Пояснительная записка

к уведомлению о разработке проекта национального стандарта
«Схемы алгоритмов и алгоритмических систем в здравоохранении
и медицине. Обозначения условные и правила выполнения»

Оглавление

Часть 1.
Положения, отличающие предлагаемый стандарт от межгосударственного стандарта ГОСТ 19.701–90 и международного стандарта ISO 5807:85

Данный материал взят из учебного пособия:
Паронджанов В.Д. Алгоритмические языки и программирование: ДРАКОН: учебное пособие для вузов. Москва: Издательство Юрайт, 2021. 437 с. — (Высшее образование).

Тема 35.
СРАВНЕНИЕ ДРАКОН-СХЕМ И БЛОК-СХЕМ

ВВЕДЕНИЕ

В теме 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.

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.

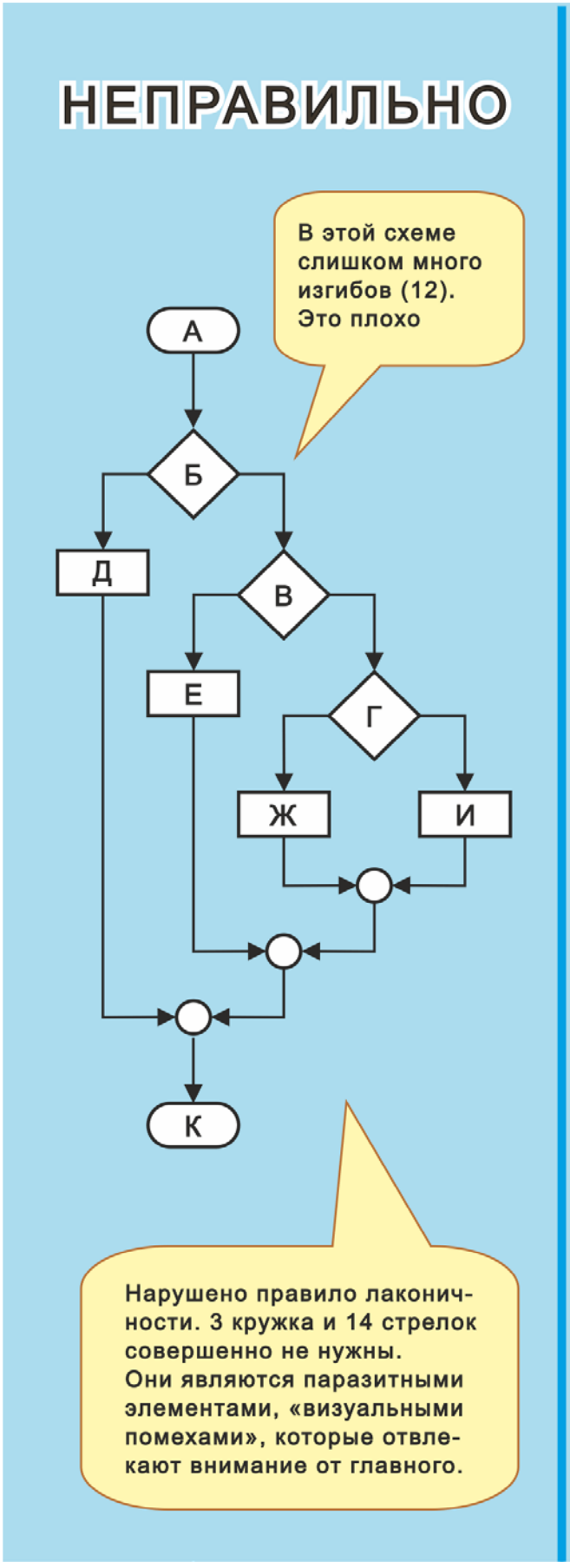


Рис. 178. Плохая схема. Недостатки: слишком много изгибов; имеются паразитные элементы

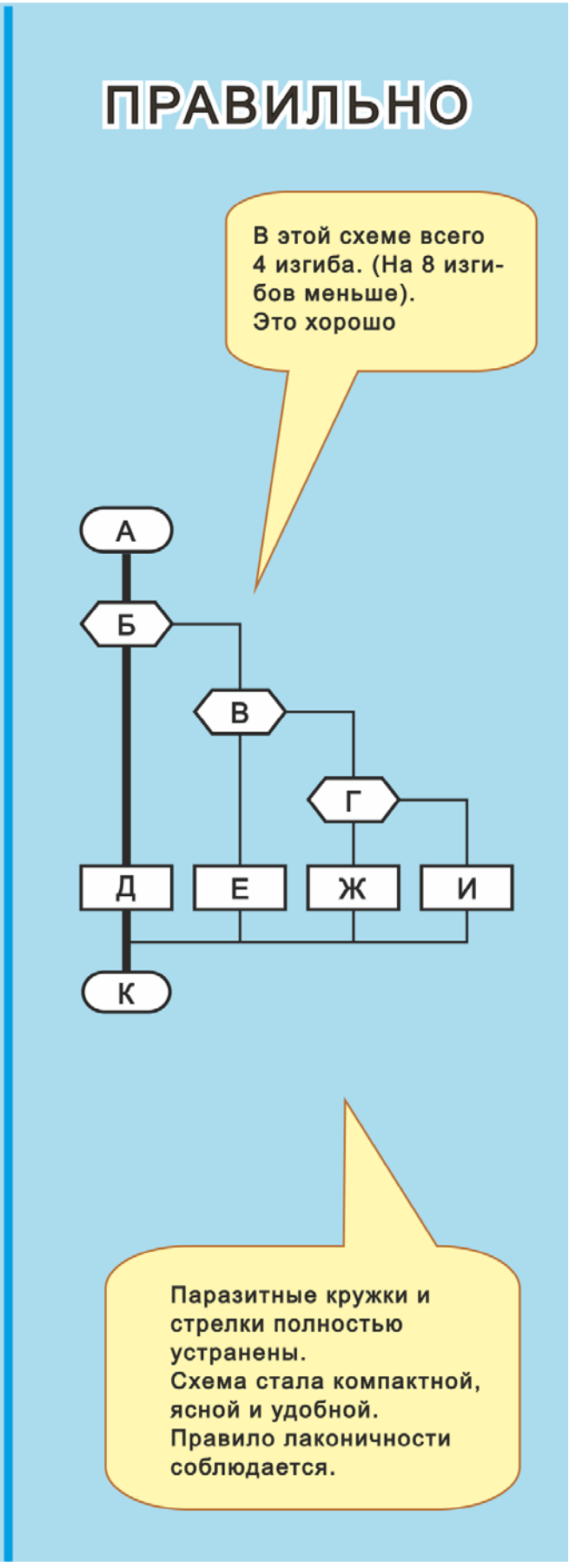


Рис. 179. Хорошая (эргономичная) схема. Она нарисована по правилам языка ДРАКОН

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:

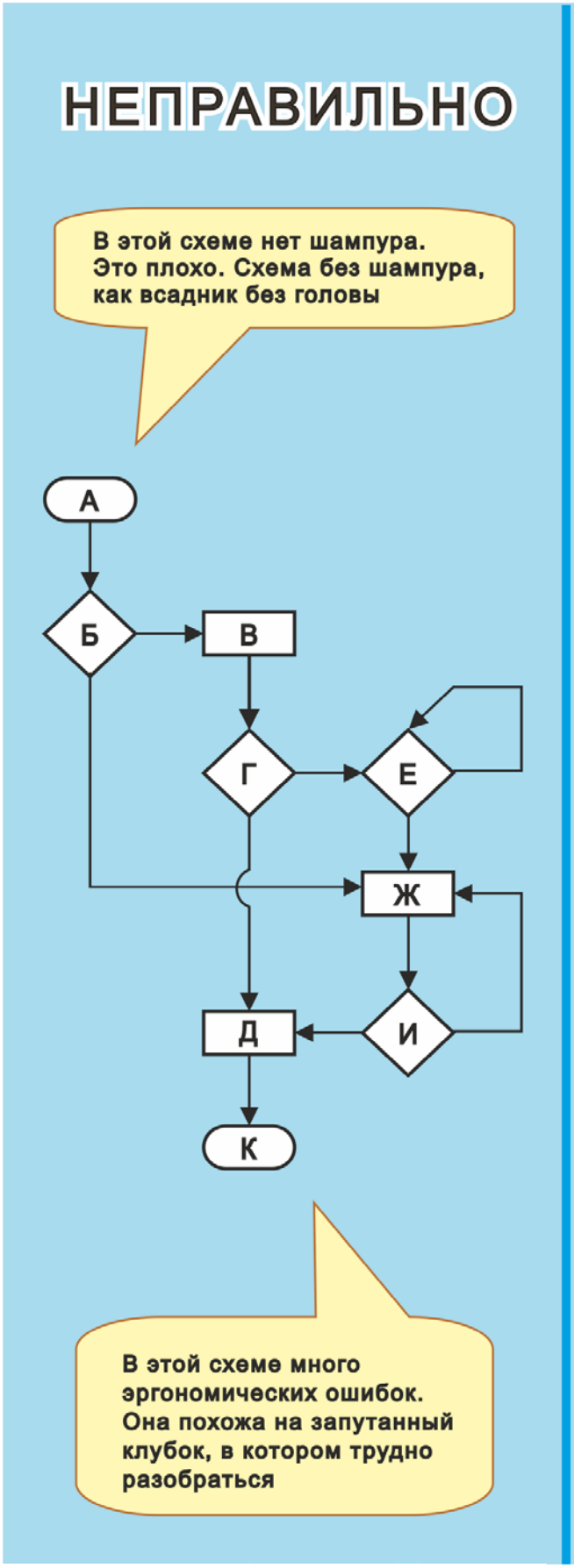


Рис. 180. Плохая схема. Такие схемы часто рисуют многие уважаемые ученые, забывающие об эргономике

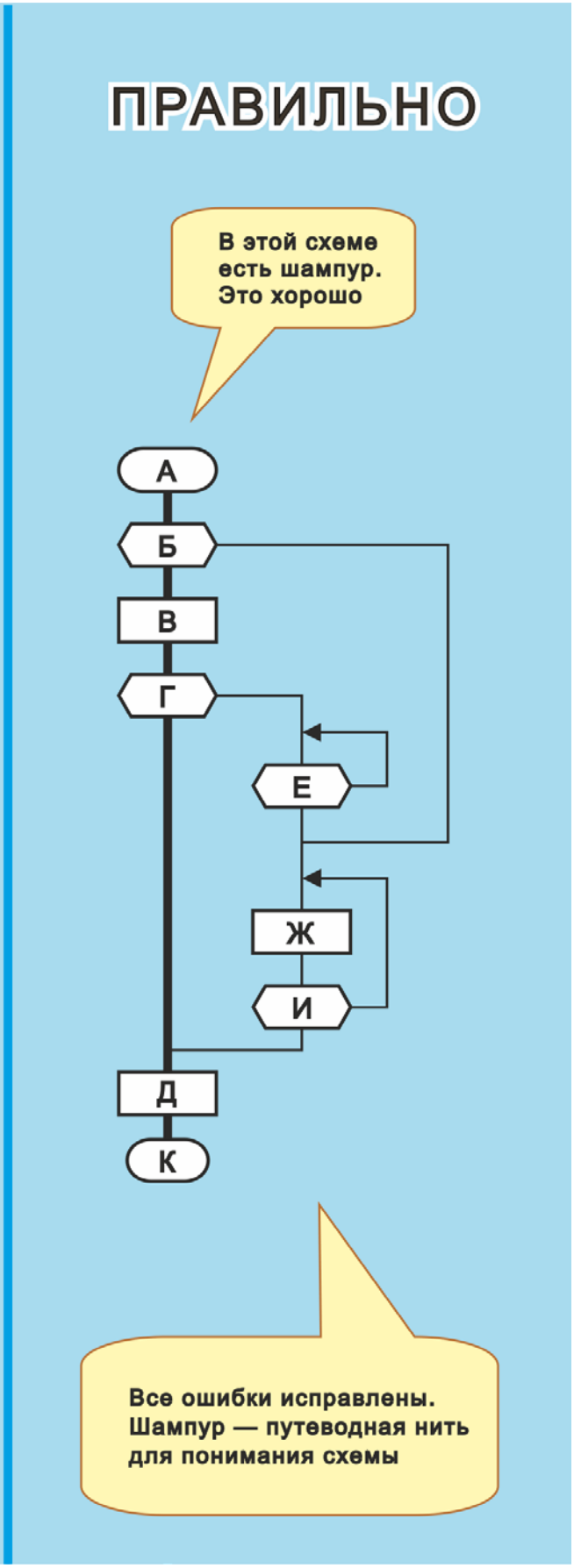


Рис. 181. Хорошая (эргономичная) схема. Она нарисована по правилам языка ДРАКОН

- 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.

