ДВУМЕРНОЕ СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ; КЛАСС УСТРЕМЛЁННЫХ ГРАФОВ (ТЕОРЕТИЧЕСКИЕ ИЗЫСКАНИЯ ИЗ ОПЫТА ЯЗЫКА «ДРАКОН»)

Ермаков И.Е., технический директор, преподаватель
НПО «Тесла»; Технологический институт ОрёлГТУ
Жигуненко Н.А., студент 3 курса
УНИ «Институт информационных технологий» ОрёлГТУ
ermakov@metasystems.ru

В докладе раскрыт вопрос алгоритмических структур, применяемых при двумерном визуальном программировании, показаны свойства конструкций визуального языка «ДРАКОН», введены некоторые новые понятия и определения, которые представляют интерес с точки зрения теории алгоритмов и схем программ и имеют практическую пользу с точки зрения новых способов структурирования алгоритмов, что актуально для задач системного и встроенного ПО и прикладной логики АСУ.

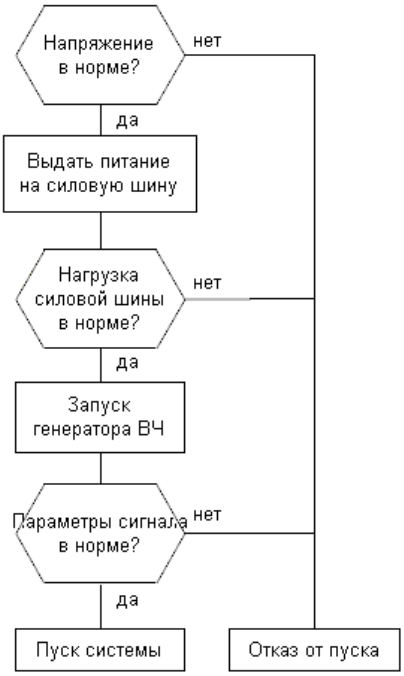
1. Структурное программирование и дисциплина программирования. Структурное программирование обусловило некогда революцию в разработке ПО за счёт обеспечения нового качества проектирования систем, на основе нескольких базовых структур, обладающих хорошо предсказуемыми свойствами.

Следующим этапом после структурного программирования стала фундаментальная работа Эдсгера Дейкстры «Дисциплина программирования» [1], в которой был показан способ убедительно рассуждать о свойствах создаваемых программ, получая сразу правильные конструкции (с точностью до опечаток). Основой для рассуждения являются утверждения о состоянии программы между операторами — предусловия, постусловия и имеющие особое значение для программирования инварианты циклов. Новизной метода Дейкстры, по сравнению с предшественниками, стало обратное рассуждение от постусловия («от цели к необходимым предпосылкам»), которое позволяет вместо громоздких доказательств «постфактум» непосредственно строить правильные программные блоки из рассуждений (значительная часть которых на практике может быть проведена нестрого или вообще в уме). Эти особенности обусловили практичность метода Дейкстры и его частных проекций (например, базовых схем циклов [2]), которые являются каждодневным подспорьем для грамотного программиста.

Заметим, что метод Дейкстры не связан напрямую со структурным программированием в варианте «трёх базовых конструкций», хотя, разумеется, без перехода к упорядоченной организации программ его открытие не представлялось возможным. **Главная ценность, которую мы должны вынести из упомянутых концепций, - это идея о необходимости инженерной культуры при разработке программ, которая подразумевает возможность рассуждать и отвечать за свойства создаваемых конструкций.**

2. Проблемы классического структурного программирования. На практике приходится сталкиваться со случаями, в которых структурное программирование приводит к плохому выражению алгоритмической логики. Такие случаи характерны в первую очередь для алгоритмов управляющего, реактивного, событийного характера (в противовес алгоритмам информационно-преобразующего, вычислительного характера).

