2. Данное исчисление является разделом *визуальной* математической логики.

3. Показано, что графический синтаксис языка ДРАКОН представляет собой исчисление икон.
4. Исчисление икон является теоретическим обоснованием языка ДРАКОН.
5. Графика дракон-алгоритмов играет важную роль.
6. Визуальное логическое исчисление служит для защиты графики от ошибок.
7. Исчисление икон реализовано во внутренних алгоритмах «ДРАКОН-конструктора».
8. Программа «ДРАКОН-конструктор» осуществляет частичное автоматическое доказательство правильности шампур-схем.
9. Почти безошибочное автоматическое проектирование графики дракон-схем является полезным достижением, повышающим производительность труда при практическом работе.

Глава 33
ИНТЕЛЛЕКТУАЛЬНАЯ ПРОГРАММА ДРАКОН-КОНСТРУКТОР. КОНЦЕПЦИЯ

ПРОБЕЛ В ТЕОРИИ АЛГОРИТМОВ. КОГНИТИВНО-ЭРГОНОМИЧЕСКАЯ ПРОБЛЕМА

Классическая теория алгоритмов охватывает теорию вычислимости, теорию сложности вычислений, теорию автоматов, теорию формальных языков. Недостаток классической теории состоит в том, что она упускает из виду когнитивно-эргономическую проблему, связанную со способностью человека выявлять ошибки при зрительном восприятии алгоритма. Когнитивная эргономика служит для повышения удобочитаемости программного кода и уменьшения сложности алгоритма.

Для решения проблемы вводится понятие «эргономичный алгоритм». Оно используется для предотвращения ошибок разработчиков алгоритмов и программ. Алгоритм можно назвать *эргономичным,* если процесс зрительного восприятия, понимания и постижения алгоритма протекает с максимальной скоростью, наименьшими умственными усилиями и ***максимальной вероятностью выявления ошибок.*** Чтобы построить эргономичный алгоритм, нужно изменить нотацию алгоритмов в благоприятном направлении, чтобы трудная и непосильная задача ручного поиска ошибок превратилась в легкую и посильную.

Эргономичный алгоритм — междисциплинарное понятие, впервые объединившее математическую теорию алгоритмов с когнитивной эргономикой в рамках единой концепции, облегчающей предотвращение ошибок [39, pp. 9, 54, 334-338, 487].

ДРАКОН-конструктор использует оба метода: математический и эргономический. Когнитивная эргономика представляет собой научный подход к эргономизации конструкций языка. Такой подход позволяет улучшить визуальные образы языка (визуальные формы фиксации знаний), согласовав их с характеристиками глаза и мозга. Эргономичность позволяет задействовать резервы человеческого фактора для предотвращения ошибок [39].

ТРЕБОВАНИЯ К ЗАЩИТЕ ОТ ОШИБОК, ПРЕДЪЯВЛЯЕМЫЕ К ИНСТРУМЕНТАМ ЯЗЫКА ДРАКОН

При создании языка ДРАКОН было выдвинуто требование: повысить эффективность защиты от ошибок, но при этом максимально сэкономить издержки. Желательно добиться цели без дополнительных затрат труда, времени и ресурсов со стороны пользователей, разработчиков дракон-схем.

Как это сделать? Необходимо разработать программу «ДРАКОН-конструктор» (drakon-builder), чтобы она умела в полуавтоматическом (диалоговом) режиме рисовать безошибочные или почти безошибочные дракон-алгоритмы (рис. 169).

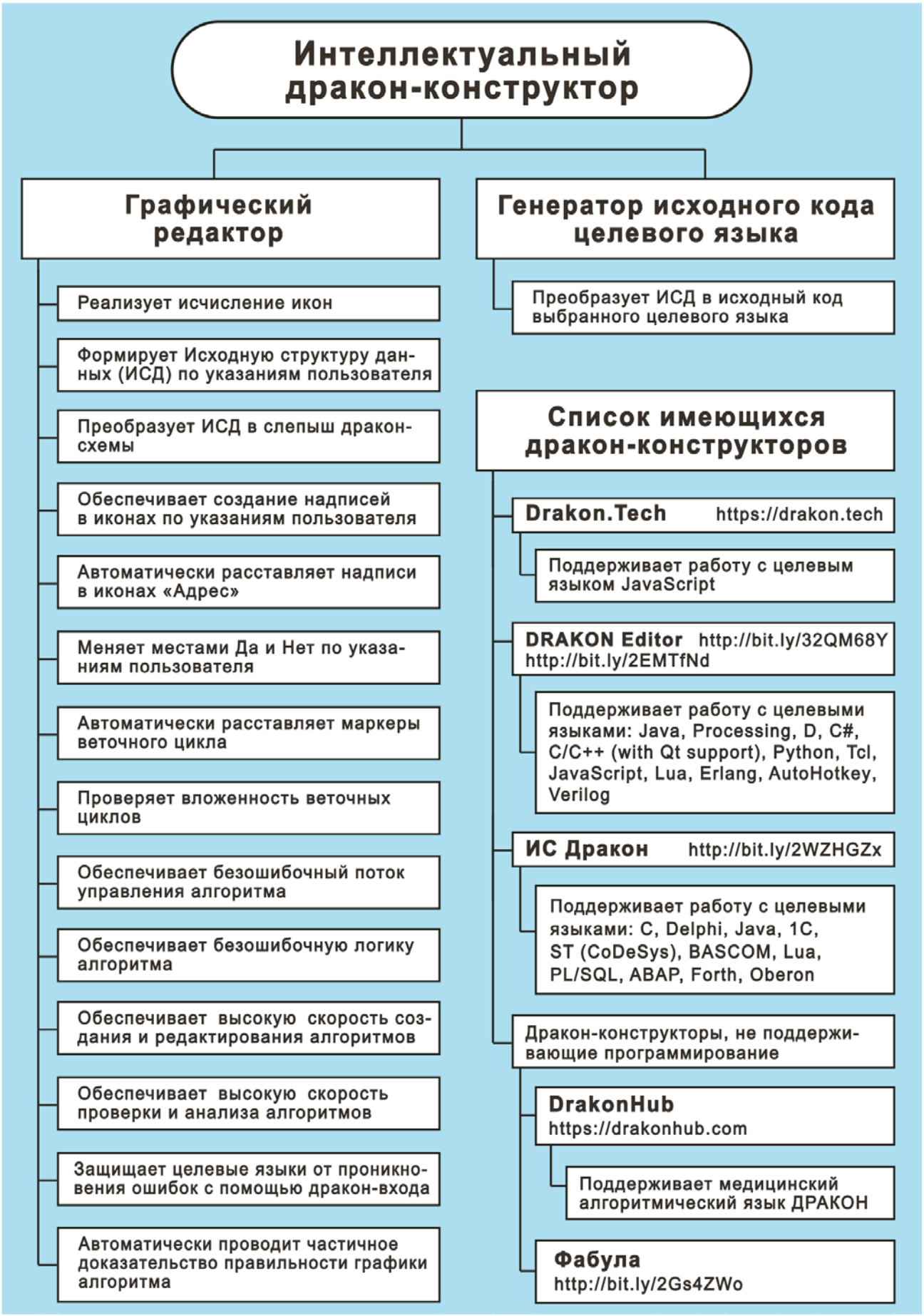


Рис. 169. Концептуальная схема ДРАКОН-конструктора

ИСХОДНАЯ СТРУКТУРА ДАННЫХ И ДРАКОН-МЕТОДОЛОГИЯ

Если рисовать дракон-схему с помощью обычного графического редактора, мы получим картинку, но не программу. Кроме того, будет много ошибок, а рисование займет много времени. Чтобы создать программу, исключить ошибки и ускорить процесс, необходимо использовать ДРАКОН-конструктор, в котором реализовано исчисление икон.

Когда программист (с помощью ДРАКОН-конструктора) рисует на экране дракон-схему, в памяти компьютера формируется исходная структура данных (ИСД). ДРАКОН-конструктор выполняет ряд функций:

- преобразует ИСД в графическое изображение дракон-схемы, которое отображается на экране компьютера;
- преобразует ИСД в исходный код выбранного целевого языка (например, языка Си);
- дракон-конструктор способен преобразовать ИСД в исходный код не одного, а нескольких целевых языков.

Преобразователь ИСД в исходный код целевого языка (т. е. генератор кода) называется *маршрутным транслятором.*

Код на выходе маршрутного транслятора подается на вход компилятора целевого языка (например, языка Си). Компилятор Си превращает исходный текст Си в исполняемый код (executable code).

ДОКАЗАТЕЛЬСТВО ВЫПОЛНЯЕТСЯ АВТОМАТИЧЕСКИ

Правильность графики алгоритма должна быть математически доказана. Допускается не полное, а частичное доказательство, которое реализует ДРАКОН-конструктор.

Частичное доказательство правильности графики алгоритма осуществляется без участия человека методом логического вывода. Оно достигается *автоматически* и бесплатно, так как дополнительные затраты труда, времени и ресурсов не требуются [39, pp. 425-472].

Программа ДРАКОН-конструктор называется интеллектуальной, так как она автоматически и безошибочно реализует многошаговый логический вывод.

ТРИ ОТЛИЧИЯ ДРАКОН-МЕТОДОЛОГИИ

При создании гибридного языка Дракон-Си необходимо, в частности, создать *маршрутный транслятор,* преобразующий ИСД в исходный код языка Си.

При использовании гибридных языков исходным текстом программы считается дракон-схема, и только она. При отладке программы не следует вносить исправления в промежуточные файлы на целевых языках, например, в си-файлы. Все исправления нужно вносить в исходный код, то есть в дракон-схему [39, pp. 263-266].

Можно указать три отличия ДРАКОН-методологии от обычной практики программирования на Си (или на другом целевом языке).

- Язык ДРАКОН упрощает и облегчает программирование на языке Си (или на другом языке), поскольку отпадает необходимость использовать наиболее сложные текстовые операторы управления, описывающие циклы и разветвления. Последние заменяются эргономичной графикой. На долю программиста остается простая работа — закодировать ***линейные участки программы.***

- Исходный код на языке Си программисты пишут вручную, а при использовании ДРАКОНа он формируется не вручную, а автоматически на выходе маршрутного транслятора.
- При традиционной отладке программисты вносят исправления в исходный код на языке Си. При использовании ДРАКОНа запрещено трогать и тем более исправлять исходный код на языке Си. Все исправления следует вносить только и исключительно в исходную дракон-программу.

КАКИЕ ЦЕЛЕВЫЕ ЯЗЫКИ МОЖНО ИСПОЛЬЗОВАТЬ

Это зависит от выбранного дракон-конструктора:

- программа DRAKON Editor преобразует дракон-программу в исходный код любого из 13 целевых языков: Java, Processing, D, C#, C/C++ (with Qt support), Python, Tcl, JavaScript, Lua, Erlang, AutoHotkey и Verilog [68];
- программа ИС Дракон поддерживает 11 языков: C, Delphi, Java, 1C, ST (CoDeSys), BASCOM, Lua, PL/SQL, ABAP, Forth, Oberon [69];
- программа Drakon.Tech поддерживает язык JavaScript [70].

Существует возможность расширить эти списки и добавить новые языки.

ГРАФИКА НУЖНА ДЛЯ ЧЕЛОВЕКА, А ТЕКСТ — ДЛЯ КОМПЬЮТЕРА

Противопоставление графики и текста на рис. 158-163 имеет глубокий смысл. Графика облегчает зрительное восприятие дракон-алгоритма; она нужна человеку и служит для предотвращения ошибок и для общения между людьми. Управляющий текст при этом рассматривается как лексический и синтаксический мусор и подлежит изъятию из дракон-алгоритма.

Но изъять не значит уничтожить. Текст перемещается в другое место — в исходный код целевого языка (например, Си). Это делается с помощью маршрутного транслятора. Последний автоматически преобразует структуру данных ИСД, связанную с дракон-программой (см. правые части на рис. 158-163) в исходный код языка Си.

Повторяем: в си-код смотреть не нужно. Так же, как мы не смотрим в машинный код после компиляции.

ЭРГОНОМИЧЕСКИЕ ВОЗМОЖНОСТИ ДРАКОН-КОНСТРУКТОРА

В данном параграфе перечислены эргономические правила, которые ДРАКОН-конструктор реализует автоматически, без участия пользователя. Правила разделены на четыре группы.

Группа 1. **Общие эргономические правила:**

- запрет на пересечения соединительных линий;
- непрерывность маршрутов между Заголовком и Концом, уменьшающая вероятность недостижимого кода;
- минимизация числа стрелок и запрет на использование стрелок вне циклов;
- уменьшение длины адресных вертикалей;
- правило ритма и метра.

Группа 2. **Правила шампура:**

- выделение шампура жирной линией;
- непрерывность шампура;
- запрет изгибов шампура.

Группа 3. **Правила семейного сходства:**

- равенство высоты всех икон «Имя ветки» независимо от числа строк;
- равенство высоты всех икон Адрес независимо от числа строк;
- равенство высоты всех икон Вариант независимо от числа строк.

Группа 4. **Правила выравнивания:**

- выравнивание икон «Имя ветки» по верху;
- выравнивание икон «Адрес» по низу;
- выравнивание икон «Вариант» по верху.

ПРИМЕРЫ

Пример 1. Поясним правило «уменьшение длины адресных вертикалей» (рис. 170). Необходимость в этом возникает, когда от иконы Вопрос (или от переключателя) вниз идут связи к иконам Адрес.

Лишние длинные линии приносят вред — они загромождают зрительную сцену и являют собой визуальные помехи. Чтобы сократить длину вертикалей, нужно опустить вниз икону Вопрос (или переключатель), т. е. приблизить ее к иконам Адрес, как показано на рис. 170, *справа.*

Пример 2. Правило семейного сходства гласит: иконы «Имя ветки» должны иметь одинаковую высоту. На рис. 171 показан ошибочный пример, где это правило не соблюдается.

В чем ошибка? В том, что высота иконы «Имя ветки» зависит от числа строк в ней. Чем больше строк, тем больше высота. Это, конечно, неправильно.

На рис. 172 ошибка исправлена и показан хороший пример. Чтобы его реализовать, ДРАКОН-конструктор должен:

— сравнить число строк во всех иконах «Имя ветки»;

— выбрать наибольшее число;

— рассчитать высоту иконы «Имя ветки» для наибольшего числа строк;

— назначить указанную высоту для всех икон «Имя ветки».

Пример 3. Правило выравнивания говорит: выровняй иконы «Имя ветки» по верхней линии. На рис. 173 изображен случай, когда правило нарушено: все иконы размещены лесенкой на разных уровнях. Ошибка исправлена на рис. 172, где аккуратно выполнены оба правила: и выравнивание, и семейное сходство.

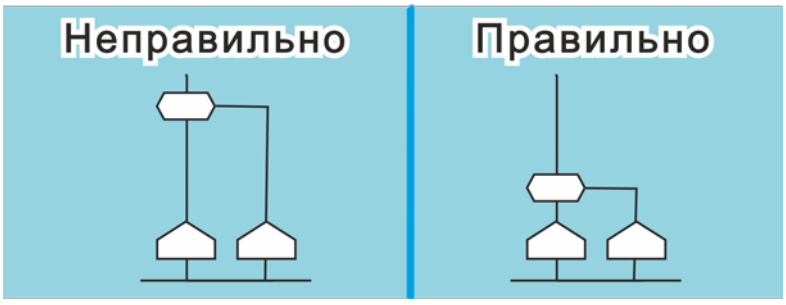


Рис. 170. Уменьшение длины адресных вертикалей.
Слева видны две длинные вертикали, а справа только одна

Рис. 171. Правило семейного сходства нарушено:
три иконы «Имя ветки» имеют разную высоту

Рис. 172. Правило семейного сходства выполнено:
три иконы «Имя ветки» равны по высоте, несмотря на разное число строк

Рис. 173. Нарушено правило выравнивания:
три иконы «Имя ветки» не выровнены по верху

ВЫВОДЫ

1. Программа «ДРАКОН-конструктор» (drakon-builder) предназначена для быстрого создания и редактирования дракон-алгоритмов с использованием математических и эргономических инструментов, специально разработанных для безошибочной работы.
2. Программа «ДРАКОН-конструктор» реализует логическое исчисление икон.
3. Частичное доказательство правильности графики алгоритма осуществляется методом визуального логического вывода. При этом дополнительные затраты труда, времени и ресурсов не требуются.
4. Алгоритм называется эргономичным, если зрительное восприятие и понимание алгоритма протекает с максимальной скоростью, наименьшими умственными усилиями и максимальной вероятностью выявления ошибок.
5. Чтобы построить эргономичный алгоритм, нужно изменить нотацию алгоритмов, чтобы трудная задача ручного поиска ошибок превратилась в легкую и посильную.

6. Дракон-конструктор:
    - преобразует структуру данных в графический образ дракон-схемы, который отображается на экране компьютера;
    - преобразует структуру данных в исходный код выбранного целевого языка.
7. ДРАКОН-конструктор реализует автоматически 14 эргономических правил, разбитых на четыре группы:
    - общие эргономические правила,
    - правила шампура,
    - правила семейного сходства,
    - правила выравнивания.

Глава 34

ПРАКТИЧЕСКАЯ РАБОТА С ДРАКОН-КОНСТРУКТОРОМ

КАК ПОЛЬЗОВАТЕЛЬ СОЗДАЕТ ДРАКОН-СХЕМУ

Пользователь рисует дракон-схему на экране компьютера с помощью программы дракон-конструктор.

Попробуйте онлайн конструктор https://drakonhub.com — вы убедитесь, что процесс рисования происходит удобно, легко и с большой скоростью.

ПРАВИЛА ДРАКОНА

Язык ДРАКОН содержит большое число правил. К счастью, их не надо учить и запоминать. Почему? Потому что правила знает назубок, хранит в своей обширной памяти и скрупулезно выполняет программа дракон-конструктор.