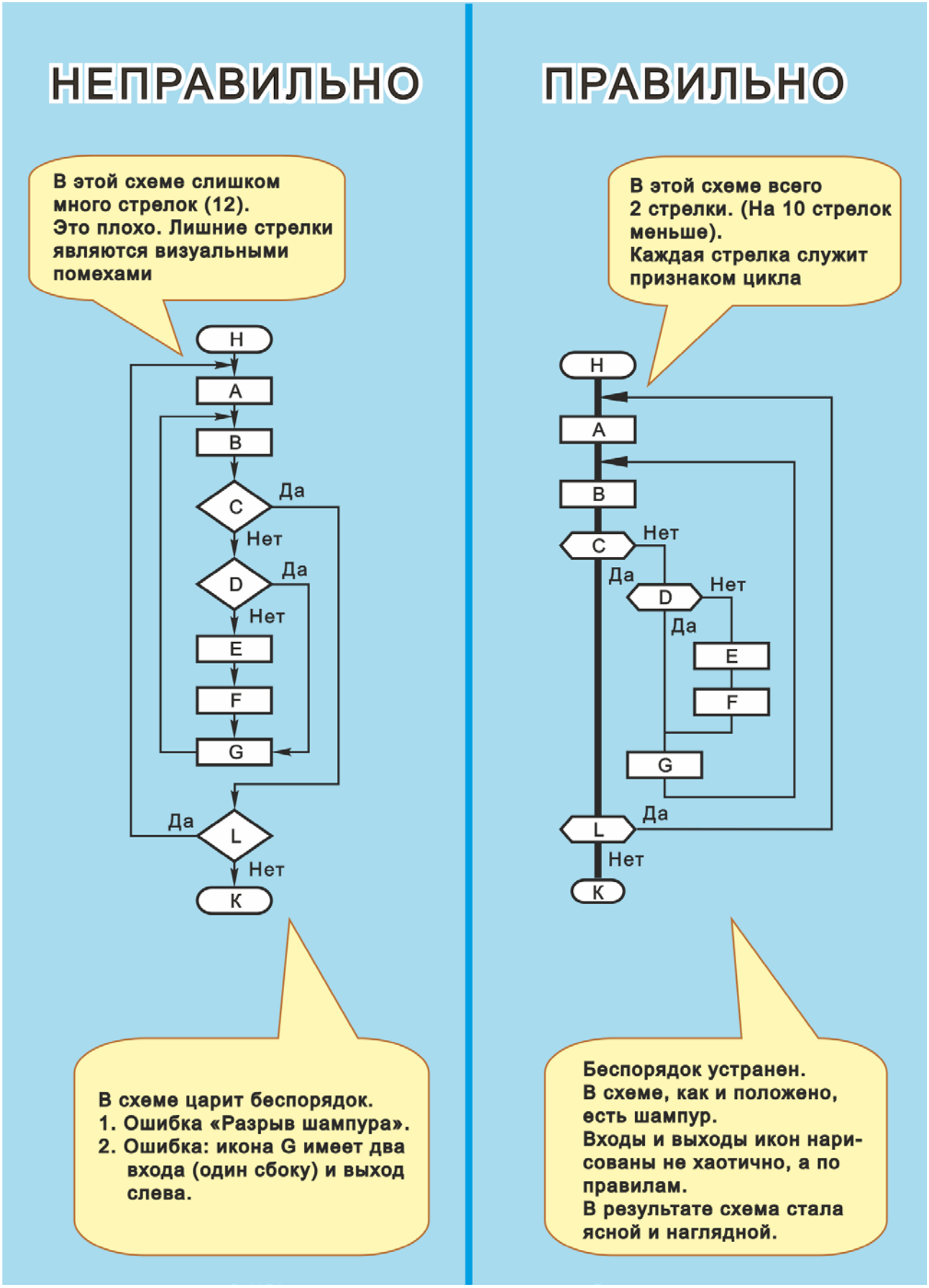


Рис. 182. Плохая схема. В ней много эргономических ошибок, что затрудняет понимание алгоритма

Рис. 183. Хорошая (эргономичная) схема. Она нарисована по правилам языка ДРАКОН

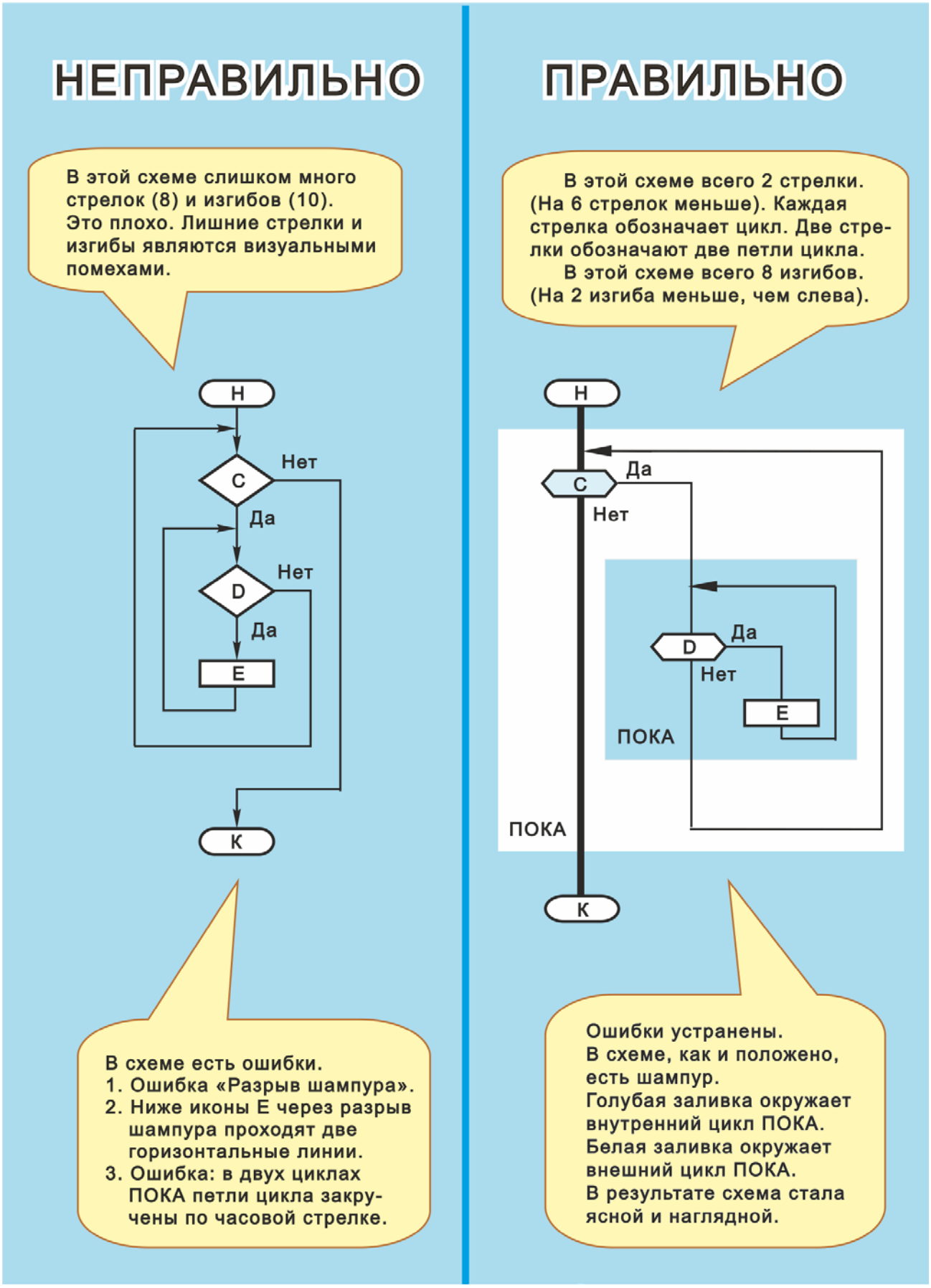


Рис. 184. Плохая схема. Вложенный цикл ПОКА изображен без учета правил эргономики. Шампур разорван.

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.

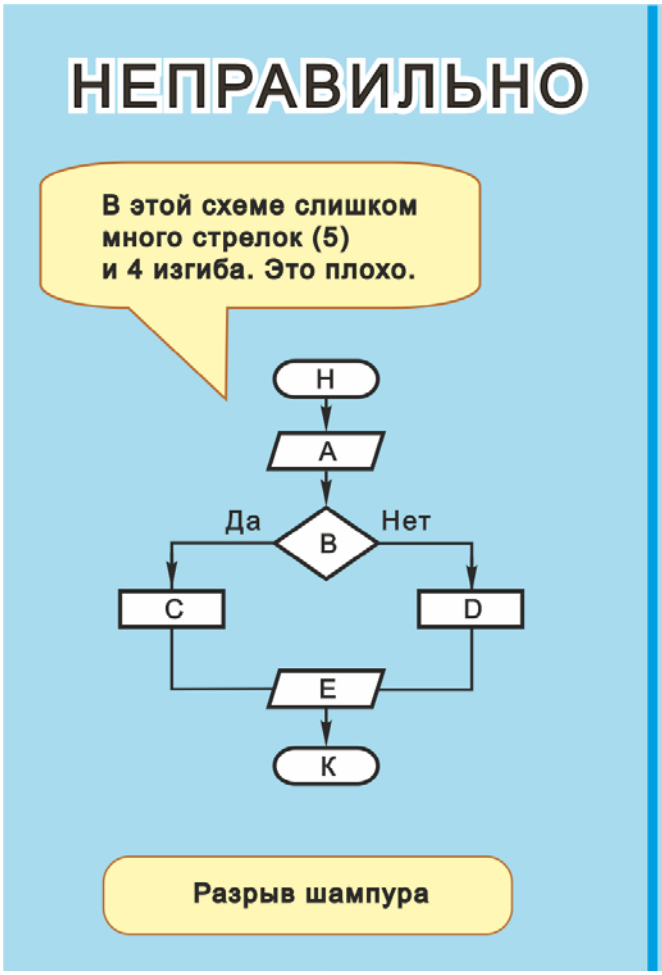


Рис. 186. Плохая схема. Выход из ромба влево (а не вниз) мешает правильному изображению шампура