Рассмотрим первый пример — последовательность действий, перемежающаяся проверками успешности. Результатом такой последовательности могут быть два исхода — успех и отказ. Ситуация типична для системного программирования. Сначала приведём фрагмент алгоритма на визуальном языке ДРАКОН, поскольку он с наибольшей точностью выражает суть дела (рис. 1а). Далее приведём пример кодирования в структурном стиле, который «болеет» глубокой вложенностью, многократным дублированием и неудобством добавления новых проверок (рис. 1б). И, наконец, приведём популярный в системном программировании способ обойти эти недостатки, путём введения явной переменной состояния (рис. 1в).




```
IF напряжение в норме THEN
  выдать питание на силовую шину;
  IF нагрузка силовой шины в норме THEN
    запуск генератора ВЧ;
    IF параметры сигнала в норме THEN
      пуск системы
    ELSE
      отказ от пуска
    END
  ELSE
    отказ от пуска
  END
ELSE
  отказ от пуска
END
```

```
VAR усп: BOOLEAN;
...
усп := напряжение в норме;
IF усп THEN
  выдать питание на силовую шину;
  усп := нагрузка силовой шины в норме
END;
IF усп THEN
  запуск генератора ВЧ;
  усп := параметры сигнала в норме
END;
IF усп THEN
  пуск системы
ELSE
  отказ от пуска
END
```

Рисунок 1а *Рисунок 1б* *Рисунок 1в*

Для управляющих алгоритмов характерно то, что каждый алгоритмический блок может вырабатывать некоторое решение, завершаться наборов *исходов*. Приведённый пример — простейший, блок имеет два исхода «успех» - «неуспех». В более сложных случаях исходов у блока гораздо больше. Смысл слова «блок» в данном случае иной, нежели принят в структурном программировании. Блок — это некая замкнутая часть системы, устанавливающая после своего завершения различимый для окружения результат.

Второй пример связан с циклическим алгоритмом, точно так же имеющим два исхода. Это обычный линейный поиск (являющийся второй базовой схемой циклов [2]). Приведём сначала запись в классическом виде (Рисунок 2а), а затем изобразим её с помощью визуальной нотации ДРАКОНа (Рисунок 2б).

```
WHILE ~(конец последовательности поиска) &
  ~ (текущее состояние удовлетворяет условию поиска) DO
  перейти к следующему состоянию
END;
IF ~(конец последовательности поиска) THEN
  состояние найдено
ELSE
  состояние не найдено
END
```