Вот пример. В дракон-схемах запрещено пересечение линий. Запрет выполняется автоматически. Конструктор гарантирует отсутствие пересечений. Рисунки в этой книге подтверждают правило. Пересечения линий нигде и никогда не встречаются.

ЗАДАЧА: ПОСТРОИТЬ СИЛУЭТ ПО ЗАДАННОМУ ОБРАЗЦУ

Как работает дракон-конструктор? Каким образом он строит силуэт? Как это происходит?

Силуэт — основной инструмент языка ДРАКОН, обладающий большим арсеналом выразительных средств. Мы знаем, что силуэт способен изображать сложные алгоритмы. Однако мы ничего не знаем о тех механизмах, которые использует конструктор при вычерчивании силуэта. Пора прояснить этот вопрос и рассказать о секретах дракон-конструктора.

Давайте построим силуэт, изображенный на рис. 174. Перед нами образец, который мы хотим воспроизвести с помощью конструктора.

Как приступить к делу? Любой силуэт выращивают из «семечка». Семечком является аксиома-силуэт, показанная на рис. 166, *слева.*

Стало быть, начать нужно с аксиомы. Мы будем последовательно, шаг за шагом ее изменять и в итоге получим нужный силуэт.

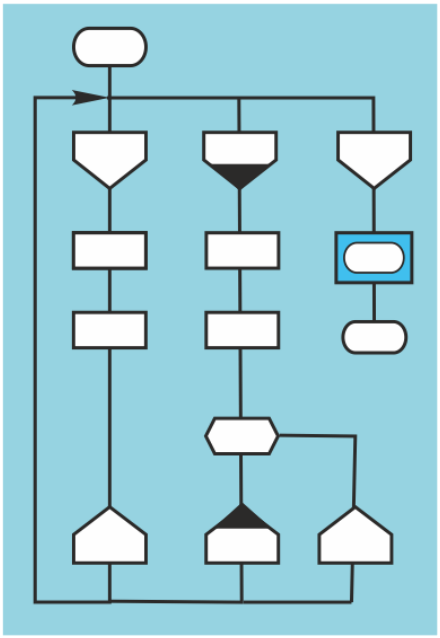


Рис. 174. Задача: нужно построить такой силуэт

Трудность в том, что аксиому изменять нельзя. Чтобы обойти это препятствие, назовем аксиому-силуэт *заготовкой.* Аксиома останется неизменной, а заготовку мы будем изменять.

Исходя из этого, поместим заготовку-силуэт туда, куда нужно — на рисунок 175 в левый верхний угол. Имея заготовку, будем постепенно, шаг за шагом, превращать ее в образец на рисунке 174.

НЕ ЦАРСКОЕ ЭТО ДЕЛО

Здесь нас ждет приятный сюрприз. Забудьте о линиях! Не царское это дело — рисовать линии! Все линии аккуратно, без ошибок и гораздо лучше вас нарисует ваш покорный слуга дракон-конструктор. Он нарисует их автоматически и моментально. На вашу долю остается легкая и приятная работа — выбирать из меню нужные иконы (кубики) и указывать на чертеже валентные точки.

Как только вы добавите в схему очередную икону, умница-конструктор сразу приделает к ней все необходимые линии.

ПРИМЕР ПОСТРОЕНИЯ ДРАКОН-СХЕМЫ «СИЛУЭТ»

Решение задачи представлено в виде десяти последовательных шагов на трех рисунках: 175 *(шаги 1–4),* 176 *(шаги 5–7) и* 177 *(шаги 8–10)*.

Результатом является точная копия образца. Этот результат находится на рис. 177 в правом нижнем углу *(см. шаг 10).* Сравнив результат с рисунком 174, можно удостовериться, что они полностью совпадают. Значит, задача решена правильно.

Что ж, «начнем, пожалуй», как говорил Ленский перед дуэлью.

На первом шаге вызываем из меню кубик Ветка (см. среднюю графу на рисунке 175, *шаг 1).*

Куда его поместить? Подводим курсор к нужной валентной точке (белый кружок) в заготовке. К той самой точке, в которой следует разорвать соединительную линию, чтобы в образовавшийся разрыв вставить выбранный кубик. Тем самым мы изменяем заготовку, добавляя в нее еще одну ветку. Результат операции виден на рисунке 175, *шаг 1, справа.*

Три следующих шага выполняем аналогично. В схему последовательно вставляем три иконы Действие (рис. 175, *шаги 2–4*).

Подведем предварительный итог. О чем говорит рисунок 175? Что сделано за первые четыре шага? Каков сухой остаток?

Глядя на среднюю графу, убеждаемся, что мы выполнили четыре команды:

- Вставь Ветку.
- Вставь Действие.
- Вставь Действие.
- Вставь Действие.

Белые и черные кружки в левой графе показывают, что для каждой команды мы заботливо указали валентную точку, куда надо вставить очередной кубик.

Обратимся теперь к самому интересному — к правой графе. Просматривая ее сверху вниз, мы видим, как постепенно, шаг за шагом, растет и обрастает новыми деталями дракон-схема «силуэт», которая и является нашей целью.

Итог наших стараний на рисунке 175 находится в правом нижнем углу. На данном этапе он похож на недостроенное здание и состоит всего из 10 икон.

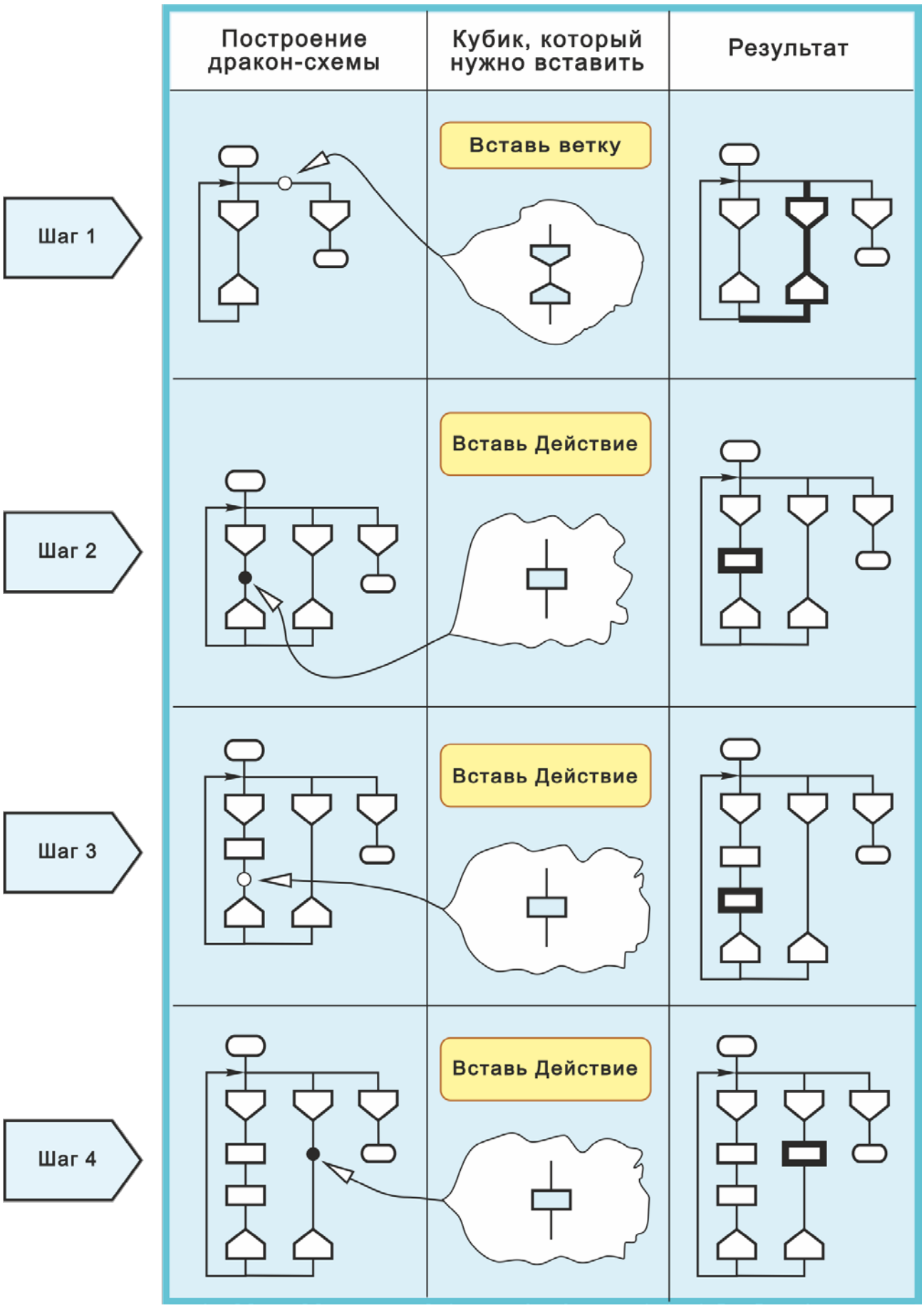


Рис. 175. Конструирование дракон-схемы силуэт с помощью дракон-конструктора

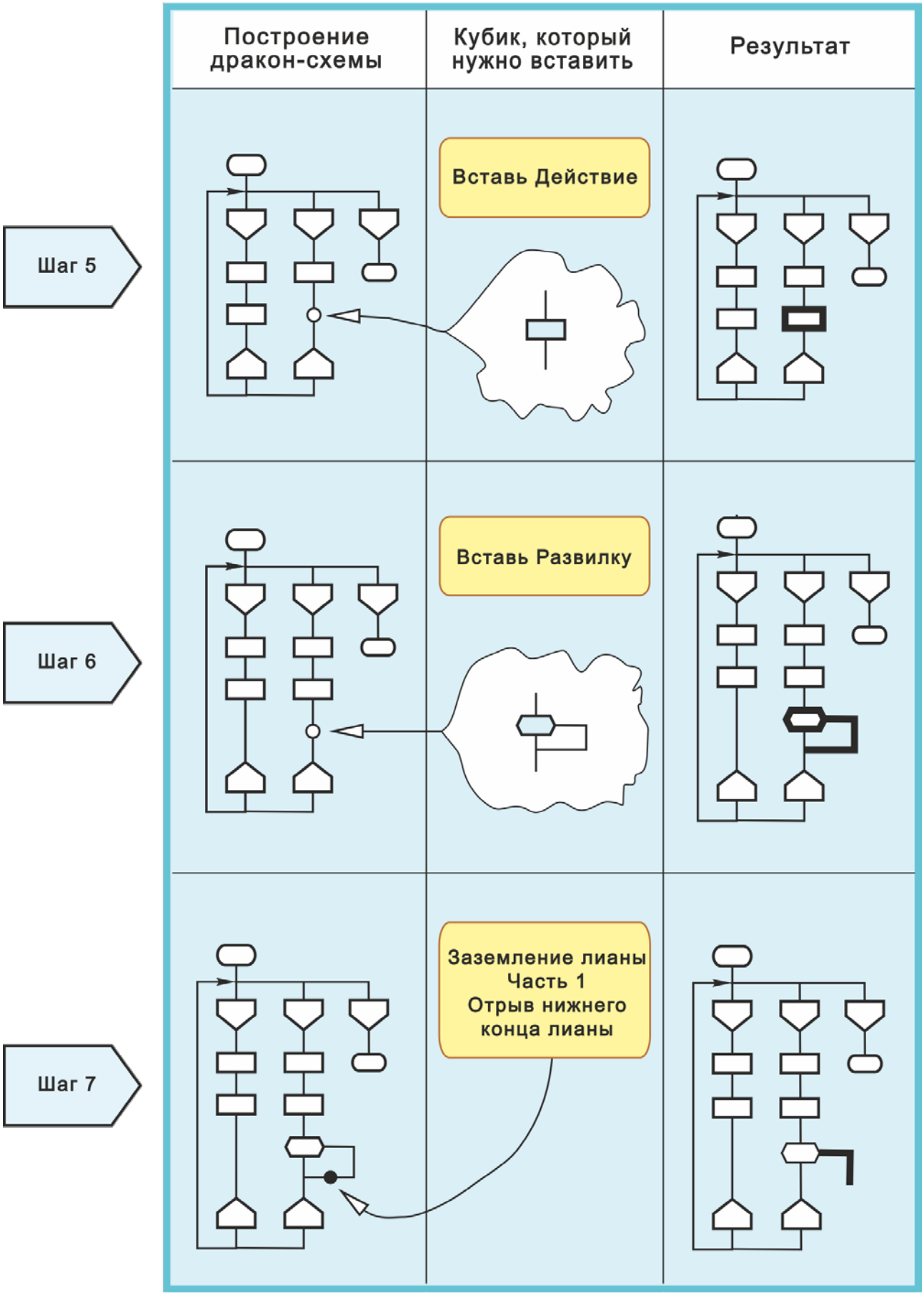


Рис. 176. Конструирование дракон-схемы силуэт (продолжение)

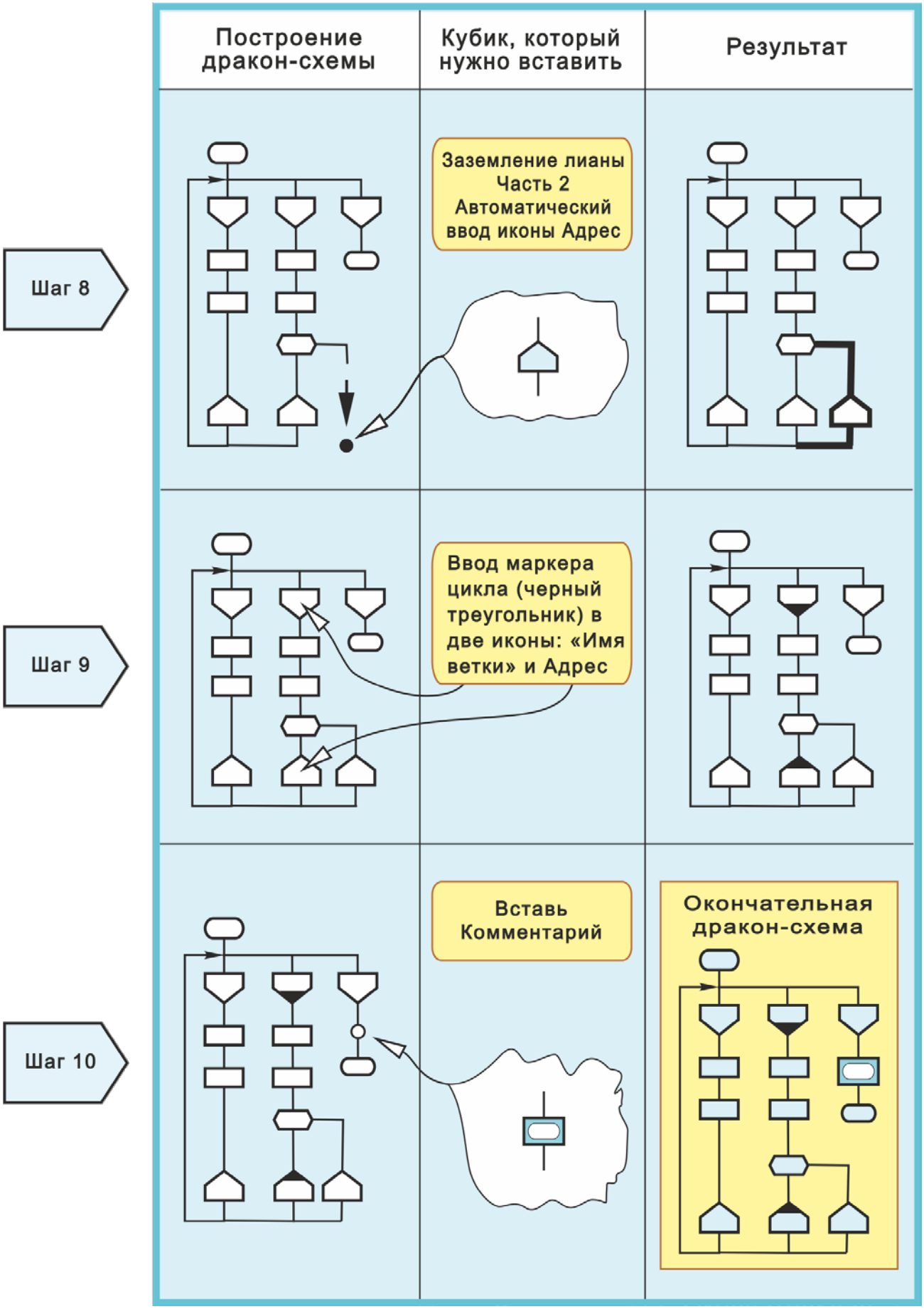


Рис. 177. Конструирование дракон-схемы силуэт (окончание)

Идем дальше и переводим взгляд на рисунок 176. По очереди вставляем в схему кубики Действие, Развилка и начинаем заземлять лиану. Начинать-то мы начинаем, но пока не заканчиваем. Мы успели лишь оторвать нижний конец лианы. Глядя в правый нижний угол, видим, что оторванный конец (это и есть лиана) сиротливо висит и ждет своей участи.

Подытожим. Глядя на среднюю графу, замечаем, что мы выполнили две команды:

- Вставь Действие.
- Вставь Развилку.

После этого мы наполовину выполнили заземление лианы (рис. 176, *шаги 5–7).*

Переходим к рисунку 177. Здесь мы вместе со слугой-конструктором совершили три подвига:

Подвиг 1. Доведено до конца заземление лианы. Пользователь щелкнул мышью по валентной точке на нижней шине силуэта. А умница-конструктор мигом все понял и автоматически:

- вставил икону Адрес;
- присоединил к иконе Адрес все необходимые линии.

Это значит, что мы создали еще один выход из ветки.