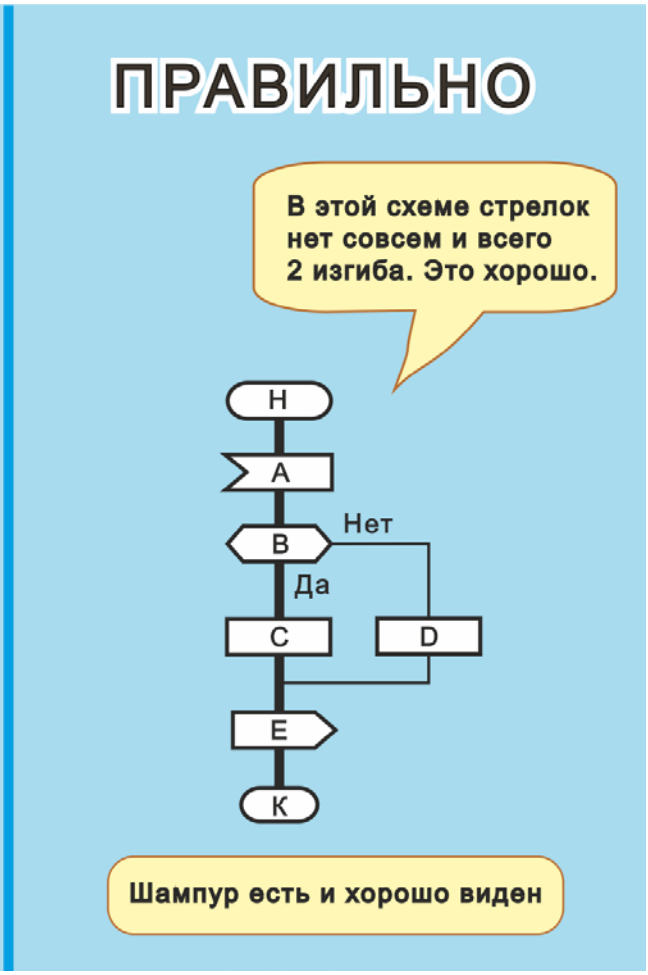


Рис. 187. Хорошая (эргономичная) схема. Шампур нарисован верно, так как икона Вопрос имеет выход вниз.

ТИПИЧНЫЕ ОШИБКИ В БЛОК-СХЕМАХ АЛГОРИТМОВ

На основании анализа алгоритмов на рис. 178-187 можно составить перечень эргономических ошибок, которые мы выявили в этой теме:

- 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 теме 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 теме 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.

Тема 36.

КРИТИЧЕСКИЙ АНАЛИЗ БЛОК-СХЕМ АЛГОРИТМОВ ПО ГОСТ 19.701-90

СРЕДСТВА ПРЕДОТВРАЩЕНИЯ ОШИБОК В ЯЗЫКЕ ДРАКОН

Язык ДРАКОН имеет средства, специально предназначенные для обеспечения безошибочности:

- исчисление икон (см. тему 32);
- алгоритмическая логика (см. часть 3);
- теория отростков;
- теория валентных точек;
- теория макроикон;
- теория стрелок;
- ДРАКОН-конструктор (см. темы 33, 34).

СРАВНЕНИЕ СО СТАНДАРТОМ ГОСТ 19.701—90

В блок-схемах по ГОСТ 19.701—90 формализованы только фигуры. За пределами формализации остаются:

- правила присоединения отростков линий к фигурам;
- точки размещения фигур, или точки ввода фигур;
- связи между фигурами.

В отличие от блок-схем, в дракон-схемах проведена полная формализация. Строго определены не только иконы, но их отростки, а также соединительные линии и точки ввода фигур. Формализацию соединительных линий обеспечивают формальные отростки икон, формальные валентные точки и макроиконы.

Правила работы с отростками, валентными точками и макроиконами описаны в работе [39].

ТЕОРИЯ ОТРОСТКОВ

Назначение теории отростков — устранить неоднозначность присоединения отростков к графическим фигурам.

В языке ДРАКОН формализация отростков выполнена так:

- иконы заданы не отдельно, а вместе с отростками;
- число отростков, тип и направление каждого отростка строго определены.

Существуют три типа отростков:

- входной;
- выходной;
- нейтральный.

Входной и выходной отростки 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. Все отростки ориентированы к центру иконы.

На рис. 188 показаны примеры входных и выходных отростков.

Нейтральный отросток — это горизонтальный отросток, соединяющий иконы Синхронизатор с иконой-хозяином.

Рис. 188. Язык ДРАКОН предлагает однозначный (единственный) вариант присоединения отростков к иконам.

Икона Действие имеет два отростка: входной (сверху) и выходной (снизу). Икона Вопрос имеет три отростка: входной (сверху) и два выходных (внизу и справа)

Перечисленные правила задают однозначную привязку отростков к иконам, являясь средством формализации.

СРАВНИВАЕМ СО СТАНДАРТОМ ГОСТ 19.701—90

Иконам Действие и Вопрос в стандарте ГОСТ 19.701—90 соответствуют блоки Процесс и Решение (рис. 189).

Легко видеть, что формализация отростков в ГОСТе отсутствует. Действительно, в стандарте для блока Процесс (рис. 189) предусмотрены четыре варианта. Больше того, число вариантов может возрасти вдвое, если учесть, что стандарт разрешает выполнять чертежи как со стрелками, так и без них:

> ≪При необходимости или для повышения удобочитаемости могут быть добавлены стрелки-указатели≫ [102].

Для сравнения: в языке ДРАКОН для иконы Действие (рис. 188) предусмотрен всего один чертеж, что исключает путаницу и предотвращает ошибки.

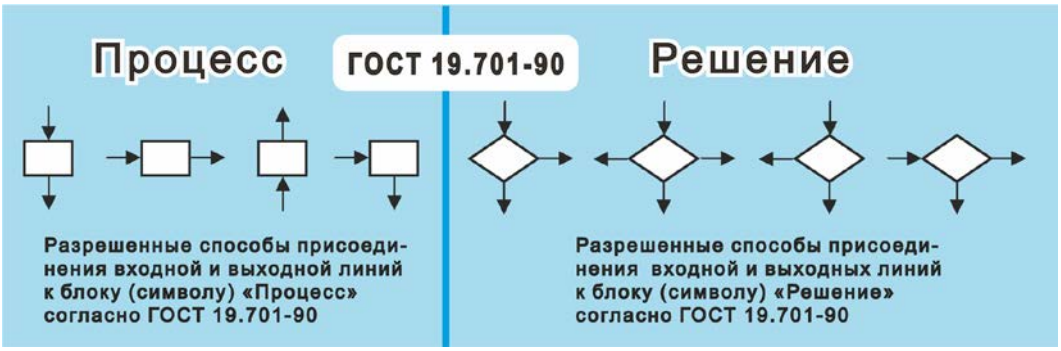


Рис. 189. ГОСТ 19.701—90 разрешает присоединять линии к блокам Процесс и Решение четырьмя разными способами, т. е. неоднозначно.

Язык ДРАКОН устраняет недостаток и предлагает однозначный (единственный) вариант.

Далее. Для блока Решение (рис. 189) ГОСТ разрешает четыре варианта (или даже восемь — с использованием стрелок и без них).

Сравним с языком ДРАКОН и иконой Вопрос (рис. 188). Для нее задан один-единственный чертеж, что обеспечивает строгую формализацию.

Неоднозначность графики стандарта является прямым следствием указания ГОСТ: ≪Символы могут быть вычерчены в любой ориентации≫ [102, с. 12].

Наличие в стандарте разных способов подключения отростков к блокам говорит об отсутствии формализации соединительных линий в блок-схемах, что является недостатком стандарта ГОСТ 19.701—90.

ТЕОРИЯ ВАЛЕНТНЫХ ТОЧЕК

Валентные точки можно рассматривать как точки размещения икон, или точки ввода икон. Расположение валентных точек на чертеже дракон-алгоритма строго определено. Иконы можно вставлять только в валентных точках, и больше нигде. На рис. 190 показаны валентные точки, находящиеся на отростках икон Действие и Вопрос. Для каждой точки определено, какие иконы и макроиконы можно вводить в данную точку.