Рисунок 2а

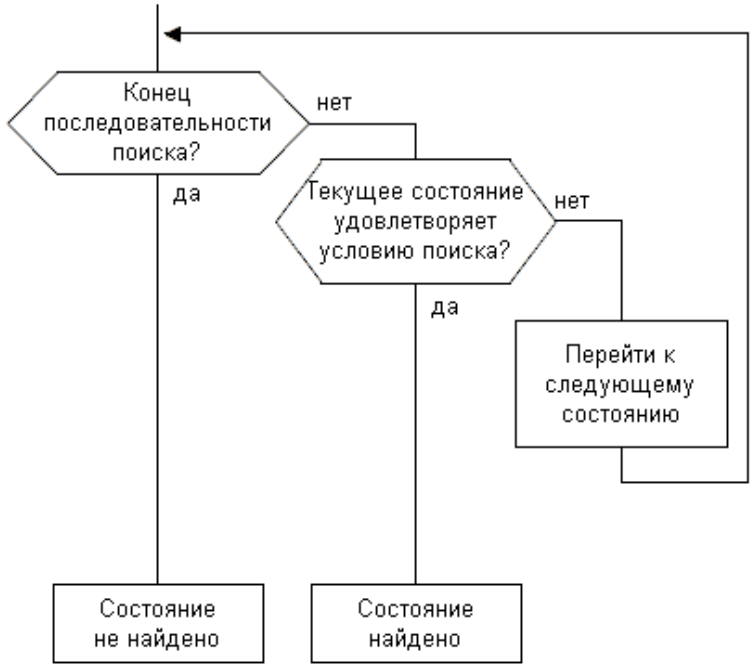


Рисунок 2б

Мы видим, что в случае текста опять присутствует дублирование проверки условия. Визуальная же схема ярко отражает семантику происходящего: вычисление конъюнкции отрицания условий представляется алгоритмическим блоком из двух проверок с тремя исходами. Перечислим эти исходы слева направо: 1 — первое условие истинно, второе не вычислялось, 2 — первое условие ложно, второе истинно, 3 — оба условия ложны. Эти исходы соответствуют принятию одного из трёх решений: 1 — искомый элемент отсутствует в последовательности, 2 — искомый элемент обнаружен и совпадает с текущим, 3 — необходимо продолжить поиск. В случае же текстовой записи с помощью цикла WHILE вся информация о том, по причине какого конъюнкта завершился цикл, просто теряется, маршруты вычислений сходятся в одной точке выхода из цикла, после чего необходимо вновь выполнять распознавание ситуации.

3. К двумерному структурному программированию. Приведённые примеры являются простейшей иллюстрацией следующих тезисов:

– возможный маршрут выполнения блока алгоритма непосредственно связан с возможными семантическими состояниями системы;

– если разрешённые топологии маршрутов сильно ограничены, то выражать семантику состояний системы приходится либо путём дублирования

фрагментов, либо с помощью дополнительных переменных состояния (проявление более общего системного принципа, что проблему, идущую от природы задачи, нельзя победить формально, она просто окажется вытесненной на другой уровень формы);

– лучше, если топология маршрутов в алгоритме позволяет выражать семантику состояний системы (в частности, возможные исходы блока) явно;

– как следствие, следует каким-либо образом разрешить работу с несколькими выходами из блока;

– поскольку текст является одномерным, невозможно удовлетворительно представлять в нём более сложные топологии, нежели классическая структурная (линейно-блочный граф), поэтому их запрещение является, видимо, наилучшим решением;

– используя двумерную графическую нотацию, возможно удовлетворительно представлять более широкий класс топологий, чем классическая структурная. Последовательность действий выстраивается по вертикали сверху вниз, переключение семантических состояний системы (переходы между маршрутами) выполняются по горизонтали слева направо (расщепление) и справа налево (объединение).

4. Обзор языка ДРАКОН. Язык ДРАКОН - «Дружелюбный Русский Алгоритмический Язык, Который Обеспечивает Наглядность/Надёжность» - разработан В.Д. Пароннджановым, описан в ряде книг и статей [3, 4, 5], является продуктом проектов космической промышленности, выполняющихся в НПЦ Автоматики и приборостроения им. Н.А. Пилюгина. ДРАКОН применяется в рамках специализированной CASE-системы ГРАФИТ-ФЛОКС для комплексной разработки ПО управляющих систем российских ракет-носителей и разгонных блоков серий «Ангара», «Протон», «Фрегат» и ряда других. В этом качестве язык позволил минимизировать участие программистов в разработке управляющих алгоритмов для изделий, переложить разработку алгоритмов на инженеров-комплексников и резко повысить производительность коллективной разработки.

Язык разработан в качестве инженерного инструмента, его правила выработаны эмпирическим и эстетическим образом. В настоящее время ДРАКОН позиционируется автором как общекультурный инструмент для формализации процедурных знаний в любых областях деятельности [3], а также как «первая ласточка» в области когнитивной эргономики — направления, занимающегося оптимизацией инструментов умственного труда.

С правилами языка можно ознакомиться из литературы, мы же выделим основные особенности:

–ДРАКОН решает рассмотренные выше проблемы текстового структурного программирования и является примером двумерного структурного программирования;

–взаимное расположение элементов схемы определено формально, однозначным образом, поэтому ДРАКОН-схема является не рисунком, допускающим произвольную компоновку (как обычные блок-схемы), а строгой

структурой;

–схемы организованы асимметрично (см. на примеры выше) — имеется главная ось («шампур») и побочные маршруты, на которые происходит переход слева направо;

- ДРАКОН-схема способствует быстрому изучению и восприятию алгоритма, за счёт оптимизации в соответствии с требованиями когнитивной эргономики. Если говорить о проверке формальных свойств алгоритма, то преимуществом визуальной схемы является возможность быстрого ответа на вопрос, какими маршрутами может прийти управление в некоторую точку алгоритма, какие варианты поведения системы возможны после этой точки;
- в языке присутствует конструкция, непосредственно соответствующая конечному автомату (силуэт), что представляет большую ценность для программирования управляющих систем;
- автор языка не даёт аксиоматического определения того, что есть правильная ДРАКОН-схема, однако вводит строгое конструктивное определение (понятие элементарной схемы и допустимые преобразования, которые сохраняют правильность схемы).