Подвиг 2. Проставлены маркеры веточного цикла (черные треугольники) в иконы «Имя ветки» и «Адрес». Этот подвиг умница-конструктор выполняет полностью самостоятельно, без вмешательства человека, так что пользователь даже мизинцем не пошевелил. Оговоримся: конструктор автоматически выполняет эту операцию, анализируя текст в иконах «Имя ветки» и «Адрес». При отсутствии текста он ничего не делает. Так что в нашем «бессловесном» примере на рис. 174 простановка маркеров невозможна; она выполняется только после расстановки надписей в иконах.

Подвиг 3 более чем скромен. Это всего лишь выполнение заурядной команды «Вставь Комментарий».

На этом летопись подвигов ДРАКОН-конструктора можно закончить. Результат последней операции показан на рис. 177 (*шаг 10, справа*). Этот результат венчает дело, потому что в точности совпадает с заданием на рис. 174.

Что дальше? Когда графический узор (слепыш) дракон-схемы построен, производится заполнение его текстом. Только после этого дракон-конструктор делает последний штрих и производит расстановку маркеров веточного цикла.

Уточним: обычно пользователь заполняет каждую икону текстом сразу после ее появления на экране, не дожидаясь завершения построения слепыша.

ЧТО ДЕЛАЕТ ДРАКОН-КОНСТРУКТОР ПРИ ЗАЗЕМЛЕНИИ ЛИАНЫ

Программа дракон-конструктор выполняет много полезной автоматической работы. Выше говорилось, что конструктор полностью освобождает пользователя от вычерчивания линий. Это наглядно видно при выполнении операции «заземление лианы», которая создает новый выход из ветки (см. шаги 7 и 8 на рис. 176 и 177).

Что должен сделать пользователь? Он должен коснуться валентной точки на нижней шине силуэта. И все.

При этом дракон-конструктор выполняет следующую серию автоматических действий:

- вставляет в схему икону Адрес;
- удлиняет нижний отросток иконы Адрес вниз до пересечения с нижней шиной;

- удлиняет нижнюю шину силуэта вправо до пересечения с нижним отростком иконы Адрес;
- соединяет оторванный конец лианы с верхним отростком иконы Адрес;
- устраняет неточное совпадение концов линий и при необходимости перекомпоновывает схему, чтобы не было лишних изгибов линий.

ФОРМИРОВАНИЕ НАДПИСЕЙ «ДА» И «НЕТ»

Возле каждой иконы Вопрос обязательно должны стоять надписи «Да» и «Нет». Эти надписи появляются на дракон-схеме всякий раз, когда из меню вызывается элемент, содержащий икону Вопрос.

Первоначально дракон-конструктор пишет «Да» у нижнего выхода иконы Вопрос и «Нет» — у правого. Пользователь может менять их местами. Для этого в конструкторе предусмотрена операция «Да/Нет». При выполнении этой операции, слова Да и Нет меняются местами.

ГДЕ СКАЧАТЬ ДРАКОН-КОНСТРУКТОР

К вашим услугам не одна, а пять программ, каждая из которых умеет рисовать дракон-алгоритмы.

Начать знакомство лучше всего с программы «DrakonHub» (автор Степан Митькин) http://drakonhub.com Это онлайн программа. Ее не надо скачивать, она работает в браузере.

Вторая офлайн программа «DRAKON Editor» также создана Степаном Митькиным https://sourceforge.net/projects/drakon-editor/files

Третья онлайн программа «Drakon.Tech» — последняя разработка Митькина.

Четвертая программа называется «ИС Дракон», автор Геннадий Тышов. http://drakon.su/programma_is_drakon

Пятая программа «Фабула», разработчик Эдуард Ильченко:
https://forum.drakon.su/viewtopic.php?t=5475

Программы непрерывно развиваются и дополняются. Не исключено, что к моменту выхода книги на сайте и форуме ДРАКОНа появится информация о создании новых, более совершенных вариантов ДРАКОН-конструктора.

Дракон-конструктор	Назначение
1. Drakon.Tech 2. DRAKON Editor 3. ИС Дракон	Для программирования
4. DrakonHub 5. Фабула	Без программирования

ГДЕ ПОЛУЧИТЬ ИНТЕРНЕТ-КОНСУЛЬТАЦИИ

Если что-то непонятно, можно получить помощь на форуме языка ДРАКОН. После обычной регистрации вы можете задавать вопросы непосредственно авторам программ и участвовать в обсуждении: https://forum.drakon.su

Официальный сайт языка ДРАКОН http://drakon.su

ВИДЕО И ПРЕЗЕНТАЦИИ

Возможно, кто-нибудь захочет быстро получить общее представление о языке ДРАКОН и его программах. Лучше всего это сделать с помощью видео и слайд-шоу:

https://forum.drakon.su/viewforum.php?f=179

http://drakon.su/video_i_prezentacii/start

Загляните также в Википедию, где статья «ДРАКОН» представлена:

- на русском языке https://ru.wikipedia.org/?oldid=94374256
- на английском языке https://en.wikipedia.org/wiki/DRAKON
- на французском языке https://fr.wikipedia.org/wiki/DRAKON

ДВЕ ТОЧКИ ЗРЕНИЯ

Работу дракон-конструктора можно рассматривать с двух точек зрения: практической и математической. В этой главе мы рассмотрели практику — превращение заготовки на рис. 166 *(слева)* в силуэт на рис. 174.

С позиций математической логики тот же самый процесс трактуется иначе — как визуальный логический вывод. К аксиоме-силуэт на рис. 166 последовательно применяются правила логического вывода. Это значит, что слепыш на рис. 174 представляет собой визуальную теорему, которая строго выводится из аксиомы-силуэт.

Нужно ли доказывать эту теорему? Нет, не нужно, так как графический синтаксис языка ДРАКОН построен как визуальное логическое исчисление (исчисление икон) [39, pp. 427-435].

Программа дракон-конструктор хранит в своей памяти правила исчисления икон и реализует их при вычерчивании графики силуэта.

Как это делается? Как аксиома-силуэт превращается в нужный слепыш? Это достигается благодаря тому, что дракон-конструктор, подчиняясь разумным указаниям пользователя (и отклоняя неразумные) автоматически применяет правила визуального логического вывода к аксиоме-силуэт.

ВЫВОДЫ

1. ДРАКОН-конструктор — общее название, обозначающее программы для конструирования и вычерчивания алгоритмов и жизнеритмов.
2. В наличии имеются пять программ ДРАКОН-конструктор:
   - «Drakon.Tech» (для программирования),
   - «DRAKON Editor» (для программирования),
   - «ИС Дракон» (для программирования),
   - «DrakonHub» (без программирования),
   - «Фабула» (без программирования).
3. Интернет-консультации по языку ДРАКОН и ДРАКОН-конструктору можно получить на форуме языка ДРАКОН https://forum.drakon.su
4. Официальный сайт языка ДРАКОН http://drakon.su/
5. Видео и презентации http://drakon.su/video_i_prezentacii/start
6. Во внутренних механизмах ДРАКОН-конструктора реализован набор правил языка ДРАКОН, что освобождает пользователя от необходимости запоминать правила.
7. ДРАКОН-конструктор создает только правильно построенные графические схемы и исключает возможность появления запрещенных схем.

Часть 8

СТАНДАРТ НА АЛГОРИТМЫ УСТАРЕЛ И НУЖДАЕТСЯ В ЗАМЕНЕ

Глава 35

СРАВНЕНИЕ ДРАКОН-СХЕМ И БЛОК-СХЕМ.

ВВЕДЕНИЕ

В главе рассмотрены примеры блок-схем алгоритмов, взятые из технической литературы. Проведен когнитивно-эргономический анализ блок-схем и выявлены ошибки эргономического характера.

Для каждой блок-схемы рядом с ней в качестве образца изображена дракон-схема, решающая ту же задачу. Показано, что ошибки, присущие блок-схемам, в дракон-схемах отсутствуют — они выявлены и устранены.

Стандарт ГОСТ 19.701-90 не может обеспечить наглядность при вычерчивании сложных алгоритмов. Это объясняется тем, что концепция стандарта отстала от жизни и построена без учета идей когнитивной эргономики.

УДОБОЧИТАЕМОСТЬ АЛГОРИТМОВ

Алгоритм, представленный в виде графической схемы, предназначен для зрительного восприятия человеком. Следовательно, алгоритм представляет собой зрительную сцену. Или, если угодно — зрительный образ, зрительную картину.

Эргономичность алгоритма — это эстетическая привлекательность его зрительного образа, в частности, блок-схемы или дракон-схемы.

Мы помним, что алгоритм можно назвать эргономичным, если процесс понимания алгоритма протекает быстро и облегчает выявление ошибок.

Чем эргономичнее алгоритм, тем быстрее и легче можно его понять. Отсюда вытекает, что удобочитаемость и элегантность алгоритмов открывают путь к экономии умственных усилий. Но не только.

Чем эргономичнее созданы зрительные образы частей алгоритма, чем изящнее они соединены в общую алгоритмическую картину, тем приятнее на них смотреть. Чем точнее и быстрее зрительный «пейзаж» обнажает глубинный смысл алгоритма, тем плодотворнее мышление.

Чем больше эргономичность, тем глубже понимание алгоритмов. Тем скорее течет наша алгоритмическая мысль. Тем легче мы постигаем суть дела. Тем быстрее и качественнее протекает важнейший производственный процесс, играющий немалую роль в мировой экономике, — процесс массовой разработки алгоритмов и программ.

И наоборот, если зрительный образ алгоритма кажется запутанным и некрасивым, процесс понимания, обдумывания и поиска ошибок неизбежно замедляется, что снижает производительность умственного труда.

ДРАКОН — графический язык, язык зрительных образов. С учетом сказанного можно уточнить: ДРАКОН — язык эргономичных зрительных образов. Но

эргономичность ДРАКОНа не самоцель. Она позволяет ощутимо повысить производительность труда при создании алгоритмов.

Мы исходим из того, что зрительные образы алгоритмов следует сознательно проектировать. Для этой цели можно (в разумных пределах) использовать *средства художественного конструирования.*

Уместно напомнить слова видного психолога Б. Ф. Ломова, основателя Института психологии Академии наук СССР:

> «Средства художественного конструирования в конечном счете направлены на то, чтобы вызвать тот или иной эффект у работающего человека…
>
> Применяя средства художественного конструирования, мы создаем положительные эмоции, облегчаем операцию приема информации человеком, улучшаем концентрацию и переключение внимания, повышаем скорость и точность действий.
>
> Короче говоря, мы пользуемся этими средствами для управления поведением человека в широком смысле слова, для управления его психическим состоянием» и умственной работоспособностью [71].

ЭРГОНОМИЧНОСТЬ — ЭТО НАБОР ПРАВИЛ

Наша ближайшая цель — разъяснить, что эргономичность есть набор правил, которым должны подчиняться зрительные образы, представленные на бумаге или экране. Возьмем за основу зрительные образы языка ДРАКОН, построенные из икон и их комбинаций.

Мы рассмотрим десяток правил ДРАКОНа и на конкретных примерах покажем, что в блок-схемах правила систематически нарушаются, а в дракон-схемах — строго соблюдаются.

ПРАВИЛО ШАМПУРА

На первое место следует поставить правило шампура: *в схеме желательно и необходимо иметь шампур.* Шампур создает систему отсчета, в которой проектируется графическая схема алгоритма.

Ниже на многих примерах будут продемонстрированы типичные ошибки блок-схем:

- нет шампура,
- разрыв шампура.

СХЕМА ДОЛЖНА БЫТЬ ЛАКОНИЧНОЙ

Схема алгоритма должна содержать лишь те элементы, которые необходимы для сообщения читателю существенной информации, точного понимания ее смысла и стимулирования правильных решений и разумных действий. Пустые украшения, избыточные, затемняющие детали должны быть удалены из схемы.

Правило лаконичности	Зрительный образ алгоритма должен быть лаконичным. Все ненужные, лишние детали должны быть отсечены
Правило Эшфорда	Бесполезно стремиться направить внимание на важнейшие характеристики, если они окружены лишними, не относящимися к ним визуальными раздражителями, мешающими восприятию главного [103].

СЛЕДУЕТ ИЗБЕГАТЬ НЕОПРАВДАННЫХ ИЗГИБОВ СОЕДИНИТЕЛЬНЫХ ЛИНИЙ

Существуют вредные мелочи, которые затрудняют понимание схемы. Одна из них — неоправданные изгибы соединительных линий.

Сравним две схемы на рис. 178 и 179. На рис. 178 показана обычная блок-схема, заимствованная из книги сотрудников IBM [72]. На рис. 179 изображена эквивалентная дракон-схема.

Рисунки позволяют выявить различия между неприглядной блок-схемой и красивой дракон-схемой. С точки зрения правил удобочитаемости, блок-схема на рис. 178 имеет следующие недостатки:

- Неоправданно большое число изгибов линий (в блок-схеме 12 изгибов, а в дракон-схеме только 4).
- Большое число паразитных элементов: 14 стрелок и 3 кружка, которые в дракон-схеме отсутствуют (поскольку они совершенно не нужны и представляют собой визуальные помехи, затемняющие суть дела).

Правило минимизации изгибов

- Чтобы алгоритм был удобным для чтения, количество изгибов соединительных линий должно быть минимальным.
- Из двух схем лучше та, где число изгибов меньше

Мы рассмотрели два эргономических правила (правило лаконичности и правило минимизации изгибов). И убедились, что в дракон-схеме они строго соблюдаются, а в блок-схеме грубо нарушены.

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ДВУХ СХЕМ

Обратимся снова к рис. 178 и 179. Мы сделали лишь первый шаг к устранению графических недочетов. На рис. 178 осталось еще немало огрехов, которые необходимо выявить и исправить.

Итак, продолжим наш критический анализ. Блок-схема на рис. 178 имеет следующие недостатки.

- Для обозначения развилки используется ромб, который занимает слишком много места. Ромб не позволяет поместить внутри необходимое количество удобочитаемого текста, состоящего из строк равной длины. В дракон-схеме верхний и нижний углы ромба отрезаны. Поэтому схема становится компактной и удобной как для записи текста, так и для чтения.
- Функционально однородные иконы *Д, Е, Ж, И* хаотично разбросаны по всей площади чертежа, занимая три разных горизонтальных уровня (что путает читателя). В дракон-схеме они расположены на *одном* уровне, что служит для читателя подсказкой об их функциональной однородности.
- Ромбы имеют выход влево, что разрушает шампур и не позволяет применить правило главного маршрута. В дракон-схеме выход влево не допускается.
- Икона *Д* и ее вертикаль расположены слева от шампура (в дракон-схеме это запрещено).
- Ниже икон *Ж и И* находятся три уровня горизонтальных линий, которые имеют паразитный характер. В дракон-схеме три уровня сведены в одну линию, что делает схему более наглядной и компактной.

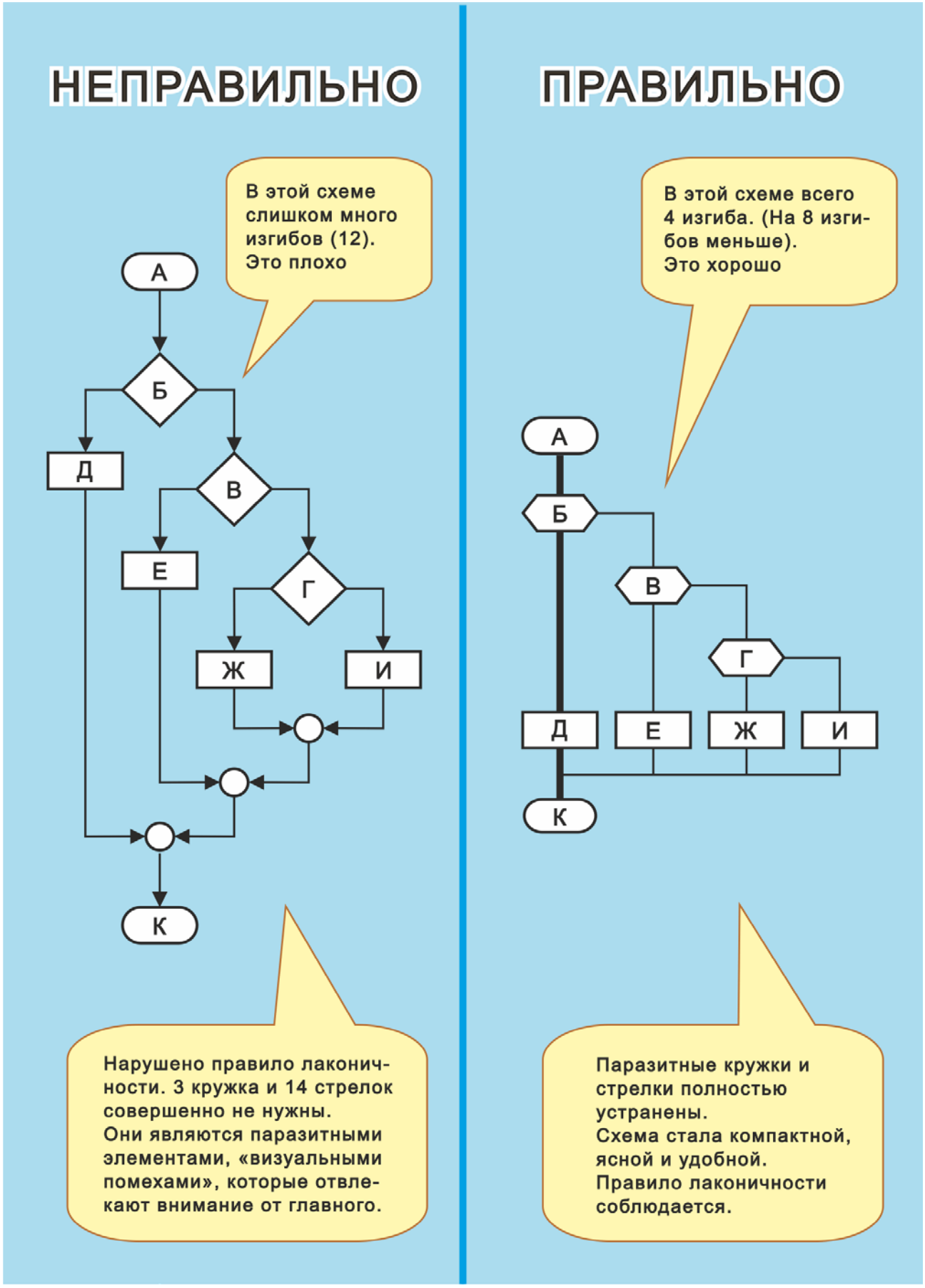


Рис. 178. Плохая схема. Недостатки: слишком много изгибов; имеются паразитные элементы

Рис. 179. Хорошая (эргономичная) схема. Она нарисована по правилам языка ДРАКОН

Да, конечно, каждое из этих улучшений является незначительным и не делает погоды. Но когда мелкие улучшения повторяются многократно и становятся массовыми, ситуация может измениться. Количество переходит в качество. В этом случае облегчение умственного труда может стать значительным.