Рис. 190. Икона Действие имеет две валентные точки, а икона Вопрос — три

На чертеже дракон-алгоритма валентные точки не изображаются, но подразумеваются. Они визуализируются (на короткое время) только в процессе построения дракон-схемы — при работе инструментальной программы ДРАКОН-конструктор.

ДИНАМИКА ВАЛЕНТНЫХ ТОЧЕК

Вспомним, что чертеж дракон-алгоритма формируется методом логического вывода из аксиом. На каждом шаге построения происходит размножение валентных точек. Процесс размножения можно рассматривать как динамический процесс преобразования валентных точек.

Рассмотрим построение дракон-схемы силуэт подробнее (рис. 191).

В аксиоме-силуэт (рис. 191, слева) всего 3 валентных точки. После добавления к аксиоме ветки силуэта (рис. 191, в центре) получается уже 5 точек. А после вставки иконы Действие (рис. 191, справа) число точек увеличивается до 6. Дальнейший процесс построения силуэта приводит к монотонному росту числа валентных точек.

Это значит, что иконы (или атомы) можно вставлять не куда угодно, а только в строго определенные места. Ввод 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 программы ДРАКОН-конструктор, который осуществляет визуальный логический вывод. Таким образом, описанная операция строго формализована и защищена от многих ошибок.

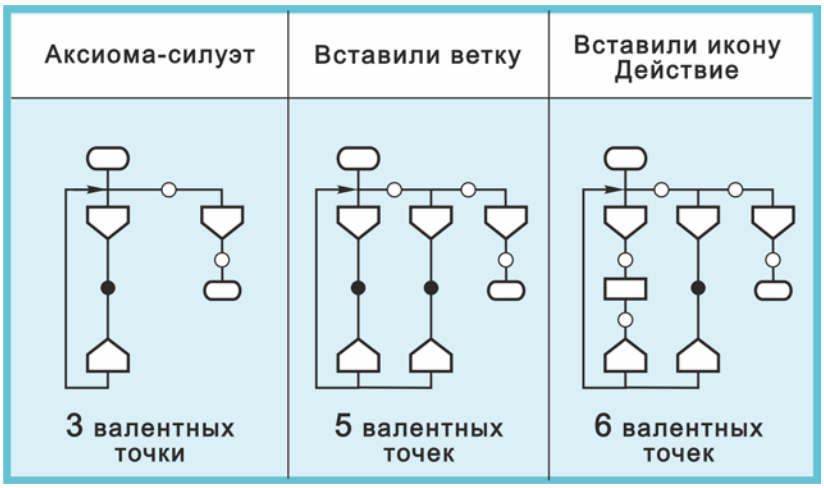


Рис. 191. Размножение валентных точек при проектировании дракон-схемы силуэт

ТЕОРИЯ МАКРОИКОН

Некоторые сочетания икон и соединительных линий повторяются чаще других; их целесообразно выделить и стандартизовать. Такие сочетания называются макроиконами. Макроиконы служат для предотвращения ошибок при проведении соединительных линий.

Проведем анализ двух макроикон: Развилка и цикл Стрелка. Обе они содержат только одну икону — икону Вопрос, а также соединительные линии и валентные точки (рис. 192).

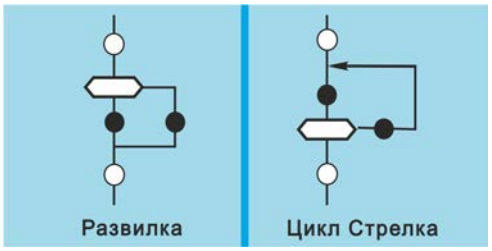


Рис. 192. Две макроиконы: Развилка и цикл Стрелка

Цикл Стрелка — пустой оператор, он служит заготовкой для построения реальных циклов с условием. Если заполнить верхнюю черную валентную точку, получим цикл ДО (do while). Если нижнюю — цикл ПОКА (while). Если обе — цикл do while do.

Макроикона Развилка — тоже пустой оператор. Если заполнить левую черную валентную точку, то получим оператор ЕСЛИ (if). Если обе — оператор ЕСЛИ ТО (if else).

Теория макроикон позволяет различать опасное и безопасное использование макроикон. Чтобы безопасно построить цикл с условием, следует использовать макроикону цикл Стрелка, но не Развилку.

Недопустимые действия. Опасность появляется тогда, когда пользователь пытается нарушить правила и переделать макроикону Развилка, превратив ее в цикл.

Это можно сделать с помощью операции ≪Пересадка лианы≫: нужно лишь оторвать правый выход иконы Вопрос в валентной точке и пересадить его наверх в нужную точку, превратив в стрелку цикла. Однако в языке ДРАКОН так поступать запрещено, так как возникает опасность допустить ошибку.

Почему? Потому что речь идет об образовании нового цикла, аналогичного оператору *goto*, который реализует переход на оператор, расположенный выше (раньше) на чертеже дракон-алгоритма.

Подобная операция в языке ДРАКОН считается небезопасной и недопустимой. Специалист по надежности программного обеспечения Гленфорд Майерс предостерегает:

> ≪Наихудшим применением оператора *goto* считается переход на оператор, расположенный выше (раньше) в тексте программы≫ [11, с. 135].

Запрет на образование нового цикла вызван тем, что переход на оператор, расположенный выше, является самым плохим (и недопустимым) использованием оператора перехода.

Как избежать ошибки. В меню программы ≪ДРАКОН-конструктор≫ предусмотрена макроикона цикл Стрелка, предназначенная для безопасного создания циклов с условием. При этом угроза ошибок снимается.

СРАВНИВАЕМ СО СТАНДАРТОМ ГОСТ 19.701—90

Формализация макроикон и валентных точек в ГОСТе не предусмотрена. Трудности понимания сложных блок-схем алгоритмов, выполненных по ГОСТ 19.701—90, связаны с тем, что в них отсутствует должный порядок — правила разработки схем не формализованы, не эргономичны и разрешают совершать недопустимые действия. Эксперты отмечают:

> ≪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].

В отличие от ГОСТ, в языке ДРАКОН икону Вопрос можно вставить не ≪в любом месте≫ чертежа, а только в валентных точках, и повести выходы не на ≪какие угодно участки≫, а только и исключительно в те места, которые разрешены согласно формальным правилам языка ДРАКОН.

МИНИМИЗАЦИЯ ЧИСЛА СТРЕЛОК

Вспомним еще раз правило Эшфорда:

> ≪Бесполезно стремиться направить внимание на важнейшие характеристики, если они окружены лишними, не относящимися к ним визуальными раздражителями, мешающими восприятию главного≫ [103].

Чтобы изобразить направление потока управления, в блок-схемах используют большое количество стрелок-указателей. Согласно Эшфорду, стрелки загромождают чертеж и отвлекают внимание, являясь лишними визуальными раздражителями.

В языке ДРАКОН направление потока управления показано другими средствами, более наглядными и эффективными — без использования стрелок. По этой причине в ДРАКОНе теория стрелок тривиальна и сводится к одной фразе: *стрелки*

запрещены всюду, *кроме цикла Стрелка и стрелки силуэта.* В результате дракон-схемы становятся более удобными и прозрачными.

ФОРМАЛИЗОВАННЫЙ ЧЕРТЕЖ АЛГОРИТМА

Формализованный чертеж алгоритма — это чертеж, для которого определены:

- формальное описание алфавита графических фигур (икон и макроикон);
- формальное описание синтаксиса, т. е. 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 с учетом элементарных теорий, формализующих отростки, валентные точки, макроиконы и стрелки.

КРИТИЧЕСКИЙ АНАЛИЗ БЛОК-СХЕМ АЛГОРИТМОВ

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, с. 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 свое 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, с. 190].