Изложенные далее соображения по структурам алгоритмов были спровоцированы идеями языка ДРАКОН. Можно говорить о том, что автор ДРАКОНа сконцентрировал в нём эмпирическим путём ряд принципов, имеющих под собой весьма фундаментальное основание.

5. Алгоритм как грамматика процессов. В работе Эдсгера Дейкстры [6, Глава 1] понятие алгоритма вводится совершенно нетипичным образом: алгоритм трактуется как шаблон возможных процессов. Важность этого определения в том, что оно выбивается из ряда формальных, вычислительных определений, отражает управляющую природу алгоритмов. Такой подход является не чисто математическим, а в большей мере естественнонаучным, соответствующем взгляду системного анализа, при котором алгоритм — это описание поведения системы.

При таком понимании алгоритма трудно не заметить точной аналогии с теорией формальных языков, в которой имеются понятия языка над алфавитом, заданного формальной грамматикой, и цепочек литер алфавита, принадлежащих языку. В случае с алгоритмом понятию языка соответствует понятие поведения системы, понятию алфавита — множество возможных событий, понятию грамматики — понятие алгоритма. Имеется некоторое возможное поведение, заданное алгоритмом, и цепочки событий (процессы), допускаемые данным поведением. Точно так же, как для формальных грамматик, понятие алгоритма можно наделять как генерационной семантикой (способностью порождать при выполнении цепочки событий), так и распознавательной семантикой (возможностью распознавать по алгоритму допустимость цепочки событий).

В теории схем программ [7] алгоритм традиционно рассматривается как граф с двумя видами вершин — преобразователями (имеют одну исходящую

дугу) и распознавателями (имеют несколько исходящих дуг), с выделенной начальной вершиной. Конечными вершинами являются все, из которых нет исходящих дуг, хотя может быть добавлена выделенная конечная вершина. Процесс выполнения алгоритма соответствует некоторому пути в графе от начальной вершины до одной из конечных.

Граф алгоритма может быть представлен различным образом, с разными ограничениями на топологию. Мы предлагаем считать отправной нормальной формой для алгоритма хорошо известную в теории схем программ полную древесную развёртку алгоритма. Чтобы её построить, достаточно начать изображать алгоритм от начальной вершины и никогда не допускать объединения маршрутов, т. е. в случае схождения нескольких дуг в одной вершине графа продублировать для каждой дуги эту вершину и все достижимые из неё. В случае наличия в алгоритме циклов, мы, очевидно, получим бесконечное дерево рекурсивной структуры, с наличием потенциально бесконечных путей в нём.

Такое представление делает явной семантику различных ситуаций в поведении системы. Ситуация соответствует некоторому узлу дерева. Каждая ситуация — это множество возможных цепей событий, которые могут развернуться из неё.

Тогда любую алгоритмическую нотацию будем рассматривать как компактную запись, упаковку нормальной формы алгоритма (полной древесной развёртки). В зависимости от ограничений нотаций, упаковка может выполняться естественно или неестественно, с частичным сохранением дублирования (что мы и наблюдали на примерах в начале статьи). Обязательное свойство упаковки алгоритма — упакованный граф порождает множество путей, эквивалентное исходному.

6. Упаковка алгоритма в устремлённый граф. Рассмотрим следующий способ упаковки нормальной формы алгоритма. Рекурсивные бесконечные окончания, в тех случаях, когда это возможно, устраняется посредством введения в граф обратных дуг, т. е. образования циклов. Далее устраняется дублирование эквивалентных окончаний цикла, путём объединения вершин. Результатом упаковки является ориентированный циклический граф с выделенной начальной вершиной. Назовём класс графов, которые могут быть получены из бесконечных рекурсивных деревьев путём такой упаковки, **устремлёнными графами**.