КРИТИКА БЛОК-СХЕМ

Цивилизация не может жить без чертежей, не может жить и без алгоритмов. Схемы алгоритмов очень важны для понимания секретов и тайн современного производства и управления. Однако многие преподаватели и разработчики алгоритмов создают неприглядные и путаные блок-схемы, в которых трудно разобраться. Иногда их даже называют «мусорными» блок-схемами, потому что хитросплетения блоков, соединенные хаосом петляющих линий, больше напоминают кучу мусора, нежели регулярную структуру.

Образчик подобного мусора представлен на рис. 180 [18, p. 102]. Этот «мусорный» алгоритм можно вылечить и превратить в изящную дракон-схему (рис. 181). Сравним что было и что стало.

Схема на рис. 180 имеет много изъянов.

- Слева от иконы *Ж* есть пересечение линий (в дракон-схеме пересечения запрещены).
- Возле иконы *Е* имеется линия под углом 45° (в дракон-схеме наклонные линии не допускаются).
- Иконы *Д, Е* и *Ж* имеют более одного входа (в дракон-схеме это запрещено).
- Иконы *В, Д, Е, Ж* имеют входы сбоку, что придает схеме неряшливый вид. В дракон-схеме вход разрешается только сверху, что упорядочивает алгоритм и создает в нем четкую ориентацию «сверху вниз»).
- Отсутствует шампур, так как выход иконы Заголовок и вход иконы Конец не лежат на одной вертикали. Исчезновение шампура означает, что в схеме отсутствует зрительный остов, художественно-композиционная главная вертикаль. Тем самым уничтожается основа для выделения главного маршрута.

Блок-схема на рис. 180, как и предыдущая (рис. 178), по всем параметрам проигрывает дракон-схеме. Мы убедились, что алгоритмическая красота достигается благодаря совокупному действию многих правил, каждое из которых, взятое по отдельности, выглядит скромным и будничным.

РАЗРЫВ ШАМПУРА — СЕРЬЕЗНАЯ ОШИБКА

Разорванный шампур искажает зрительный образ схемы и может повлечь за собой неприятности. Между тем такая схема не только не осуждалась, а наоборот, в свое время была рекомендована стандартом *ANSI* (Американский национальный институт стандартов).

На рис. 182 представлена схема, взятая из источника [73], где она характеризуется как «стандартная блок-схема *ANSI*».

Сравнение этой схемы с эквивалентной дракон-схемой на рис. 183 позволяет выявить различные дефекты:

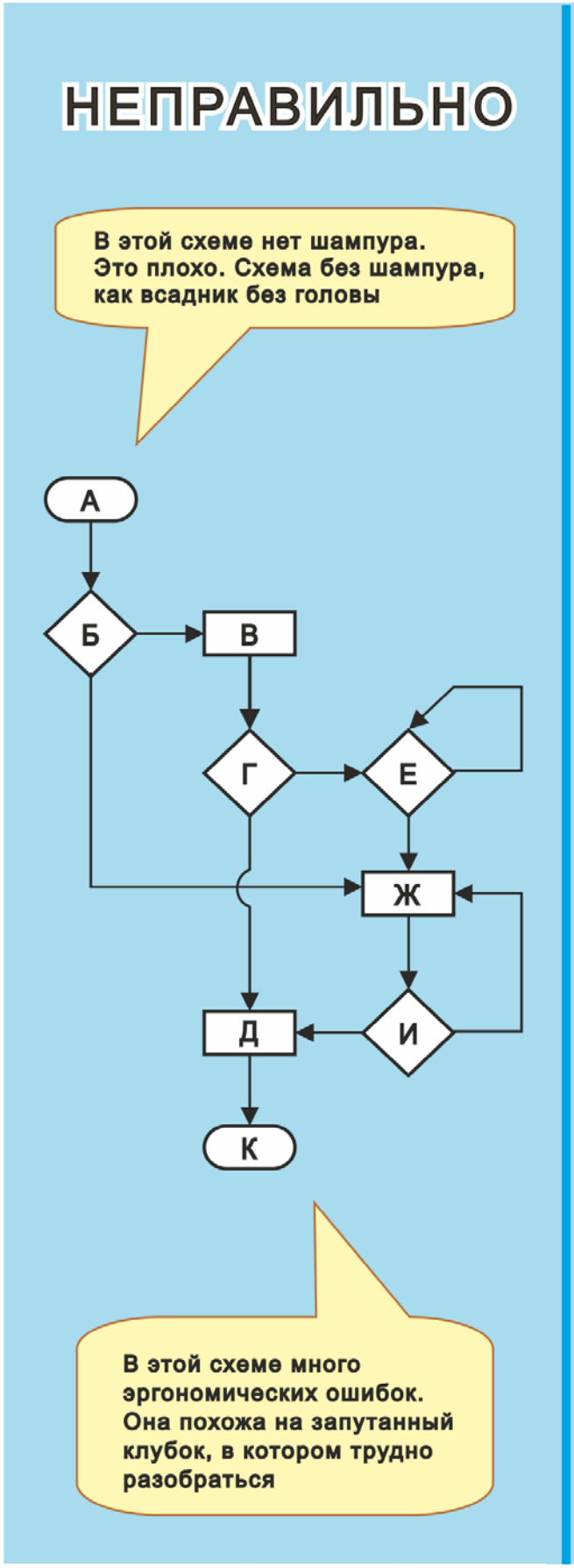


Рис. 180. Плохая схема. Такие схемы часто рисуют многие уважаемые ученые, забывающие об эргономике

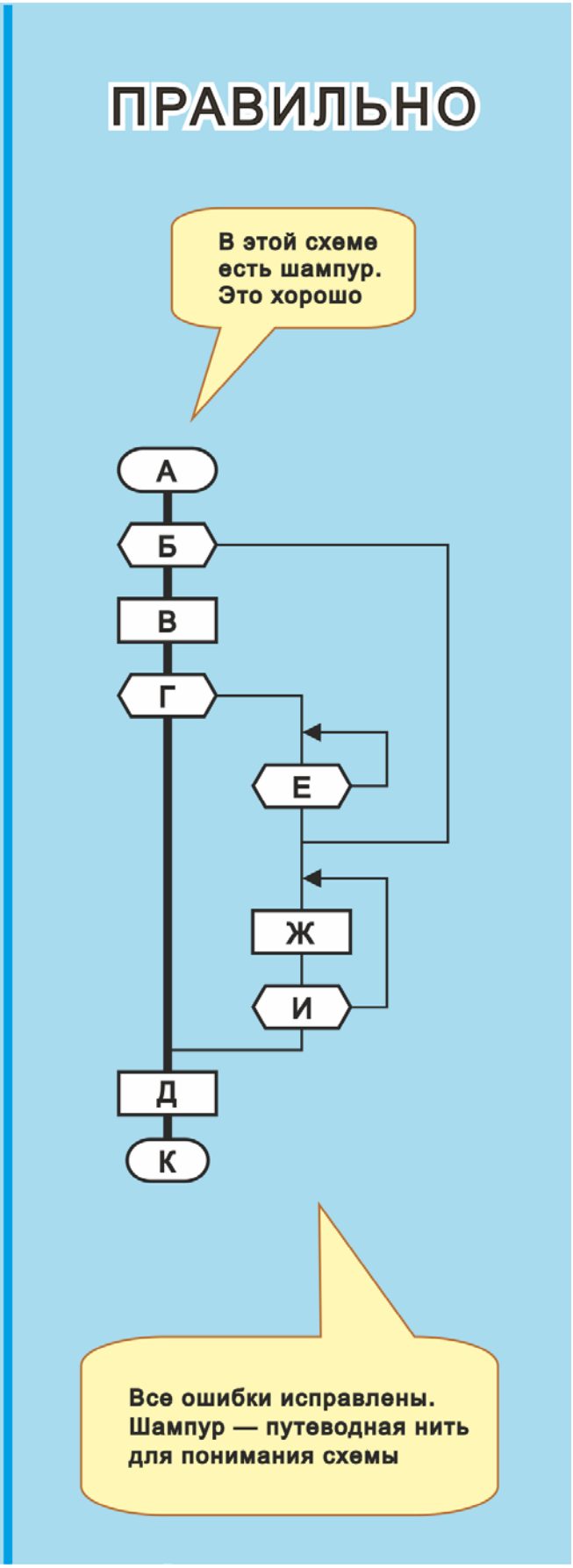


Рис. 181. Хорошая (эргономичная) схема. Она нарисована по правилам языка ДРАКОН

- Ниже иконы *G* имеет место разрыв шампура (нарушено правило, согласно которому один из путей, идущих от входа к выходу, должен проходить по главной вертикали).
- Икона *G* имеет два входа (в дракон-схеме разрешается только один вход).
- Икона *G* имеет вход сбоку (в дракон-схеме это запрещено).
- У иконы *G* выход находится слева (в дракон-схеме он должен быть снизу).
- Две петли обратной связи обычного цикла находятся слева от шампура и закручены по часовой стрелке (в дракон-схеме они расположены справа от шампура и закручены против часовой стрелки).
- Используются неудобные ромбы (в дракон-схеме их заменяют эргономичной иконой Вопрос).
- Ромб *L* имеет выход слева (в дракон-схеме он должен быть справа).
- Используются 12 стрелок, из которых 10 — паразитные (в дракон-схеме всего 2 стрелки).
- Имеется один избыточный изгиб линии (в блок-схеме 9 изгибов, в дракон-схеме только 8).

Таким образом, данная блок-схема, как и предыдущие примеры, проигрывает дракон-схеме.

АНАЛИЗ ВЛОЖЕННОГО ЦИКЛА «ПОКА»

Графическое изображение цикла должно быть удобным, стандартным и легко запоминающимся. В блок-схемах эта проблема не решена: нередко каждый автор изображает цикл по-своему, что путает читателей. Язык ДРАКОН предлагает стандартное начертание для каждого типа цикла.

На рис. 184, 185 изображены блок-схема и дракон-схема вложенного цикла ПОКА (while). Блок-схема взята из учебного пособия [74].

На блок-схеме (рис. 184) можно указать следующие недочеты:

- Между иконами *E* и *K* имеет место разрыв шампура (нарушено правило, согласно которому один из путей, идущих от входа к выходу, должен проходить по главной вертикали).
- Ниже иконы *E* через разрыв шампура проходят две нежелательные горизонтальные линии
- Две петли обратной связи циклов ПОКА закручены по часовой стрелке (в дракон-схеме они закручены против часовой стрелки).
- Используются неудобные ромбы (в дракон-схеме их заменяют эргономичные иконы Вопрос).
- Используются 8 стрелок, из которых 6 — паразитные (в дракон-схеме всего 2 стрелки).
- Имеется два избыточных изгиба линии (в блок-схеме 10 изгибов, в дракон-схеме только 8).
- В блок-схемах отсутствует графическая стандартизация циклов. В дракон-схемах стандартизация циклов строго соблюдается. Об этом можно судить по рис. 185. Голубая заливка окружает внутренний цикл ПОКА. Белая заливка окружает внешний цикл ПОКА.

Сравнивая рис. 184, 185 легко убедиться, что дракон-схемы во всех отношениях лучше, чем блок-схемы.

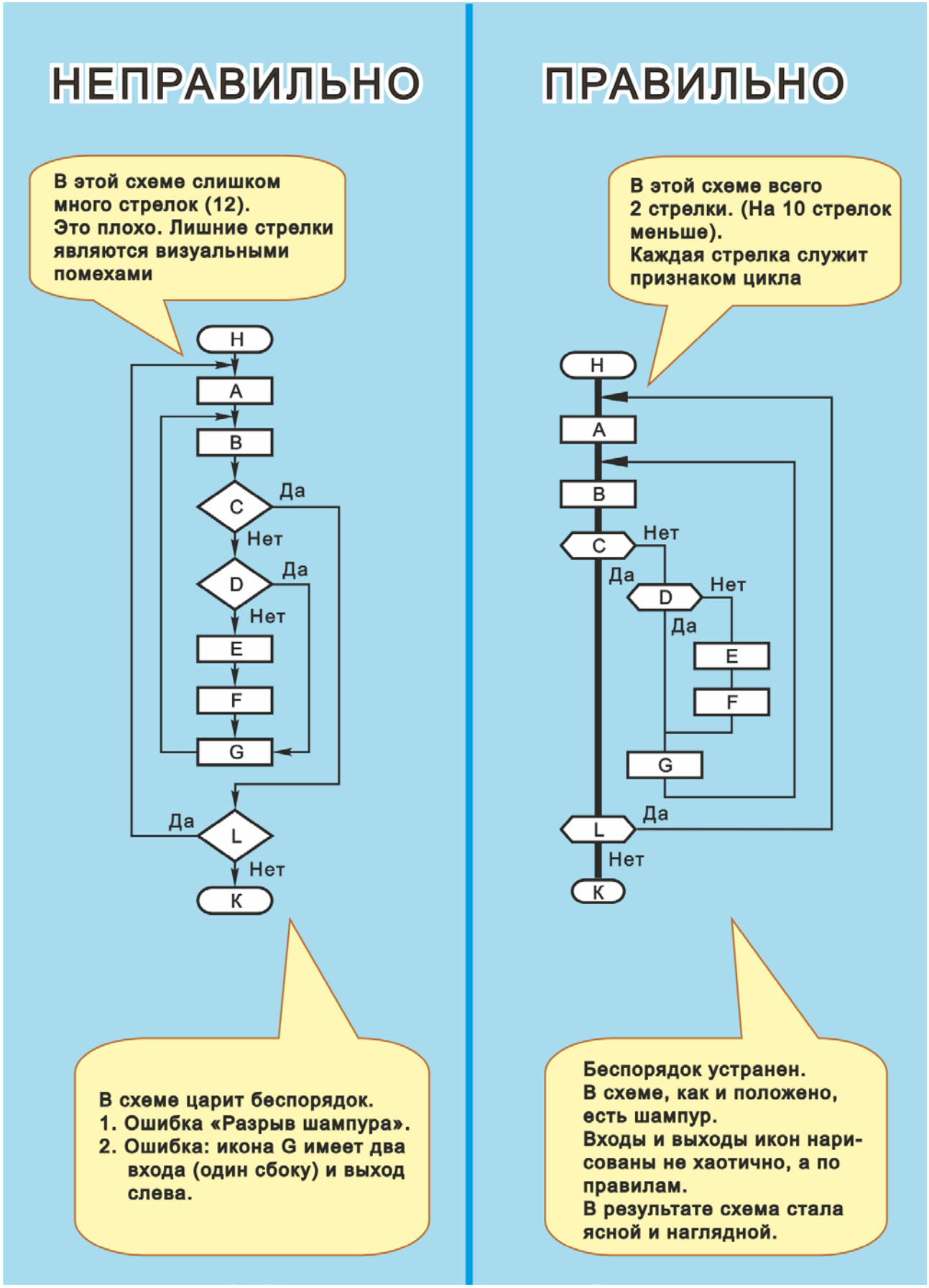


Рис. 182. Плохая схема. В ней много эргономических ошибок, что затрудняет понимание алгоритма

Рис. 183. Хорошая (эргономичная) схема. Она нарисована по правилам языка ДРАКОН

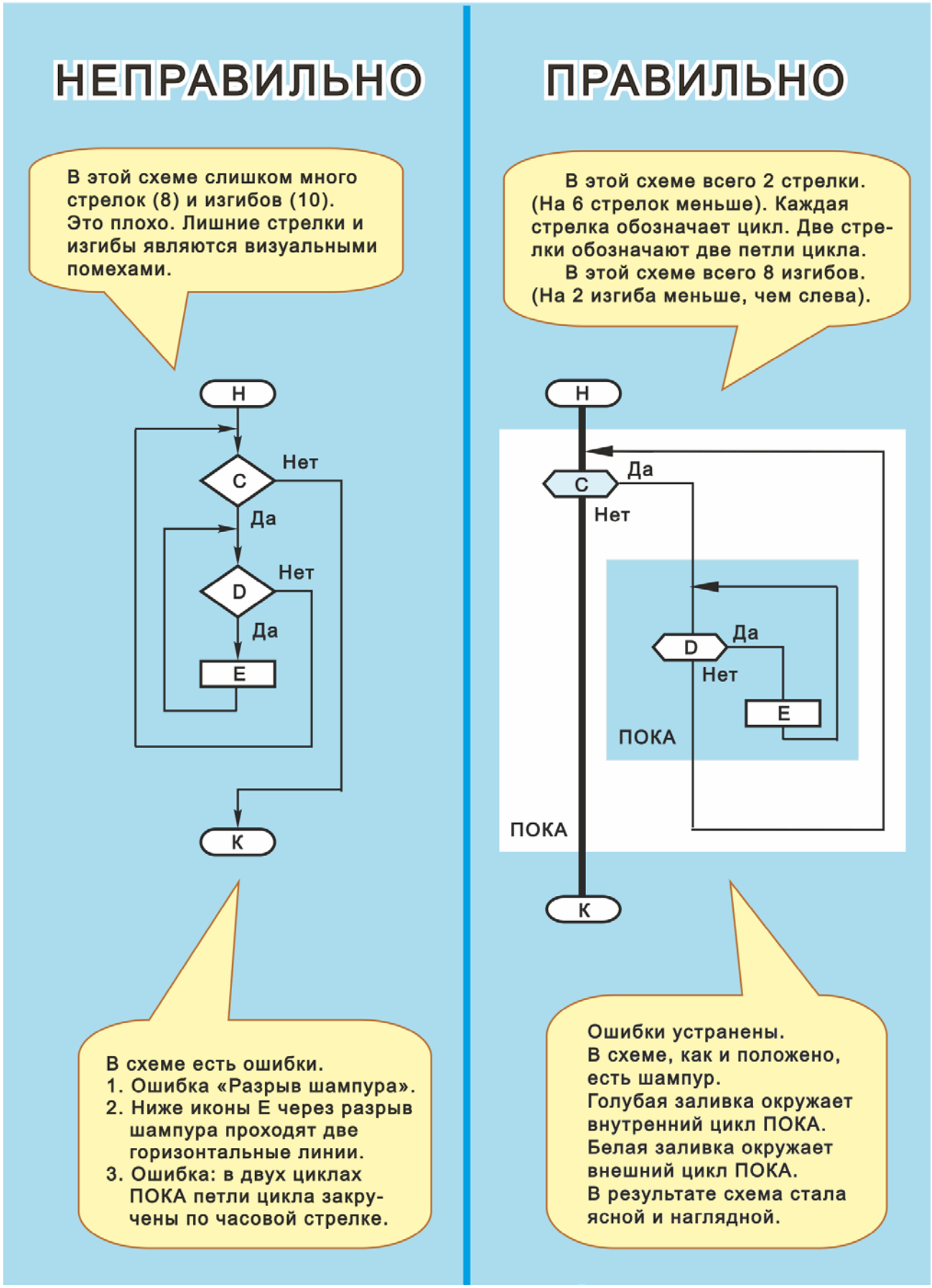


Рис. 184. Плохая схема. Вложенный цикл ПОКА изображен без учета правил эргономики. Шампур разорван.