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 “flawchart” — порочными схемами)… 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, с. 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 по ГОСТ 19.701—90 не имеют выразительных средств и обозначений для описания алгоритмов реального времени.

Чтобы устранить недостатки стандарта 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[1].

Последний, в частности, используется в системе образования для обучения основам алгоритмизации.

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 Э. Дейкстра в классической работе ≪Заметки по структурному программированию≫ [82, с. 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.

---

[1] В 1992 г. государства — члены СНГ заключили соглашение, которым признали действующие стандарты ГОСТ СССР в качестве межгосударственных с сохранением аббревиатуры ≪ГОСТ≫ за вновь вводимыми межгосударственными стандартами.

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, с. 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, с. 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, с. 25—28]. Чтобы у читателя не осталось сомнений, мы приводим точные копии подлинных рисунков Дейкстры (рис. 193, 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, с. 190];
- 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, с. 393—448].

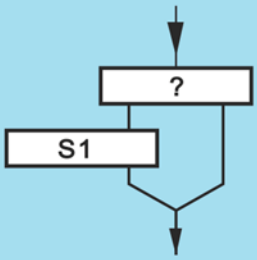
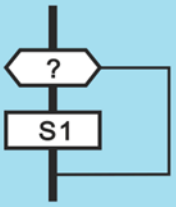
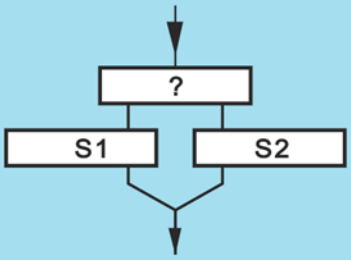
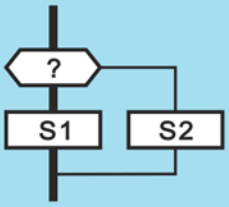
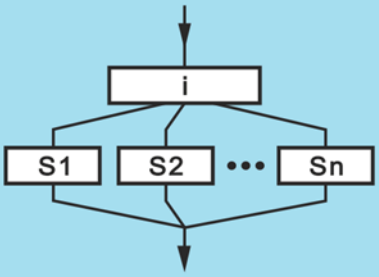
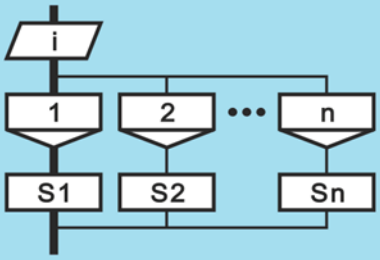
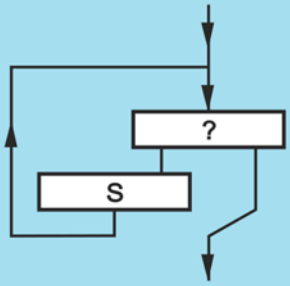
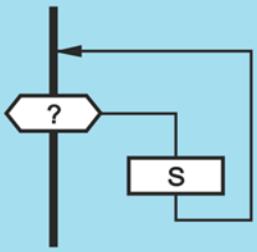
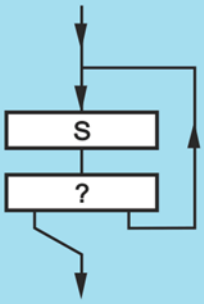
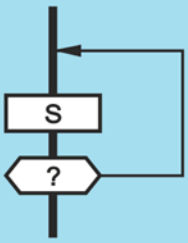
Структурные операторы Дейкстры	Структурные блок-схемы Дейкстры	Дракон-схемы
if ? do S1		
if ? then S1 else S2		
case i of (S1;S2;...Sn)		
while ? do S		
repeat S until ?		

Рис. 193. Структурные блок-схемы Дейкстры (в центре) и эквивалентные им дракон-схемы (справа)

Кроме того, в языке ДРАКОН — на основе четырех принципов структуризации блок-схем Дейкстры — разработан двумерный структурный подход к алгоритмам и программам (шампур-метод) [39, с. 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 АШКРОФТА И Зохара МАННЫ

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 о 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 — Манне [84] и Йодану [10, с. 192—196]. Затем переменная состояния удаляется, превращаясь в идентификаторы икон Имя ветки и Адрес [39, с. 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 Э. 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-Манне 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 предусмотрены специальные средства предотвращения ошибок:

- формализация построения графики с помощью исчисления икон;
- формальные средства алгоритмической логики;
- формализация отростков;
- формализация валентных точек;
- формализация макроикон;
- формализация и минимизация стрелок;
- использование ДРАКОН-конструктора.

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.

СПИСОК ЛИТЕРАТУРЫ

[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.

[15] Степанов А.Н. Курс информатики для студентов информационно-математических специальностей. (Серия «Учебник для вузов»). — СПб.: Питер, 2018. — 1088 с.

[18] Криницкий Н.А. Алгоритмы вокруг нас. — М.: Наука, 1984. — 224 с. — С. 6.

[33] Семёнов Н. М. Программирование и основы алгоритмизации. Учебное пособие. — Томск: Томский политехнический университет, 2009. — С. 71. — 90 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.

[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.

[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.

[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.

Часть 2.
Клиническая алгоритмическая медицина. Алгоритмы диагностики и лечения на медицинском языке ДРАКОН

Справка о медицинском языке ДРАКОН

На основе космической технологии (языка программирования ДРАКОН), в России создан самостоятельный графический язык ДРАКОН для медицинских целей. Он предназначен для разработки алгоритмов действий врача в виде **эргономичных** (легких для восприятия, изучения и анализа) **чертежей** — блок-схем специального вида (дракон-схем).

Создано автоматизированное рабочее место врача-разработчика алгоритмов действий. Рабочее место содержит компьютерную программу ДРАКОН-конструктор «ДРАКОНПРО» https://drakonpro.ru/ и базу данных для хранения создаваемых и уже созданных алгоритмов.

Детальная информация о языке ДРАКОН приведена в учебном пособии **«Клиническая алгоритмическая медицина. Алгоритмы диагностики и лечения на медицинском языке ДРАКОН»** [1].

На книгу получены 7 внешних рецензий от докторов наук [2] и рецензия Экспертной комиссии по работе с учебными изданиями, утвержденная Председателем Координационного совета ректором П.В. Глыбочко (протокол №074 от 16 ноября 2023 г.) [3].

Экспертный совет по науке Департамента здравоохранения Москвы рекомендовал учебное пособие (методические рекомендации № 36) «к печати и последующему внедрению в практику московского здравоохранения» (протокол №10 от 15 сентября 2023 г.) [4].

Препринт (в электронном виде) учебного пособия доступен по ссылке https://drakon.su/_media/klinicheskaja_alg_med10_.pdf Инструкция по построению алгоритмов действий врача с помощью программы ДРАКОН-конструктор приведена в главе 4 учебного пособия, стр. 74-87. В печатной книге [1, pp. 79-94].

Медицинский язык ДРАКОН и ДРАКОН-технология позволяют создавать **эргономичные** (doctor-friendly, user-friendly) **алгоритмы высокой точности**. В качестве примера использования языка ДРАКОН для создания алгоритмов высокой точности разработаны и продемонстрированы профессиональные клинические алгоритмы [1, pp. 217-334]:

- Респираторная терапия дыхательной недостаточности, ассоциированной с COVID-19 (9 алгоритмов).
- Неврология. Персонализированная терапия эпилепсии препаратами вальпроевой кислоты (1 алгоритм).
- Фтизиатрия. Алгоритмы фтизиатрической службы (10 алгоритмов).

Теоретические основы медицинской алгоритмизации. 1. Графическая логика высказываний

Чтобы создавать клинические алгоритмы высокой точности, нужно уметь вычислять сложные логические выражения. Рассмотрим задачу: требуется соединить 3 логические переменные логическими связками конъюнкции («И»). Решим задачу двумя способами:

- с помощью текста в классической логике высказываний;
- с помощью графики в графической логике высказываний.

При использовании текста решение выглядит, например, так:

A&B&C,

где A, B, C — абстрактные логические переменные,

& — логическая связка, обозначающая операцию «И».

Графическое решение дано на рисунке 1, где показана графическая логическая схема, выполняющая операцию «И» между тремя логическими переменными, записанными в трех иконах Вопрос. Каждая логическая переменная имеет форму вопроса и принимает два значения: «Да» и «Нет». Вопросы на рисунке 1 записаны в виде профессионального медицинского текста.

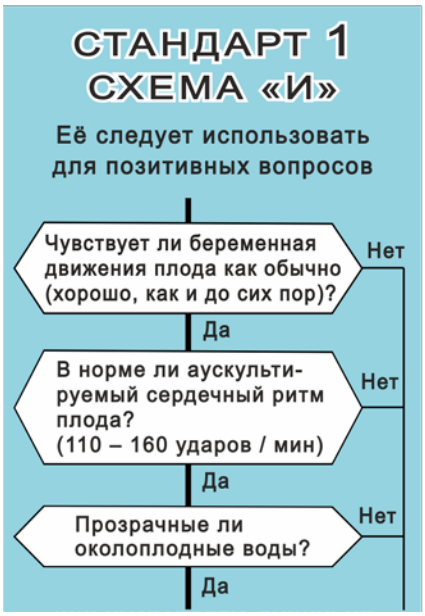


Рис. 1. Стандартная схема **1** «И» с тремя позитивными вопросами

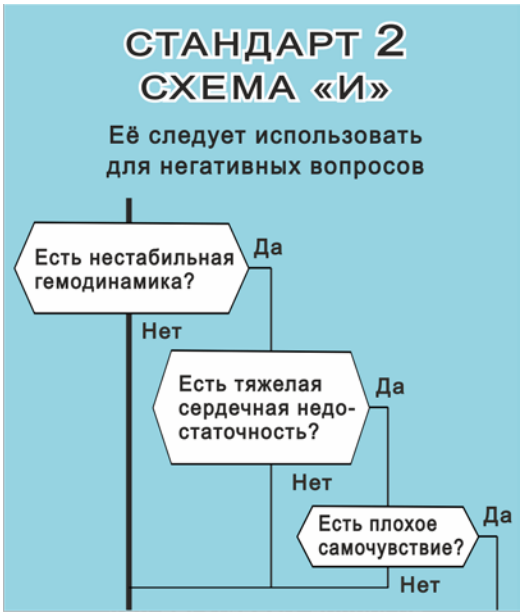


Рис. 2. Стандартная схема **2** «И» с тремя негативными вопросами

Сравним два полученных решения. Буква A кодирует медицинский текст в верхней иконе Вопрос, буква B — текст в средней иконе, буква C — в нижней иконе Вопрос.

На рисунке 1 логическая связка «И» отсутствует, потому что операция «И» выполняется графически. Когда маршрут алгоритма проходит сверху вниз через 3 иконы Вопрос и 3 выхода «Да», это значит, что все три медицинские условия выполнены, роженица и плод находятся в благоприятном состоянии. Если же хотя бы одно условие нарушено, маршрут сворачивает вправо через «Нет» и ситуация становится неблагоприятной.

Невидимая математика. Мы убедились, что в случае клинических алгоритмов текстовая логическая запись A&B&C становится бесполезной и ненужной. Графический метод работает непосредственно с медицинскими текстами и успешно решает задачу, не требуя преобразования их в абстрактную буквенную форму.

При этом логические операции «И», «ИЛИ», «НЕ» выполняются в полном объеме, но логические связки «И», «ИЛИ», «НЕ» бесследно исчезают. Они реализуются с помощью графики и становятся невидимыми. Отсюда название «невидимая математика».

Схема на рисунке 1 имеет два выхода: левый и правый. Будем считать, что буквы A, B, C обозначают тексты в 3-х иконах Вопрос. Тогда левый выход строго математически вычисляет функцию A&B&C методом невидимой математики, а правый выход вычисляет инверсию (логическое отрицание) этой функции.

Язык ДРАКОН позволяет существенно упростить сложную логику медицинских алгоритмов и представить ее в виде наглядных зрительных образов.

Таким образом, графика языка ДРАКОН полностью заменяет текстовые логико-математические выкладки, отображая логическую сущность проблемы графическими средствами.

Позитивные и негативные вопросы. *Позитивный вопрос* при ответе «Да» указывает на благоприятное для больного положение дел. *Негативный вопрос,* наоборот, при ответе «Да» говорит о неблагоприятном для пациента состоянии.

Вопросы необходимо различать, для них предназначены разные схемы:

- *для позитивных* вопросов рекомендуется стандартная схема 1 «И» (рис. 1);
- *для негативных* — стандартная схема 2 «И» (рис. 2).

Картографический критерий. Чтобы облегчить понимание, дракон-алгоритм следует упорядочить и придать смысл движению по вертикали и горизонтали. В нашем примере время движется вертикально вниз, а движение слева направо подчиняется принципу «чем правее, тем хуже». На рис. 1 левая вертикаль описывает более благоприятное для пациента положение дел, а правая — менее благоприятное.

Графическая логика высказываний языка ДРАКОН подробно описана в книге [1, pp. 348-397] см. «Часть VI. Алгоритмическая логика в клинических алгоритмах».

Теоретические основы медицинской алгоритмизации. 2. Аксиоматический метод проектирования графики и визуальное логическое исчисление икон

Клинические алгоритмы имеют качественные отличия от математических алгоритмов. В отличие от последних клинические алгоритмы не имеют теоретического обоснования, теоретические основы медицинской алгоритмизации не разработаны. Это является одной из причин системных дефектов алгоритмов действий врача.

Автором сделана попытка разработать теоретические основы клинических алгоритмов в работе «Алгоритмические языки и программирование: ДРАКОН» [5], см. «Часть VI. Алгоритмы и жизнеритмы».