Рис. 185. Хорошая (эргономичная) схема. Вложенный цикл ПОКА нарисован по правилам языка ДРАКОН

НЕЭРГОНОМИЧНЫЕ «ОБРАЗЦЫ ИТОГОВЫХ ЗАДАНИЙ»

В журнале «Информатика и образование» опубликованы рекомендуемые «образцы итоговых заданий по оценке качества подготовки школьников по информатике» [75].

В материале приведены четыре блок-схемы с блоком «Решение», которые препятствуют использованию шампура. Причина эргономической ошибки состоит в том, что нижний выход ромба удален и нарисован слева (рис. 186). Ошибку можно легко исправить, как показано в дракон-схеме на рис. 187.

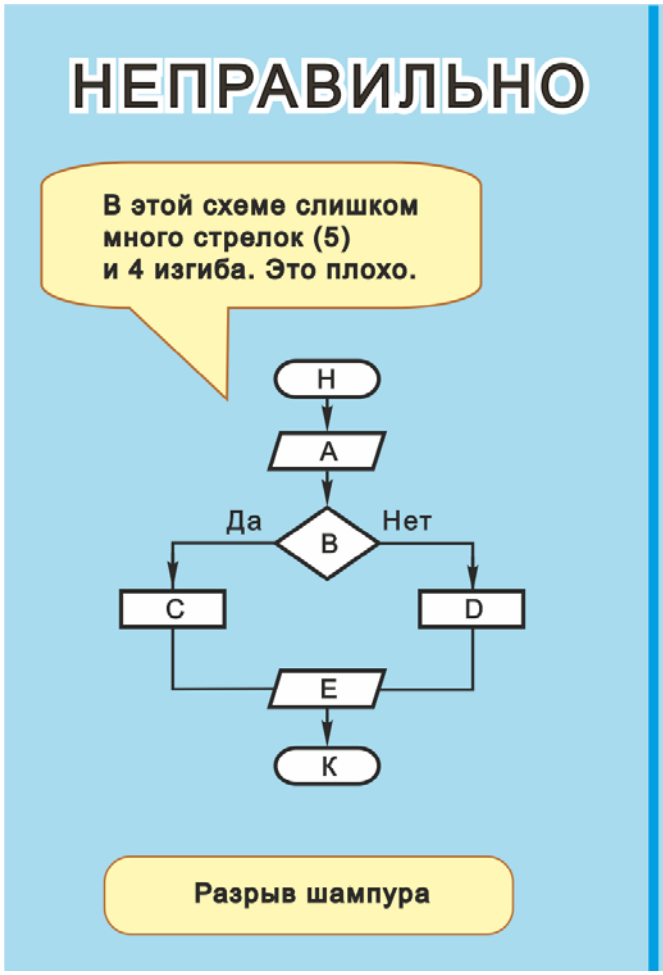


Рис. 186. Плохая схема. Выход из ромба влево (а не вниз) мешает правильному изображению шампура

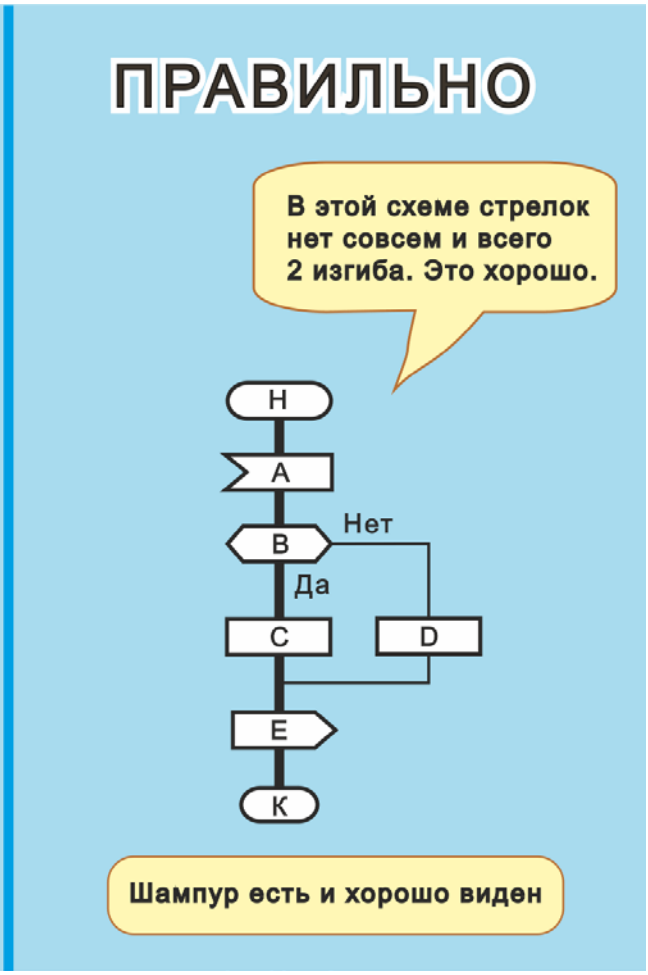


Рис. 187. Хорошая (эргономичная) схема. Шампур нарисован верно, так как икона Вопрос имеет выход вниз.

ТИПИЧНЫЕ ОШИБКИ В БЛОК-СХЕМАХ АЛГОРИТМОВ

На основании анализа алгоритмов на рис. 178-187 можно составить перечень эргономических ошибок, которые мы выявили в этой главе:

- нет шампура;
- разрыв шампура;
- блоки Заголовок и Конец на разных вертикалях;
- ненужные стрелки;
- ненужные изгибы линий;

- ненужные кружки;
- два входа в один блок;
- три входа в один блок;
- вход в блок слева;
- вход в блок справа;
- пересечение линий;
- наклонная линия;
- выход из блока Процесс слева.

ПРИМИТИВ И СИЛУЭТ

В данной главе мы рассмотрели простые блок-схемы, состоящие примерно из 10 блоков. Их можно «вылечить» с помощью дракон-схемы примитив.

В сложных проектах могут использоваться большие многостраничные блок-схемы, насчитывающие 100 и более блоков. В таких случаях работать с блок-схемой становится трудно, поскольку блок-схема полностью теряет и наглядность, и регулярность структуры. Чтобы поправить дело, нужно использовать алгоритмическую конструкцию силуэт, которая сохраняет наглядность даже для многостраничных схем.

ВЫВОДЫ

1. Стандарт ГОСТ 19.701-90 не может обеспечить наглядность при вычерчивании сложных алгоритмов, так как концепция стандарта устарела и построена без учета идей когнитивной эргономики.
2. Дракон-схемы принципиально отличаются от блок-схем тем, что подчиняются когнитивно-эргономическим правилам.
3. В данной главе указаны эргономические недостатки блок-схем и описаны парные им достоинства дракон-схем.
4. В зрительно-смысловой структуре алгоритма шампур играет важную роль. Шампур способен быстро и естественно привлечь к себе внимание читателя, правильно сориентировать его в структуре алгоритма.
5. При отсутствии шампура уничтожается основа для выделения главного маршрута.
6. Разрыв или отсутствие шампура делает зрительный образ алгоритма «бесформенным», лишенным композиционного центра. Читатель лишается необходимых зрительных ориентиров, что затрудняет чтение алгоритма.
7. Зрительный образ алгоритма должен быть лаконичным. Все ненужные, лишние детали должны быть исключены.
8. Схема должна содержать лишь те элементы, которые необходимы читателю для понимания смысла алгоритма и выявления ошибок.
9. Чтобы схема была удобной для чтения, количество изгибов соединительных линий должно быть минимальным.
10. Стрелки нужны только как признак цикла. Стрелки, показывающие направление потока управления, следует удалить.
11. Дракон-схемы созданы на основе блок-схем с целью их совершенствования. Дракон-схемы — это упорядоченные блок-схемы.

Глава 36

КРИТИЧЕСКИЙ АНАЛИЗ БЛОК-СХЕМ АЛГОРИТМОВ ПО ГОСТ 19.701-90

ФОРМАЛИЗАЦИЯ СОЕДИНИТЕЛЬНЫХ ЛИНИЙ

В главе 32 уже отмечалось, что в стандарте на блок-схемы алгоритмов ГОСТ 19.701-90 проведена лишь частичная формализация. Формализованы только фигуры, но не связи между ними. Формализация связей является актуальной задачей.

В языке ДРАКОН формализация выполнена так:

- иконы заданы не отдельно, а вместе с отростками соединительных линий (рис. 19, 20);
- число отростков, тип и направление каждого отростка строго определены.

Существуют три типа отростков:

- входной,
- выходной,
- нейтральный.

Примеры нейтральных отростков приведены на рис. 19, 20, п. 11, 21-23.

Входной и выходной отростки иконы направлены сверху вниз и лежат на одной вертикали, причем входной отросток входит в икону сверху, а выходной — исходит из нее снизу. Икона Вопрос имеет не один, а два выхода; второй выходной отросток направлен по горизонтали вправо.

Перечисленные правила задают однозначную привязку отростков к иконам. Примеры привязки для икон Действие и Вопрос показаны на рис. 188, ***внизу.*** Этим иконам в стандарте ГОСТ 19.701-90 соответствуют блоки «Процесс» и «Решение» (рис. 188, ***вверху***).

Легко видеть, что формализация линий в ГОСТе отсутствует. Действительно, для иконы Действие (рис. 188, ***внизу***) мы видим только один чертеж, а для блока Процесс (***вверху***) — четыре варианта. Больше того, число вариантов может возрасти чуть ли не вдвое, если учесть, что стандарт разрешает выполнять чертежи как со стрелками, так и без них.[5]

Такая же ситуация имеет место для иконы Вопрос (рис. 188, ***внизу***): для нее задан только один чертеж. А для блока Решение (***вверху***) — четыре варианта.

Наличие в стандарте разных способов подключения линий к блокам говорит об отсутствии формализации соединительных линий в блок-схемах алгоритмов.

---

[5] «При необходимости или для повышения удобочитаемости могут быть добавлены стрелки-указатели» [102].

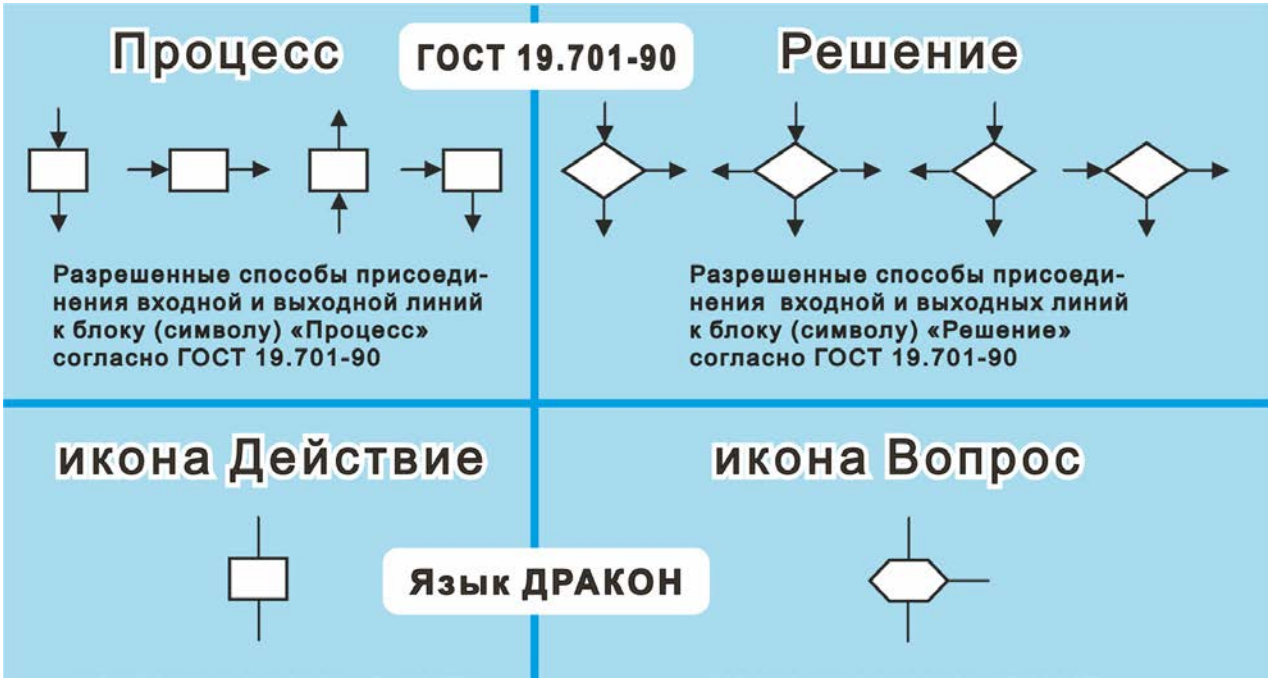


Рис. 188. ГОСТ 19.701-90 (сверху) разрешает присоединять линии к блокам Процесс и Решение четырьмя разными способами, т. е. неоднозначно. Язык ДРАКОН (внизу) устраняет недостаток и предлагает однозначный (единственный) вариант

ФОРМАЛИЗАЦИЯ ТОЧЕК РАЗМЕЩЕНИЯ ИКОН

Валентные точки можно рассматривать как точки размещения икон, или точки ввода икон.

На рисунках 175-177 мы наблюдали построение силуэта. На каждом шаге построения происходит размножение валентных точек. Процесс размножения показан на рис. 189. В аксиоме-силуэт всего 3 валентных точки. После добавления ветки будет уже 5 точек. А после вставки иконы Действие число точек увеличивается до 6 (рис. 189). Дальнейший процесс построения силуэта приводит к монотонному росту числа валентных точек.

Это значит, что иконы (атомы) можно вставлять не куда угодно, а только в строго определенные места. Для каждой валентной точки определен список икон (атомов), которые разрешается вставить в данную конкретную точку.

Ввод атома производится так. Сначала происходит разрыв соединительной линии в выбранной пользователем валентной точке. Затем в место разрыва вставляется атом.

Все списки (списки валентных точек и списки разрешенных икон) хранятся в памяти дракон-конструктора, который осуществляет визуальный логический вывод. Таким образом, описанная операция строго формализована и защищена от многих ошибок.

ФОРМАЛИЗОВАННЫЙ ЧЕРТЕЖ АЛГОРИТМА

Формализованный чертеж алгоритма — это чертеж, для которого определены:

- формальное описание алфавита графических фигур (икон и макроикон);
- формальное описание синтаксиса, то есть соединительных линий между фигурами;

- визуальное логическое исчисление, позволяющее из аксиомы-силуэт и аксиомы-примитив строить формализованный чертеж алгоритма методом логического вывода.

Язык ДРАКОН разработан исходя из этих предпосылок.