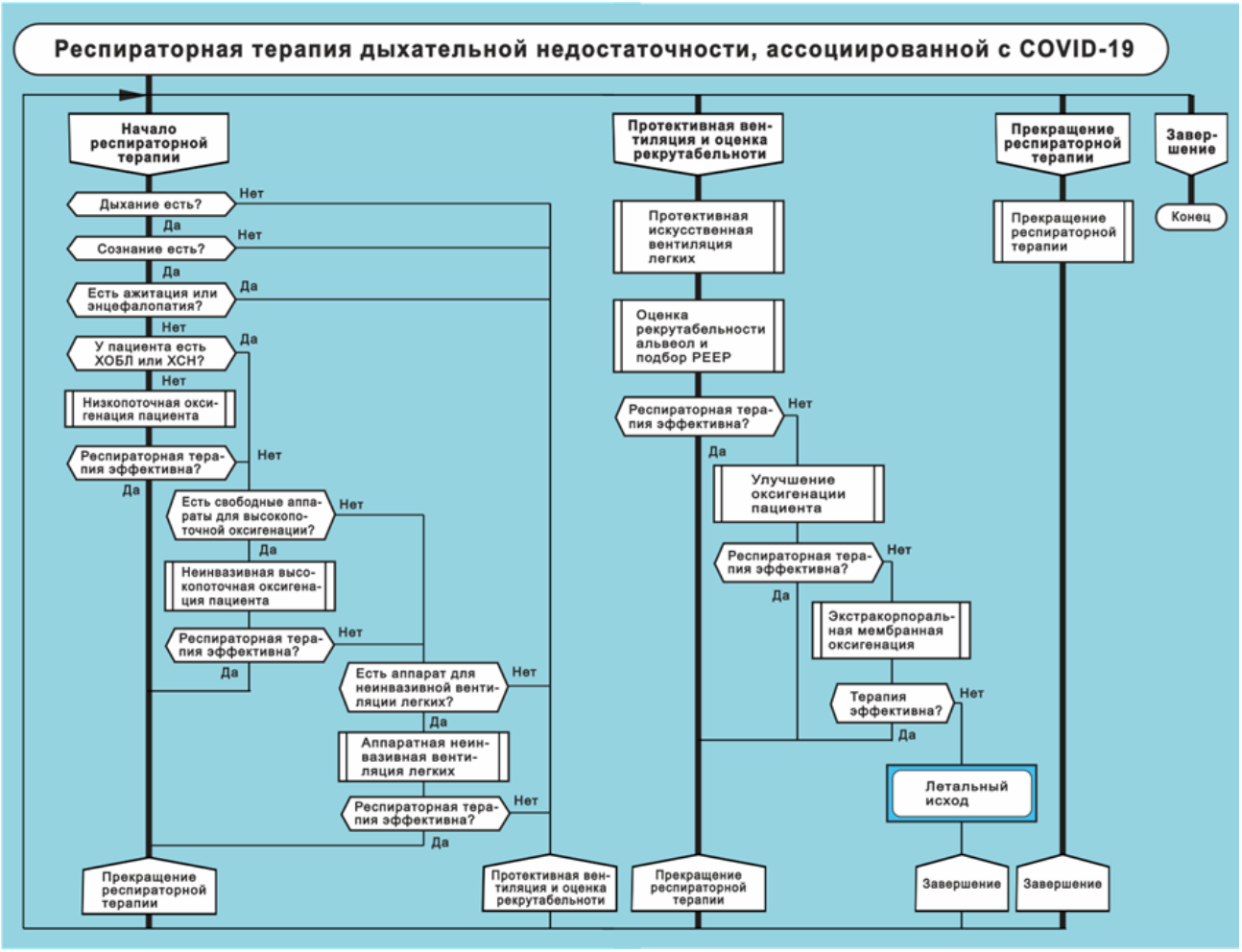


Рис. 3. Комплексный алгоритм «Респираторная терапия дыхательной недостаточности, ассоциированной с COVID-19» (см. [1], рис. 104)

На рис. 3 видно, что в графическом алгоритме важную роль играет *структура алгоритма*, которая реализуется через соединительные линии. Во избежание ошибок разработчику клинического алгоритма запрещено проводить соединительные линии. Все линии автоматически проводит программа ДРАКОН-конструктор.

А.Н. Степанов в «Курсе информатики» отмечает: принцип визуализации «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» [6, pp. 1017-1019].

Указанная теоретическая основа является общей для семейства ДРАКОН-языков, в которое входят как языки программирования, так и медицинский язык ДРАКОН. Программа ДРАКОН-конструктор имеет в своем составе специальный математический аппарат, благодаря чему разработка структуры (проведение соединительных линий на дракон-схеме) выполняется автоматически. Подобная автоматизация имеет важный эффект: скорость построения графических алгоритмов действий врача резко возрастает, а число ошибок сокращается. Специальный математический аппарат (аксиоматический метод, визуальный логический вывод и визуальное логическое исчисление икон) изложен в работе [5] см. темы 32, 34, 36.

Примечание. Медики-разработчики алгоритмов действий врача могут ничего не знать о специальном математическом аппарате, им это совсем не нужно. Указанный аппарат необходим только программистам, создающим программу ДРАКОН-конструктор.

СПИСОК ЛИТЕРАТУРЫ

[1] Клиническая алгоритмическая медицина. Алгоритмы диагностики и лечения на медицинском языке ДРАКОН: учебное пособие / Сморкалов А.Ю., Бочанова Е.Н., Гусев С.Д., Дмитренко Д.В., Шнайдер Н.А., Насырова Р.Ф., Лозовская М.Э., Малышева А.Ю., Бармин Д.Б., Васильева Е.Б., Пароджанов В.Д., Митькин С.Б. / учебное пособие. – Москва: Издательство «РУСАЙНС», 2024. – 536 с. – Иллюстраций: 200.

[2] Семь рецензий от докторов наук на учебное пособие. Ссылка активна на 18.09.24. https://drakon.su/_media/0_shest_recenzij_dlja_sajta_.pdf

[3] Рецензия Экспертной комиссии по работе с учебными изданиями. Протокол №074 от 16 ноября 2023. Ссылка активна на 18.09.24. https://drakon.su/_media/grif_filippova_2420.pdf

[4] Экспертный совет по науке ДЗМ. Протокол №10 от 15 сентября 2023. Ссылка активна на 18.09.24. https://bit.ly/3ZKk5Ol

[5] Пароджанов В.Д. Алгоритмические языки и программирование: ДРАКОН: учебное пособие для вузов. Москва: Издательство Юрайт, 2021. 437 с.

[6] Степанов А.Н. Курс информатики для студентов информационно математических специальностей. (Серия «Учебник для вузов»). СПб.: Питер, 2018. 1088 с.

ЛИТЕРАТУРА ПО ЯЗЫКУ ДРАКОН

1. Клиническая алгоритмическая медицина. Алгоритмы диагностики и лечения на медицинском языке ДРАКОН. / Сморкалов А.Ю., Бочанова Е.Н., Гусев С.Д., Дмитренко Д.В., Шнайдер Н.А., Насырова Р.Ф., Лозовская М.Э., Малышева А.Ю., Бармин Д.Б., Васильева Е.Б., Пароджанов В.Д., Митькин С.Б. / учебное пособие. – Москва: Издательство «РУСАЙНС», 2024. – 536 с. – Иллюстраций: 200.

2. Клиническая алгоритмическая медицина. Алгоритмы диагностики и лечения на медицинском языке ДРАКОН. / Сморкалов А.Ю., Бочанова Е.Н., Гусев С.Д., Дмитренко Д.В., Шнайдер Н.А., Насырова Р.Ф., Лозовская М.Э., Малышева А.Ю., Бармин Д.Б., Васильева Е.Б., Пароджанов В.Д., Митькин С.Б. / учебное пособие. – Москва: Препринт, 2022. – 477 с. – Иллюстраций: 200. *(Электронная книга).* https://drakon.su/_media/klinicheskaja_algoritmicheskaja_medicina_.pdf

3. Пароджанов В.Д. Алгоритмические языки и программирование: ДРАКОН : учебное пособие для вузов. — Москва: Издательство Юрайт, 2021. — 436 с. — Иллюстраций 193. — (Высшее образование).

4. Паронджанов В.Д. Почему врачи убивают и калечат пациентов, или Зачем врачу блок-схемы алгоритмов? Иллюстрированные алгоритмы диагностики и лечения – перспективный путь развития медицины. Клиническое мышление высокой точности и безопасность пациентов. / Предисловие члена-корр. РАН Г.В. Порядина – Москва: ДМК Пресс, 2017. – 340 с. – Иллюстраций: 130.

5. Паронджанов В.Д. Учись писать, читать и понимать алгоритмы. Алгоритмы для правильного мышления. Основы алгоритмизации. — Москва: ДМК Пресс, 2012. — 520 с. — Иллюстраций: 272.

6. Паронджанов В.Д. Дружелюбные алгоритмы, понятные каждому. (Как улучшить работу ума без лишних хлопот). — М.: ДМК Пресс, 2010. — 464 с. — Иллюстраций 233.

7. Паронджанов В.Д. Как улучшить работу ума. Алгоритмы без программистов — это очень просто! — Москва: Издательство Дело, 2001. — 360 с. — Иллюстраций: 154.

8. Паронджанов В.Д. Как улучшить работу ума. (Новые средства для образного представления знаний, развития интеллекта и взаимопонимания). — Москва: Издательство Радио и связь, 1998, 1999. — 352 с. — Иллюстраций: 154.

9. В 1996 г. 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. https://drakon.su/_media/biblioteka/progr_drakon.pdf