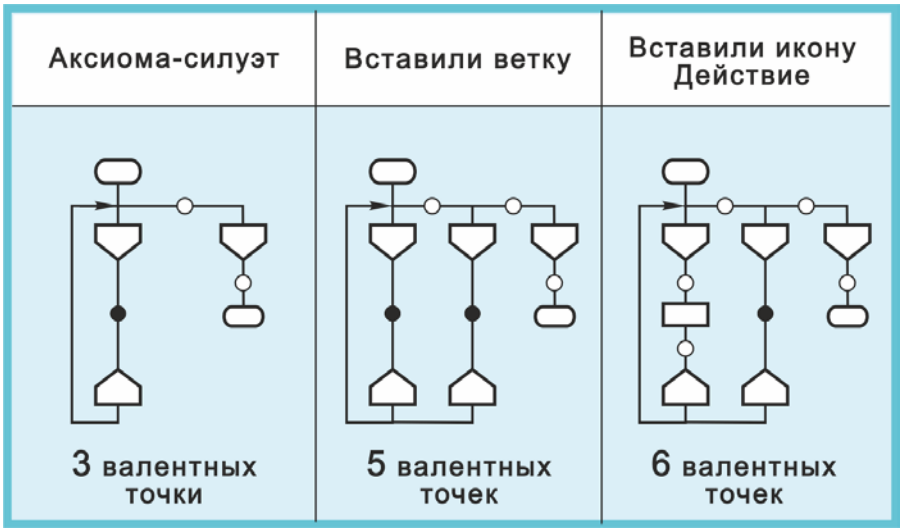


Рис. 189. Размножение валентных точек при проектировании дракон-схемы

КРИТИЧЕСКИЙ АНАЛИЗ БЛОК-СХЕМ АЛГОРИТМОВ

Перейдем к анализу блок-схем алгоритмов, выполненных согласно стандартам ГОСТ 19.701-90 и ISO 5807:1985.

Лесли Робертсон (Австралия) в своем учебнике программирования характеризует блок-схемы как «альтернативный метод представления алгоритмов» и перечисляет их достоинства:

> «Блок-схемы популярны, так как они графически отображают логику программы с помощью стандартных геометрических фигур и соединительных линий. Блок-схемы относительно легко выучить, и они представляют собой интуитивно понятный метод представления управляющей последовательности алгоритма» [76].

Вместе с тем блок-схемы подвергаются критике. Противники блок-схем утверждают, что они не поддаются формализации, поэтому их «нельзя использовать как программу для непосредственного ввода в машину» [77]. Гленфорд Майерс полагает, что они не согласуются со структурным программированием, поскольку в значительной степени ориентированы на использование оператора перехода goto [11, p. 150]. Это подтверждают и другие авторы:

> «Блок-схемы… затемняют особенности программ, созданных по правилам структурного программирования» [78].

> «С появлением языков, отвечающих принципам структурного программирования… блок-схемы стали отмирать» [79].

Существенно, что при большой степени детализации блок-схемы становятся «громоздкими и теряют своё основное достоинство — наглядность структуры алгоритма» [33]. Обозримыми и понятными являются блок-схемы только для небольших алгоритмов [80].

> «Если для простой задачи схемы алгоритмов обеспечивают безусловную наглядность, то по мере роста сложности программы ее логическая структура начинает “тонуть” в “клубке спагетти”, в который постепенно превращается схема алгоритма» [81].

Представление с помощью блок-схем «сложных алгоритмов затруднительно из-за существенно возрастающей громоздкости и быстрой потери наглядности» [15, p. 190].

Трудности понимания сложных блок-схем связаны с тем, что в них отсутствует должный порядок — правила разработки схем не формализованы, не эргономичны и разрешают совершать нежелательные действия:

> «Основной недостаток блок-схем заключается в том, что они не приучают к аккуратности при разработке алгоритма. Ромб можно поставить в любом месте блок-схемы, а от него повести выходы на какие угодно участки. Так можно быстро превратить программу в запутанный лабиринт, разобраться в котором через некоторое время не сможет даже сам ее автор» [79].

Бертран Мейер дает блок-схемам жесткую отрицательную оценку:

> «В свое время блок-схемы были весьма популярны для выражения структуры управления программы. И сегодня их можно встретить для описания процессов, не связанных с программированием. В программировании они потеряли репутацию. (Некоторые авторы называют их не “flowchart”, а “flaw chart” — порочными схемами)…
>
> «Наши программы делают все более сложные вещи. Большие блок-схемы с вложенностью внутри циклов и условных операторов быстро становятся запутанными…
>
> «Аккуратно отформатированный программный текст с отступами, отражающими уровень вложенности, дает лучшее представление о порядке выполнения операторов», чем блок-схемы [55, pp. 205, 206].

Но и это не все. В настоящее время активно развиваются системы управления и робототехника, широко используются алгоритмы реального времени. Однако блок-схемы плохо подходят для таких алгоритмов.

Почему? Потому что правила вычерчивания блок-схем согласно стандартам ГОСТ 19.701-90 и ISO 5807:1985 не имеют выразительных средств и обозначений для описания алгоритмов реального времени.

Чтобы устранить недостатки стандартов и обеспечить работу в реальном времени, в языке ДРАКОН предусмотрены иконы: Пауза, Период, Таймер и Синхронизатор, отсутствующие в блок-схемах.

ДЕЙСТВУЮЩИЙ СТАНДАРТ АЛГОРИТМОВ НЕ ИМЕЕТ НАУЧНОГО ОБОСНОВАНИЯ

Является ли международный стандарт на блок-схемы ISO 5807:1985 научно обоснованным? Тот же вопрос относится и к его отечественному аналогу межгосударственному стандарту ГОСТ 19.701-90 [6] . Последний, в частности, используется в системе образования с целью обучения основам алгоритмизации.

---

[6] В 1992 году государства-члены СНГ заключили соглашение, которым признали действующие стандарты «ГОСТ» СССР в качестве межгосударственных с сохранением аббревиатуры «ГОСТ» за вновь вводимыми межгосударственными стандартами.

Упомянутые стандарты были созданы давно. Они опираются на опыт XX века и отражают исторически сложившийся набор правил выполнения документации по обработке данных. К настоящему времени правила устарели и *не имеют научного обоснования.*

Стандарты на блок-схемы алгоритмов и программ прямо противоречат принципам структуризации блок-схем, которые предложил Эдсгер Дейкстра в классической работе «Заметки по структурному программированию» [82, pp. 25-28].

ЧЕТЫРЕ ПРИНЦИПА СТРУКТУРИЗАЦИИ БЛОК-СХЕМ, ПРЕДЛОЖЕННЫЕ Э. ДЕЙКСТРОЙ

Непосредственный анализ первоисточника [82] со всей очевидностью подтверждает простую мысль. Дейкстрианская «структурная революция» началась с того, что Дейкстра, использовал блок-схемы как инструмент анализа структуры программ и предложил, наряду с другими важными идеями, четыре принципа структуризации блок-схем.

Эти принципы таковы.

1. ***Принцип ограничения топологии блок-схем.*** Структурный подход должен приводить

> «к ограничению топологии блок-схем по сравнению с различными блок-схемами, которые могут быть получены, если разрешить проведение стрелок из любого блока в любой другой блок. Отказавшись от большого разнообразия блок-схем и ограничившись ...тремя типами операторов управления, мы следуем тем самым некоей последовательностной дисциплине» [82, p. 28].

2. ***Принцип вертикальной ориентации входов и выходов блок-схемы.*** Имея в виду шесть типовых блок-схем (if-do, if-then-else, case-of, while-do, repeat-until, следование), Дейкстра пишет:

> «Общее свойство всех этих блок-схем состоит в том, что у каждой из них один вход вверху и один выход внизу» [82, p. 27].

3. ***Принцип единой вертикали.*** Вход и выход каждой типовой блок-схемы должны лежать на одной вертикали.

4. ***Принцип нанизывания типовых блок-схем на единую вертикаль.*** При последовательном соединении типовые блок-схемы следует соединять, не допуская изгибов соединительных линий, чтобы выход верхней и вход нижней блок-схемы лежали на одной вертикали.

Хотя Дейкстра не дает словесной формулировки третьего и четвертого принципов, они однозначно вытекают из имеющихся в его работе иллюстраций [82, pp. 25-28]. Чтобы у читателя не осталось сомнений, мы приводим точные копии подлинных рисунков Дейкстры (рис.190, средняя и левая графа).

УПРАВЛЯЮЩИЙ ГРАФ АЛГОРИТМА

А.А. Тюгашев полагает, что «теоретической основой графического языка блок-схем служит управляющий граф программы» [83]. Это верно лишь отчасти:

- язык блок-схем не удовлетворяет требованиям, предъявляемым к формальному языку, их нельзя подать на вход компилятора. Как отмечает А.Н. Степанов, для конечного исполнителя-компьютера блок-схемы не обеспечивают свойство понятности: «процессорами компьютеров не воспринимаются алгоритмы, заданные в виде блок-схемы» [15, p. 190];

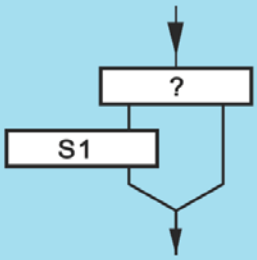
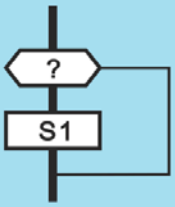
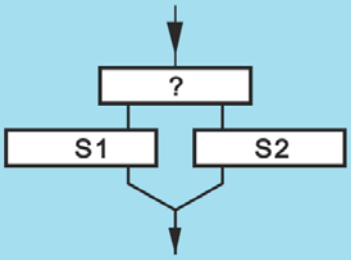
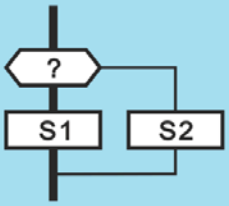
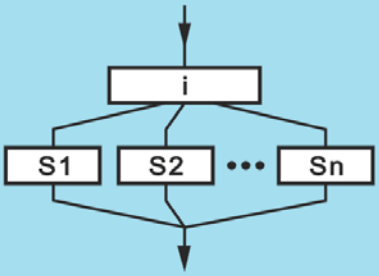
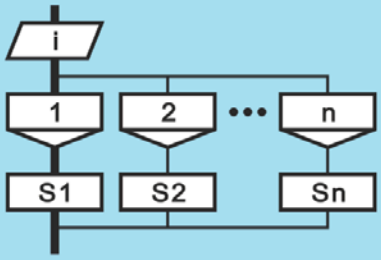
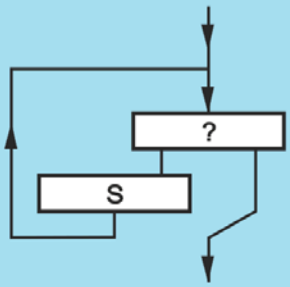
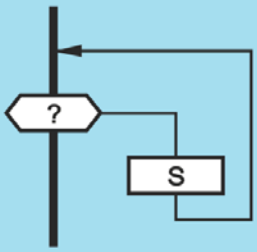
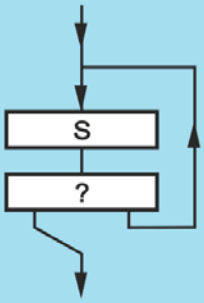
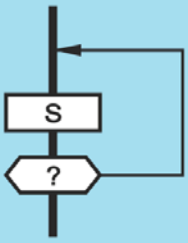
Структурные операторы Дейкстры	Структурные блок-схемы Дейкстры	Дракон-схемы
if ? do S1		
if ? then S1 else S2		
case i of (S1;S2;...Sn)		
while ? do S		
repeat S until ?		

Рис. 190. Структурные блок-схемы Дейкстры (в центре) и эквивалентные им дракон-схемы (справа)

- принцип ограничения топологии блок-схем Дейкстры накладывает принципиальные ограничения на управляющий граф алгоритма *(control flow graph)* и является частью теоретического фундамента блок-схем. В стандартах ГОСТ 19.701-90 и ISO 5807:1985 эти ограничения никак не учтены, что делает указанные стандарты уязвимыми для критики, недостоверными и нелигитимными.

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ЯЗЫКА ДРАКОН

В отличие от блок-схем, теоретической основой языка ДРАКОН служит визуальное логическое исчисление, благодаря чему графический синтаксис ДРАКОНа является не только формальным и безопасным, но и эргономичным, облегчающим выявление слабых мест [39, pp. 393-448].

Кроме того, в языке ДРАКОН — на основе четырех принципов структуризации блок-схем Дейкстры — разработан двумерный структурный подход к алгоритмам и программам (шампур-метод) [39, pp. 449-472]. Метод содержит большое число правил, которые хранит в своей памяти программа дракон-конструктор. Последняя, выполняя черчение дракон-схем, автоматически обеспечивает выполнение правил и не допускает ошибок.

МЕТОД Эдварда АШКРОФТ И Зохара МАННА

Известно, что пересечения соединительных линий в блок-схемах затрудняют их понимание. Поэтому ГОСТ 19.701-90 предписывает: «В схемах следует избегать пересечения линий», что свидетельствует об неумении теоретически решить проблему пересечений.

В отличие от блок-схем, в языке ДРАКОН эта проблема решена; разработан теоретический метод, исключающий пересечения и разрывы линий, а также внутренние соединители. Метод использует введение переменной состояния по Ашкрофту-Манна [84] и Йодану [10, pp. 192-196]. Затем переменная состояния удаляется, превращаясь в идентификаторы икон «Имя ветки» и Адрес [39, pp. 436-448]. Метод реализован на практике во внутренних алгоритмах программы дракон-конструктор.

ВЫВОДЫ

1. Стандарт на блок-схемы алгоритмов ГОСТ 19.701-90 прямо противоречит принципам структуризации блок-схем, которые предложил Эдсгер Дейкстра в классической работе «Заметки по структурному программированию».
2. Принцип ограничения топологии блок-схем Дейкстры накладывает жесткие ограничения на управляющий граф алгоритма (control flow graph) и является частью теоретического фундамента схем алгоритмов.
3. В стандарте ГОСТ 19.701-90 эти ограничения никак не учтены, что делает указанный стандарт уязвимым для критики и нелигитимным.
4. В языке ДРАКОН — на основе четырех принципов структуризации блок-схем Дейкстры — разработан двумерный структурный подход к алгоритмам (шампур-метод).
5. В отличие от блок-схем в языке ДРАКОН решена проблема пересечений соединительных линий: разработан теоретический метод, исключающий

пересечения и разрывы линий. Метод использует введение переменной состояния по Ашкрофту-Манна и Йодану с последующей модификацией.

6. Правила разработки блок-схем алгоритмов по ГОСТ 19.701-90 не формализованы, не эргономичны и разрешают совершать небезопасные действия.
7. В отличие от блок-схем в дракон-схемах проведена:
    - формализация соединительных линий;
    - формализация точек размещения икон;
    - формализация построения графики с помощью визуального логического вывода.
8. Представление с помощью блок-схем сложных алгоритмов вносит неоправданные трудности из-за существенно возрастающей громоздкости и быстрой потери наглядности.
9. Блок-схемы не имеют выразительных средств для представления алгоритмов реального времени, робототехники и систем управления.
10. С появлением дракон-схем (drakon-charts) блок-схемы алгоритмов по ГОСТ 19.701-90 полностью потеряли свое значение, так как они во всех отношениях уступают дракон-схемам.

Глава 37

КАКИМ ДОЛЖЕН БЫТЬ СТАНДАРТ НА АЛГОРИТМЫ

ПРОБЛЕМА СТАНДАРТИЗАЦИИ АЛГОРИТМОВ

Стандарт графического представления алгоритмов (далее стандарт алгоритмов) есть единая система обозначений (единая нотация) для записи алгоритмов. В настоящее время проблема стандартизации алгоритмов не решена. На практике для записи алгоритмов применяются разнообразные средства: псевдокоды, блок-схемы (flowcharts), схемы деятельности (activity diagrams) языка UML, дракон-схемы (drakon-charts).

В программировании (с появлением структурного программирования) подробные блок-схемы алгоритмов стали ненужными; вместо них используются псевдокоды, как правило, не подлежащие стандартизации.

При выпуске документации на алгоритмы действует межгосударственный стандарт ГОСТ 19.701-90, полученный методом прямого применения из международного стандарта ISO 5807:1985.

Проблема в том, что стандарты ГОСТ 19.701-90 и ISO 5807:1985 обладают существенными недостатками; они не удовлетворяют требованиям для записи алгоритмов.

ТРЕБОВАНИЯ К СТАНДАРТУ АЛГОРИТМОВ

Существующие нотации и алгоритмические языки не предотвращают алгоритмические ошибки, а наоборот, содействуют их появлению, являясь своеобразным катализатором ошибок. Новая нотация должна в максимальной степени содействовать сокращению числа ошибок в алгоритмах.

Проблема ошибок является чрезвычайно важной и актуальной; поэтому предотвращение ошибок должно быть предусмотрено на уровне стандарта для записи алгоритмов.

ЧТО ЛУЧШЕ ДЛЯ РОССИЙСКОГО ОБРАЗОВАНИЯ: ДРАКОН-СХЕМЫ ИЛИ БЛОК-СХЕМЫ ПО ГОСТ 19.701-90?

Многие авторы высказываются в поддержку языка ДРАКОН, например:

> «Визуальный язык ДРАКОН образует наглядную среду для первоначального обучения программированию и мог бы быть весьма полезен при организации школьных курсов информатики» [85].
>
> «Блок-схемы, нарисованные по правилам языка ДРАКОН, отличаются поразительной четкостью, наглядностью и прозрачностью структуры. А

> наглядность и доходчивость алгоритмов — это именно то, чего так остро недостает школьным учебникам» [86].
>
> «ДРАКОН — это… эргономичный стандарт для графического представления учебной информации… Язык ДРАКОН учит нас, методистов и учителей правильному составлению блок-схем… Насколько я знаю, нет другой литературы, где тому же самому можно научиться настолько просто и даже увлекательно» [87].
>
> «Язык усовершенствованных графических схем ДРАКОН обеспечивает разработку сложных алгоритмов с сохранением наглядности даже для многостраничных схем» [88].
>
> «Алгоритмический язык ДРАКОН… используется в технике, биологии, медицине и образовании. Преимуществом этого языка является то, что схемы легко рисовать и понимать, они очень наглядны» [89].
>
> «Язык ДРАКОН – удобный инструмент для записи и структурирования деятельности в виде алгоритмов…, даёт глубокое понимание сложных проблем, позволяет проектировать сложную деятельность, бизнес-процессы, формализовать свои профессиональные знания» [90].
>
> «Использование языка ДРАКОН и гибридных языков может позволить полностью отказаться от традиционного подхода к разработке ПО РК [программного обеспечения робототехнических комплексов], снижая интеллектуальную нагрузку на разработчика алгоритма, исключая ошибки толкования исходных данных программистом… Но самое главное — это позволит резко сократить затраты на разработку ПО РК, что сделает роботов более доступными для потребителей самого разного уровня» [91].

Приведем также отзыв рядового пользователя, размещенный в сети интернет участником с ником dvuugl:

> «Если нужно рисовать алгоритм, теперь только и только на Драконе. Считаю, что он должен стать государственным стандартом для блок-схем вместо существующего. Удивительно, что авторы книг продолжают использовать прежние схемы, на которые после Дракона без ужаса смотреть невозможно».

С другой стороны, несмотря на явное преимущество дракон-схем, в большинстве российских школ и вузов язык ДРАКОН не изучают, предпочитая устаревшие блок-схемы. Причина проста. Блок-схемы опираются на авторитет стандартов ГОСТ 19.701-90 и ISO 5807:1985, а дракон-схемы такой поддержки не имеют. Многие преподаватели вузов и учителя школ продолжают знакомить студентов и школьников с неэргономичными блок-схемами, оправдывая свои действия необходимостью соблюдать требования стандарта ГОСТ 19.701-90.

СТАНДАРТЫ, КОТОРЫЕ ОТСТАЛИ ОТ ЖИЗНИ

Проведенный анализ позволяет сделать вывод, что морально устаревший стандарт ISO 5807:1985 (и его калька ГОСТ 19.701-90) препятствуют распространению новых, более удобных и эффективных форм представления визуальных алгоритмов. Указанные стандарты оказывают негативное воздействие на систему среднего и высшего образования России.

Таким образом, налицо парадоксальная ситуация. Система образования должна распространять лучшее, а не худшее. Однако на деле происходит обратное.

Российским преподавателям, учащимся и специалистам навязан устаревший и не имеющий научного обоснования стандарт только потому, что он скопирован с международного стандарта. Возникло противоречие между устаревшим стандартом и новой практикой.

Это противоречие следует устранить. Учитывая вышеизложенное, необходимо отказаться от некачественных стандартов ISO 5807:85 и ГОСТ 19.701-90 и в качестве государственного стандарта разработать стандарт, основанный на языке ДРАКОН. Как отмечает профессор Я. В. Безель в журнале «Вестник Российской академии наук»:

> «при разработке единого стандарта [на блок-схемы], снабженного компьютерной поддержкой и рассчитанного на постепенное внедрение во всех отраслях и предметных областях, целесообразно взять за основу язык ДРАКОН» [92].

ЯЗЫК ДРАКОН УСТРАНЯЕТ НЕДОСТАТКИ БЛОК-СХЕМ

ДРАКОН упорядочивает блок-схемы за счёт формализации, эргономизации и неклассической структуризации [39] [93]. С появлением дракон-схем (drakon-charts) блок-схемы алгоритмов по ГОСТ 19.701-90 полностью потеряли свое значение, так как они ***во всех отношениях уступают дракон-схемам*** [39, pp. 32-36, 242-254].

В 1996г. Государственный комитет Российской Федерации по высшему образованию по решению Президиума научно-методического совета по информатике под председательством академика РАН Ю. И. Журавлева включил изучение языка ДРАКОН в Примерную программу дисциплины «Информатика» для бакалавров для направлений:

510000 – Естественные науки и математика,
540000 – Образование,
550000 – Технические науки,
560000 – Сельскохозяйственные науки [4].

А.Н. Степанов в «Курсе информатики для студентов информационно-математических специальностей» (2018) отмечает:

> «Существуют близкие к блок-схемам языки визуального программирования, такие как… язык программирования и моделирования ДРАКОН. В этом языке используются графические элементы, аналогичные стандартным элементам блок-схем… Но для обеспечения возможности автоматического преобразования графической программы в машинный язык введены строгие правила задания графических и текстовых элементов такой программы» [15, p. 190].

Далее Степанов излагает концепцию языка ДРАКОН и указывает его преимущества:

> «В рамках этого подхода основные управляющие конструкции следования, ветвления и цикла, которые в обычных алгоритмических языках задаются с помощью служебных слов, таких, как *begin, end, if, then, else, while, do* и т. д., заменяются управляющей графикой, похожей на стандартные элементы блок-схем…
>
> «Язык двумерного структурного программирования ДРАКОН является полным по Тьюрингу и относится к универсальным языкам программирования… Использование гибридных языков позволяет отказаться от текстовых управляющих структур, используемых в обычных языках, и заменить их управляющей графикой языка ДРАКОН. Написание

программы становится более понятным и удобным для человека, повышается производительность труда программистов» [15, pp. 1017-1019].

СЛЕДУЕТ РАЗЛИЧАТЬ АЛГОРИТМЫ И ПРОГРАММЫ

В литературе термины *алгоритм* и *программа* нередко используются как синонимы. Однако при создании стандарта это недопустимо; их необходимо строго различать.

Стандарт ГОСТ 19.701-90 нарушает это правило; он называется «Схемы алгоритмов, программ, данных и систем» и не проводит границы между алгоритмами и программами.

ТЕЗИС АКАДЕМИКА ДОРОДНИЦЫНА

Академик А. А. Дородницын четко проводит указанную границу, подчеркивая, что «без алгоритмов предмета информатики не существует» [94]. Более того, он предлагает выделить «алгоритмические средства» как отдельную, самостоятельную сущность:

> «…состав информатики — это три неразрывно и существенно связанные части: технические средства, программные средства и алгоритмические средства. Если о первых двух частях никогда не забывают — … они получили специальные термины «hardware» и «software», — то алгоритмическая часть информатики остается почему-то в тени… об этой важнейшей части информатики просто забывают» [94].

Таким образом, согласно Дородницыну, ***алгоритмические средства должны составлять третью, самоценную часть информатики, наряду с программными средствами (software) и техническими средствами (hardware)*** [94].

Чтобы в полной мере реализовать идею Дородницына, нужно иметь отдельный стандарт, посвященный алгоритмам, содержащий эргономичную нотацию для записи алгоритмов, пригодную для подавления ошибок.

ВЫВОДЫ

1. Блок-схемы алгоритмов (flowcharts), описанные в ГОСТ 19.701-90 и международном стандарте ISO 5807:85, обладают серьезными дефектами. Пользоваться ими недопустимо.
2. Вместо блок-схем для записи алгоритмов следует использовать дракон-схемы (drakon-charts).
3. Необходимо разработать и выпустить отдельный стандарт, посвященный алгоритмам, на основе языка ДРАКОН.
4. В 1996 году Государственный комитет РФ по высшему образованию по решению Президиума научно-методического совета по информатике включил изучение языка ДРАКОН в программу курса «Информатика».
5. Тем не менее преподаватели вузов и учителя школ продолжают знакомить студентов и школьников с морально устаревшими блок-схемами, обосновывая такую практику необходимостью ориентироваться на действующий стандарт ГОСТ 19.701-90.

6. Преподавателям и учителям можно рекомендовать ознакомиться с языком ДРАКОН, убедиться в его преимуществах и организовать изучение языка ДРАКОН на лекциях, уроках, практических занятиях, курсовых и дипломных работах.

Заключение

АЛГОРИТМЫ — ВАЖНАЯ ЧАСТЬ ЧЕЛОВЕЧЕСКОЙ КУЛЬТУРЫ

КРИТИКА ТРАДИЦИОННЫХ ПОДХОДОВ

Цель книги — облегчить изучение алгоритмов и сделать их доступными для широкой аудитории. Стандартный курс алгоритмов чрезмерно сложен и подходит далеко не всем — его понимают лишь программисты и математики. Все остальные остаются не у дел и страдают от отсутствия алгоритмических знаний.

Можно ли сломать эту преграду? Можно ли облегчить работу с алгоритмами? Можно ли расширить круг посвященных и сделать алгоритмы, образно говоря, доступными для народа?

Да, можно, если заменить текст на хорошую и наглядную графику.

Традиционные формы представления алгоритмов отжили свой век и должны сойти со сцены. Именно они несут ответственность за господствующую на нашей планете алгоритмическую неосведомленность.

НОВАЯ НОТАЦИЯ

Предложен новый способ записи алгоритмов — дракон-схемы. Благодаря этому новшеству алгоритмы становятся доходчивыми, ясными, прозрачными. Дракон-схемы облегчают изучение алгоритмов и работу с ними.

Новый способ записи дает возможность коренным образом изменить систему образования в области алгоритмизации. И познакомить с алгоритмами более широкие слои населения.

Использование дракон-схем устраняет неоправданные трудности при знакомстве с алгоритмами, позволяет стимулировать алгоритмическое мышление и ликвидировать алгоритмическую безграмотность.

МАТЕМАТИКА И ЭРГОНОМИКА

Алгоритм — слуга двух господ: математики и эргономики. Математика обеспечивает строгость и точность алгоритмов, а когнитивная эргономика делает их удобочитаемыми, облегчает восприятие.

Существующий способ записи алгоритмов должен претерпеть значительные изменения и повернуться лицом к эргономике. В книге почти 200 рисунков, в которых показаны образцы эргономичных алгоритмов. Правила их построения подробно описаны.

Эргономичные алгоритмы отличаются от обычных, как небо от земли. Обычные алгоритмы трудны для постижения и анализа, а эргономичные, наоборот, воспринимаются легко и почти без усилий.

СТАНДАРТ, КОТОРЫЙ ОТСТАЛ ОТ ЖИЗНИ

Действующий стандарт ГОСТ 19.701-90 определяет порядок выпуска документации на алгоритмы и используется в системе образования.

Парадокс в том, что данный стандарт не имеет научного обоснования. Он прямо противоречит принципам структуризации блок-схем, которые предложил Эдсгер Дейкстра в классической работе «Заметки по структурному программированию». В стандарте ГОСТ 19.701-90 эти принципы никак не учтены, что делает указанный стандарт уязвимым для критики и нелегитимным.

В языке ДРАКОН, напротив, все четыре принципа структуризации блок-схем Дейкстры аккуратно соблюдаются; на их основе разработан двумерный структурный подход к алгоритмам и программам.

НУЖЕН СТАНДАРТ, ОСНОВАННЫЙ НА ЯЗЫКЕ ДРАКОН

Показано, что блок-схемы алгоритмов по ГОСТ 19.701-90 по всем параметрам хуже, чем дракон-схемы. Блок-схемы учат беспорядку и хаосу, а язык ДРАКОН наводит порядок и изгоняет хаос. Морально устаревший стандарт ГОСТ 19.701-90 препятствует распространению новых, более удобных и эффективных форм представления визуальных алгоритмов.

Российским преподавателям, учащимся и специалистам навязан устаревший и не имеющий научного обоснования стандарт только потому, что он скопирован с международного стандарта ISO 5807:85. Возникло противоречие между устаревшим стандартом и новой практикой.

Это противоречие следует устранить. Необходимо отказаться от некачественного стандарта на алгоритмы ГОСТ 19.701-90 и вместо него в качестве государственного стандарта разработать стандарт, основанный на языке ДРАКОН.

ПОДАВЛЕНИЕ ОШИБОК

Стандарт ГОСТ 19.701-90 не обеспечивает защиту от ошибок. Ромб можно поставить в любом месте блок-схемы, а выходы соединить с любым участком. Подобная вседозволенность противоречит принципу ограничения топологии блок-схем Дейкстры и является источником ошибок.

Язык ДРАКОН запрещает подобную практику и предоставляет мощные средства для подавления ошибок. При работе дракон-конструктора защита от ошибок действует автоматически, без дополнительных затрат труда, времени и ресурсов.

Текстовые операторы управления, описывающие циклы и ветвления, заменяются безошибочной графикой. Ошибки могут сохраниться лишь на линейных участках программы, где их можно легко обнаружить и устранить.

АЛГОРИТМЫ И ЖИЗНЕРИТМЫ

Понятие алгоритма изучено не полностью. Теория алгоритмов распространяется лишь на строгие алгоритмы и упускает из виду обширный класс нестрогих алгоритмов, который включает медицинские алгоритмы, описывающие действия врачей при профилактике, диагностике, лечении, а также потоки работ (workflows), бизнес-процессы, стандартные операционные процедуры.

Для обозначения нестрогих алгоритмов введен термин «жизнеритмы». Для разграничения алгоритмов и жизнеритмов предложен критерий Ланды-Непейводы: жизнеритмы внешне похожи на алгоритмы, но имеют дефект — некоторые шаги жизнеритма не полностью определены.

Жизнеритмы играют важную роль в науке, технике, образовании, культуре и других областях.

ПОВЫСИТЬ КАЧЕСТВО ЖИЗНЕРИТМОВ

Жизнеритмы незаменимы во многих сферах жизни общества, но у них не решена проблема качества: в отличие от алгоритмов они не обладают детерминированностью, точностью и однозначностью.

Язык ДРАКОН позволяет решить проблему: предложена теория жизнеритмов, созданы эффективные средства для их записи, стандартизации и предотвращения ошибок.

ГДЕ СКАЧАТЬ ДРАКОН-КОНСТРУКТОР

Ответ дан в главе 34.

Как связаться с автором

Владимир Данилович Пароджанов
Mobile: +7-916-111-91-57
Viber: +7-916-111-91-57
E-mail: vdp2007@bk.ru
Skype: vdp2007@bk.ru
Website: http://drakon.su/
Webforum: http://forum.drakon.su/

СПИСОК ЛИТЕРАТУРЫ

[1] Паронджанов В.Д. Визуальный алгоритмический язык ДРАКОН в ракетной технике и медицине // Современные автоматизированные системы управления реального времени как прикладное развитие научных достижений кибернетики», (К 100-летию со дня рождения И.А. Полетаева). Материалы межведомственной конференции 24 марта 2016 г. — ФГБУ «3 ЦНИИ» Минобороны РФ, 2016. — 218 с. — С. 57-78. — https://bit.ly/2BDhYCB .

[2] Морозов В. В., Трунов Ю. В., Комиссаров А. И. и др. Система управления межорбитального космического буксира «Фрегат» \ Вестник ФГУП «НПО им. С. А. Лавочкина», 2014, № 1. — С. 16-25. — https://bit.ly/2RtCdaU.

[3] Вирт Н. Систематическое программирование. Введение. Пер. с англ. / Под ред. Ю.М. Баяковского. – М.: Мир, 1977. — 184 с. — С. 15.

[4] Примерная программа дисциплины «Информатика». Издание официальное. — М.: Госкомвуз, 1996. — С. 3, 4, 15, 16. — 21 с. — https://drakon.su/_media/biblioteka/progr_drakon.pdf.

[5] Цит. по: Гусев В. А. Теория и методика обучения математике: психолого-педагогические основы. — М. : БИНОМ. Лаборатория знаний, 2013. — 456 с. — С. 8.

[6] Карасева Т.В. Сборник задач и упражнений по основам информатики и вычислительной техники. — М.: Колледж, 1994. — 128 с. — С. 61.

[7] Клини С. Введение в метаматематику. / Пер. с англ. А.С. Есенина-Вольпина / Под ред. В.А. Успенского. — Изд-во иностранной лит-ры, 1957. — 526 с. — С. 125.

[8] Кондаков Н.И. Логический словарь-справочник. 2-е изд., испр. и доп. — М.: Наука, 1976. — 720 с. — С. 29, 600.

[9] Агарева О.Ю., Селиванов Ю.В. Математическая логика и теория алгоритмов: учеб. пособие. — М.: МАТИ, 2011. — 80 с. — С. 26.

[10] Йодан Э. Структурное проектирование и конструирование программ. / Пер. с англ. / Под ред. Л.Н. Королева. — М.: Мир, 1979. – 415 с. — С. 252.

[11] Майерс Г. Надежность программного обеспечения / Пер с англ. / Под ред. В.Ф. Кауфмана. — М.: Мир, 1980. — 360 с. — С. 292.

[12] Руководство по профилактической медицине. — М.: Новая слобода, 1993. – С. 40.

[13] Практическое руководство для врачей общей (семейной) практики / Под ред. академика РАМН И.Н. Денисова. — М.: Геотар-Мед, 2001. – С. 501–504.

[14] Марценюк В. Б. Книга, которая учит, как вырастить помидоры. Учебное пособие. — Пермь: ПС Гармония, 2002. – С. 7–21.

[15] Степанов А.Н. Курс информатики для студентов информационно-математических специальностей. (Серия «Учебник для вузов»). — СПб.: Питер, 2018. — 1088 с.

[16] Основы информатики и вычислительной техники. Проб. учеб. пособие для сред. учеб. заведений. В 2 ч. Ч. 1. / А.П. Ершов, В.М. Монахов, С.А. Бешенков и др. / Под ред. А.П. Ершова, В.М. Монахова. — 2 изд. — М.: Просвещение, 1988. — 96 с. — С. 17.

[17] Паронджанов В.Д. Почему врачи убивают и калечат пациентов, или Зачем врачу блок-схемы алгоритмов? Иллюстрированные алгоритмы диагностики и лечения — перспективный путь развития медицины / Предисл. члена-корр. РАН Г.В. Порядина. — М.: ДМК Пресс, 2017. 340с.

[18] Криницкий Н.А. Алгоритмы вокруг нас. — М.: Наука, 1984. — 224 с. — С. 6.

[19] Ланда Л. Н. Алгоритмизация в обучении. / Под общей ред. и со вступительной статьей Б. В. Гнеденко и Б. В. Бирюкова. — М.: Просвещение, 1966. — 523 с.

[20] Москалева И.С. Обучение студентов педвуза проектированию образовательного процесса // Российское педагогическое образование в условиях модернизации: Сб. науч. труд. 9 заоч. науч.-метод. конф. – Саратов: Центр Наука, 2013. – 332 с. — С. 203.

[21] Железняков А.В. Совершенствование подготовки специалистов с использованием активных методов обучения // Высшее техническое образование: проблемы и пути развития. Мат. VIII науч.-метод. конф. 17-18 ноября 2016. В 2 ч. Ч. 1. — Минск: БГУИР, 2016. – С. 151.

[22] Ковальчук М., Михайленко Л. Алгоритмическая культура как компонент алгоритмической деятельности // Knowledge, Education, Law, Management 2018 № 1 (21).

[23] Евсельева Г.В. Использование алгоритмических предписаний на уроках физики – средство активизации познавательной деятельности учащихся // Успехи современного естествознания. – 2013. – № 10 – С. 139-139.

[24] Хизбуллина Р.З., Салихова И.К., Бакиева Э.В. Роль и значение алгоритмического предписания при решении географических задач учениками // Символ науки. №12-2/2016. — С. 257.

[25] Колдаев В.Д. Основы алгоритмизации и программирования: учебное пособие / под ред. проф. Л.Г. Гагариной. — М.: Форум, Инфра-М, 2009. — 416 с. — С. 19.

[26] Computational thinking // Wikipedia — https://en.wikipedia.org/wiki/Computational_thinking.

[27] Газейкина, А.И. Стили мышления и обучение программированию студентов педагогического вуза // Информационные технологии в образовании. 2006. — http://ito.edu.ru/2006/Moscow/I/1/I-1-6371.htm.

[28] Цит. по: Рогожкина И.Б. Развивающий эффект обучения программированию: психолого-педагогические аспекты // Психология. Журнал Высшей школы экономики, 2012. Т. 9, № 2. С. 141–148.

[29] Филатова Л.Ю., Филатова А.С. Развитие алгоритмического стиля мышления при обучении студентов вуза // Наука ЮУрГУ: материалы 67-й научной конференции. Секции естественных наук. — С. 469-472.

[30] Медицина в афоризмах и крылатых выражениях от истоков до наших дней. / Ачкасов Е. Е., Мискарян И. А., – М.: «Профиль-2С», 2009. – 448 с. – С. 198.

[31] Гусев С. Д. Алгоритмы и блок-схемы в здравоохранении и медицине : учеб. пособие. – Красноярск: КрасГМУ, 2018. — 122 с. — https://bit.ly/2Vr4eT8.

[32] Алгоритмы диагностики и лечения в хирургии / Мак-Интайр Р. Б., Стигманн Г. В., Айсман Б. Перевод с английского под ред. акад. В.Д. Федорова, член-корр. В. А. Кубышкина. – М.: ГЭОТАР-Медиа, 2009. – 744 с.

[33] Семёнов Н. М. Программирование и основы алгоритмизации. Учебное пособие. — Томск: Томский политехнический университет, 2009. — С. 71. — 90 с.

[34] Порядин Г.В. Предисловие. Перспективы развития медицины и медицинского образования // Пароджанов В.Д. Почему врачи убивают и калечат пациентов, или Зачем врачу блок-схемы алгоритмов? — М.: ДМК Пресс, 2017. — 340 с. — С. 16-20.

[35] To Err is Human: Building a Safer Health System / Linda T. Kohn, Janet M. Corrigan, and Molla S. Donaldson, editors. – Committee on Quality of Health Care in America, Institute of Medicine. 1999, 2000. – 312 p. – http://nap.edu/9728.

[36] Hayward R.A., Hofer T.P. Estimating Hospital Deaths Due to Medical Errors: Preventability Is in the Eye of the Reviewer // JAMA: the Journal of the American Medical Association. — July 25, 2001, Vol. 286, No. 4. — pp. 415–420.

[37] Dzau Victor J. A Call to Action // 2nd Global Ministerial Summit – A Global Movement on Patient Safety. 29-30 March 2017, Bonn, Germany — https://bit.ly/2NUgYBW.

[38] Пароджанов В. Д. Дружелюбные алгоритмы, понятные каждому. Как улучшить работу ума без лишних хлопот. — М.: ДМК-пресс, 2010, 2014, 2016. — 464 с.

[39] Пароджанов В. Д. Учись писать, читать и понимать алгоритмы. Алгоритмы для правильного мышления. Основы алгоритмизации. — М.: ДМК Пресс, 2012, 2014, 2016. — 520 с.

[40] Космический ДРАКОН. Как заброшенный проект "Роскосмоса" подарил язык литовской медицине // Николай Воронин, корреспондент Русской службы BBC по науке. — https://bbc.in/2YPqGtC.

[41] Космический ДРАКОН. Как заброшенный проект "Роскосмоса" подарил язык литовской медицине // Николай Воронин, корреспондент по вопросам науки, 5 августа 2019. — https://www.bbc.com/russian/features-48583773.

[42] Заключение Федерального учебно-методического объединения в системе высшего образования по укрупненной группе специальностей и направлений подготовки «Клиническая медицина». — https://drakon.su/_media/otvet_iz_minzdrava.pdf.

[43] Пароджанов В. Д. Как улучшить работу ума (новые средства для образного представления знаний, развития интеллекта и взаимопонимания). — М.: Радио и связь, 1998, 1999. — 352 с.

[44] Пароджанов В. Д. Как улучшить работу ума. Алгоритмы без программистов — это очень просто. — М.: Дело, 2001. — 360 с.

[45] Бондарев П. А., Колганов С. К. Основы искусственного интеллекта. – М.: Радио и связь, 1998. – 128с. – С. 65.

[46] Джорджеф М. П., Лэнски Э. Л. Процедурные знания // ТИИЭР. 1986. Т. 74. № 10. — С. 101.

[47] wikiHow — https://www.wikihow.com.

[48] Ньютон, И. Всеобщая арифметика или книга об арифметическом синтезе и анализе. — М. : Изд-во Академии наук СССР, 1948. — 444 с. — С. 243.

[49] Тео Леггетт. Что случилось в кабине "Боинга"? // Русская служба BBC. 17 мая 2019 года — https://www.bbc.com/russian/resources/idt-sh/boeing_two_deadly_crashes_russian.

[50] Maneuvering Characteristics Augmentation System // Wikipedia. — https://en.wikipedia.org/wiki/Maneuvering_Characteristics_Augmentation_System.

[51] Boeing 737 MAX groundings // Wikipedia. — https://en.wikipedia.org/wiki/Boeing_737_MAX_groundings.

[52] Maneuvering Characteristics Augmentation System # Certification inquiry on 737 MAX // Wikipedia — http://bit.ly/2XHE9US.

[53] Фокс Дж. Программное обеспечение и его разработка. / Пер. с англ. / Под ред. Д.Б. Подшивалова. — М.: Мир, 1985. — 368 с. — С. 241.

[54] Dijkstra E. Go To Statement Considered Harmful // Communications of the ACM, Volume 11, No. 3, March 1968, pp. 147-148. — Association for Computing Machinery, Inc.

[55] Мейер Б. Почувствуй класс. Учимся программировать хорошо с объектами и контрактами. Перевод с англ. / Под ред. В.А. Биллига. — М.: ИНТУИТ, БИНОМ, 2011. — 775 с.

[56] Вельбицкий И.В. Графическое программирование и доказательство правильности программ. — 9th IEEE Computer Science & Information Technologies Conference, Armenia, Yerevan, 2013 http://glushkov.org/wp-content/131120CSITrus.pdf.

[57] Дал У., Дейкстра Э., Хоор К. Структурное программирование. / Пер с англ. / Под ред. Э.З. Любимского, В.В. Мартынюка. — М.: Мир, 1975. — 247 с.

[58] Кауфман В.Ш. Языки программирования. Концепции и принципы. — М.: ДМК Пресс, 2010. — 464 с. — С. 431, 432.

[59] Bohm, Corrado; and Giuseppe Jacopini (May 1966). «Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules». Communications of the ACM 9 (5): pp. 366–371.

[60] Harlan D. Mills Mathematical Foundations for Structured Programming. — February 1972. — Federal Systems Division. IBM Corporaton. Gaithersburg, Maryland. pp. 31-39. — http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1055&context=utk_harlan.

[61] Wagner F., Schmuki R., Wagner T., Wolstenholme P. Modeling Software with Finite State Machines. — New York: Auerbach Publications, 2006. — P. VI.

[62] Босуэлл Д., Фаучер Т. Читаемый код или Программирование как искусство. — СПб.: Питер, 2012. — 208 с.

[63] Тюгашев А.А. Основы программирования. Часть I. Учебное пособие. – СПб.: Университет ИТМО, 2016. – 160 с. — 121.

[64] Ершов Ю. Л., Палютин Е. А. Математическая логика. — М.: Наука, 1979. — 320 с. — С. 12, 13.

[65] Поспелов Г. С. Искусственный интеллект — основа новой информационной технологии. — М.: Наука, 1988. — 278 с. — С. 41.

[66] Андерсон Р. Доказательство правильности программ. — М.: Мир, 1988. — С. 152.

[67] Грис Д. Наука программирования. — М.: Мир, 1984. — С. 303.

[68] Mitkin S. DRAKON language and DRAKON Editor. — http://drakon-editor.sourceforge.net/.

[69] Тышов Г. Программа ИС Дракон. — http://bit.ly/2WZHGZx.

[70] Mitkin S. Drakon.Tech is a visual online development environment for JavaScript. — https://drakon.tech.

[71] Ломов Б. Ф. Эргономические (инженерно-психологические) факторы художественного конструирования // Учебно-методические материалы по художественному конструированию. — М., 1965.

[72] Хьюз Дж., Мичтом Дж. Структурный подход к программированию / пер. с англ., под ред. В. Ш. Кауфмана. – М.: Мир, 1980. – 278 с. – С. 80.

[73] Питерс Л. Дж. Методы отображения и компоновки программных средств // Труды института по электротехнике и радиоэлектронике ТИИЭР. 1980. Т. 68, №9. – С. 60.

[74] Семакин И. Г. Основы алгоритмизации и программирования. — М.: ИЦ Академия, 2008. – С. 32, рис. 113в.

[75] Образцы итоговых заданий по оценке качества подготовки школьников по информатике // Информатика и образование. 1999. №9. — С. 8-21.

[76] Робертсон Л. А. Программирование — это просто. Пошаговый подход / Перевод с 4-го англ. издания. — М.: БИНОМ. Лаборатория знаний, 2008. — 383 с. — С. 265.

[77] Вельбицкий И. В. // Знакомьтесь, Р-технология // НТР: Проблемы и решения. — 1987. № 13. — С. 5.

[78] Толковый словарь по вычислительным системам. / Под ред. В. Иллингуорта, Э.Л. Глейзера, И.К. Пайла / Пер. с англ. / Под ред. Е.К. Масловского. — М.: Машиностроение, 1991. — 560 с. — С. 193.

[79] Очков В. Ф., Пухначев Ю. В. 128 советов начинающему программисту. 2-е изд. — М.: Энергоатомиздат, 1992. — 256 с. — С. 21.

[80] Дробушевич Л. Ф., Конах В. В. Анализ топологий визуальных нотаций для записи алгоритмов и программ // Информационные технологии и системы 2011 (ИТС 2011) : материалы межд. науч. конф., БГУИР, Минск, 26 окт. 2011. — 306 с. — С. 212—213.

[81] Пышкин Е.В. Структуры данных и алгоритмы: реализация на C/C++. — СПб.: ФТК СПб. гос. политехн. ун-т, 2009. — 200 с. — С. 35.

[82] Дейкстра Э. Заметки по структурному программированию // Дал У., Дейкстра Э., Хоор К. Структурное программирование. / Пер с англ. / Под ред. Э.З. Любимского, В.В. Мартынюка. — М.: Мир, 1975. — 247 с. – С. 7–97.

[83] Тюгашев А.А. Графические языки программирования и их применение в системах управления реального времени. – Самара: Изд-во СНЦ РАН, 2009. — 98 с. — С. 50.

[84] Ashcroft E, Manna Z. The translation of “goto” programs into “while” programs // Proceedings of IFIP Congress, Ljubljana, Yugoslavia, 1971. — pp. 250-255.

[85] Пышкин Е. В. Структурное проектирование: основание и развитие методов. С примерами на языке C++: Учеб. пособие. — СПб.: Политехнический ун-т, 2005. — 324 с. — С. 283. — http://kspt.icc.spbstu.ru/media/files/people/pyshkin/books/StructDesign-Excerpt.pdf.

[86] Окулова Л. П. Проектирование образовательного процесса в соответствии с требованиями педагогической эргономики // Вестник Наука и практика (29 мая 2012 — 31 мая 2012) — Мат. конф. «Инновации и научные исследования, а также их примен. на практике». Варшава, — http://xn--e1aajfpcds8ay4h.com.ua/pages/view/730.

[87] Беляков Е. Новый алгоритм: раздевайся и быстро ложись спать. Диалог на языке «Дракона» // Учительская газета. 13 марта 2001. № 10. — С. 16.

[88] Фокин Ю. Г. Теория и технология обучения: деятельностный подход: учебное пособие для студентов высших учебных заведений. — 3-е изд., испр.. — М.: Издательский центр «Академия», 2008. — С. 233. — 240 с.

[89] Лозовская М. Э., Васильева Е. Б., Клочкова Л. В., Яровая Ю. А., Степанов Г. А. Новый вектор в преподавании фтизиатрии студентам-педиатрам // Туберкулёз и болезни лёгких, Том 97, № 5, 2019. — С. 73, 74.

[90] Косова А.Е. Язык ДРАКОН как средство повышения качества профессиональной компетентности // Современное образование: повышение профессиональной компетентности преподавателей вуза —, гарантия обеспечения качества образования: науч.-метод. конф. 1-2 фев 2018. — Томск: изд Томск. ун-та сист. упр. и радиоэлектрон., 2018. — 311 с. — С. 89-91 .

[91] Пичкалев А.В. Разработка программного обеспечения робототехнических комплексов с использованием графического языка ДРАКОН // Робототехника и искусственный интеллект:, мат-лы VII Всеросс. науч.-тех. конф. (г. Железногорск, 11 дек 2015) / под науч. ред. В.А. Углева. – Красноярск: Сибирский федер. ун-т, 2016. – 194 с. — С. 90-94.

[92] Безель Я. Б. Можно ли улучшить работу ума? Новый взгляд на проблему // Вестник РАН. 2003. Т. 73. № 4. — С. 364—365.

[93] Пароджанов В. Д. Графический синтаксис языка ДРАКОН // Программирование. — 1995. Т. 3. — С. 45-62. — http://drakon.su/_media/video_i_prezentacii/graphical_syntax_.pdf.

[94] Дородницын А.А. Информатика: предмет и задачи // Вестник Академии наук СССР. 1985. №2. — С. 85-89.

[95] Непейвода Н.Н. Алгоритм // Новая философская энциклопедия: В 4-х т. / Ин-т философии РАН. / Под. ред Степина В.С., Гусейнова А.А. и др. — М.: Мысль, 2010. — Том 1. — 744с. — С. 76. http://iph.ras.ru/elib/0111.html.

[96] Понимаемость программного средства (Understandability) по ГОСТ 28806-90. Совокупность свойств программного средства, характеризующая затраты усилий пользователя на понимание логической концепции этого программного средства., согласованный и обоснованный характер и позволяющие логически точно определять конкретное назначение и содержание этих правил [из п. 3.1 Прил. 2 ГОСТ 28806-90], Примечание. Под логической концепцией подразумеваются основополагающие понятия, принципы и соглашения, придающие системе правил работы пользователя с программным средством.

[97] Тохметов А.Т., Танченко Л.А. Введение в теорию алгоритмов. Учебное пособие. — Павлодар: Павлодарский гос. пед. ин-т, 2011. — 100 с. — С. 4. — ftp://ppi.kz/Tohmetov_Tanchenko_vvedenie_v_teor_algoritm.pdf.

[98] Кемп П., Арнс К. Введение в биологию. — М.: Мир, 1988. — 672 с. — С. 612, 613.

[99] Сонин Н. И., Сапин М. Р. Биология. Человек. 8 класс. Учебник. — М.: Дрофа, 2004. — С. 130.

[100] World Health Organization. 10 facts on patient safety. Updated March 2018. — http://www.who.int/features/factfiles/patient_safety/en.

[101] Ефанов С.Д. Программирование микроконтроллеров на ДРАКОНе // Сайт Easyelectronics, 10 января 2012. — http://bit.ly/332kL3K.

[102] ГОСТ 19.701-90 пункт 3.3.1.1..

[103] Венда В. Предисловие к русскому изданию. – В кн.: Боумен У. Графическое представление информации. — М.: Мир, 1971. – С. 9.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

А

Б

В

Г

Д

Ж

З

И

К

Конъюнкт
Конъюнктивная форма
Конъюнкция
Контур
Критерий сверхвысокого понимания
Критическая валентная точка

Л

Лиана
Литерал
Логическая последовательность исполнения веток
Логическая операция И
Логическая операция ИЛИ
Логическая операция НЕ
Логическая переменная величина
Логическая связка
Логический фрагмент

М

Макроалгоритм
Макроикона
- – Веточный цикл
- – Ввод по таймеру
- – Вставка по таймеру
- – Вывод по таймеру
- – Действие по таймеру
- – Действие с заданной длительностью
- – Длительность группы действий. Время слева
- – Длительность группы действий. Время справа
- – Заголовок с параметрами
- – Параллельный процесс
- – Параллельный процесс по таймеру
- – Переключатель
- – Переключатель по таймеру
- – Переключающий цикл
- – Развилка
- – Развилка по таймеру
- – Решение с заданной длительностью
- – Цикл ДЛЯ
- – Цикл ДЛЯ по таймеру
- – Цикл ЖДАТЬ
- – Цикл ЖДАТЬ по таймеру
- – Цикл со стрелкой
- – Цикл со стрелкой по таймеру

Маркер
- – арка
- – белый овал
- – вертикаль
- – глаза

Н

О

П

Р

С

Т

У

Ф

Ц

Ч

Ш

Э

A

C

D

E

F

G

H

I

K

L

P

S

T

U

W