Можно показать, что класс устремлённых графов эквивалентен известному классу сводимых графов [8, 9], который, как известно, эквивалентен классу аранжируемых графов. Опорным для доказательства эквивалентности устремлённых и сводимых графов является свойство деления дуг графа на два непересекающихся класса — прямые и обратные. Теорема Евстигнеева [8, Теорема 14, с. 89] гласит, что граф сводим тогда и только тогда, когда такое деление может быть выполнено (множество прямых дуг даёт ациклический подграф-носитель общего графа).

Ценным для практики свойством устремлённых графов является то, что обратная дуга цикла может идти только в вершину, являющуюся доминатором

для той, из которой исходит эта обратная дуга (доминатором для вершины X называется любая вершина D, через которую проходят все пути в X из начальной вершины).

Благодаря этому свойству топология устремлённых графов имеет большую ценность при проектировании структур систем. С точки зрения системного анализа мы называем это свойство **управляемостью структуры** — из двух вершин, соединённых дугой, однозначно можно выбрать «старшую» и «младшую», относительно начальной вершины в графе.

Очевидно, что простой, но наиболее ограниченной управляемой топологией является дерево. Более свободная топология — направленный ациклический граф (DAG). Также свойством управляемости обладает линейно-блочная топология, принятая в классическом структурном программировании (в ациклическом случае она ещё более ограничена, чем DAG, но в общем случае может допускать циклы). Наконец, наиболее общим классом относительно свойства управляемости является класс устремлённых графов.

Видимо, ограничения устремлённости являются наиболее свободными из тех, которые можно наложить на топологию алгоритмов, без утраты их приемлемой структурированности.

7. Аксиоматика ДРАКОН-схем. Оказывается, что топология ДРАКОН-схем (в алгоритмах-примитивах и внутри каждой ветки алгоритмов-силуэтов) соответствует классу устремлённых (сводимых, аранжируемых) циклических ориентированных графов, с дополнительно наложенным ограничением планарности (укладки на плоскости без пересечений). Таким образом, эмпирически полученный В.Д. Пароджановым визуальный язык и его конструктивное определение оказываются эквивалентными самому общему классу графов, которые обладает свойством управляемости, т. е. остаются в достаточной мере структурированным для построения алгоритмов с легко предсказуемыми свойствами. ДРАКОН-топологию можно рассматривать как оптимальное расширение классической структурной, предназначенное для использования в двумерном визуальном программировании.

Важной возможностью является однозначное формальное (т. е. автоматическое) взаимное размещение элементов таких алгоритмических схем. Авторами исследованы и применяются на практике при разработке САПР свойства и алгоритмы, позволяющие выполнять такое размещение для планарных устремлённых графов.

Литература

1. Дейкстра, Э. Дисциплина программирования. / Э. Дейкстра. — М. : Мир.— 1978. — 276 с.

2. Ермаков, И.Е.; Рюмшин, Б.В. О грамотной алгоритмизации. // Сборник докладов секции «Информатика образования» VII межд. конф-и памяти акад. А.П. Ершова «Перспективы систем информатики». — Новосибирск, 2009. Web: http://metasystems.ru/download/edu/I21-a-057a-Novosib-PSI-2009-eierbv.pdf

3. Пароджанов, В.Д. Как улучшить работу ума: Алгоритмы без программистов — это очень просто! — М.: Дело. — 2001. — 360 с.

4. Страница и форум языка ДРАКОН // Проект OberonCore. Web: http://oberoncore.ru/wiki/drakon/start

5. Шамардина, Е.И.; Манюнин, П.А. Язык программирования «ДРАКОН» и его применения за пределами ракетно-космических проектов. Разработка математической модели и редактора. // Актуальные проблемы российской космонавтики: Труды XXXIV Академических чтений по космонавтике. Москва, январь 2010 г. — М. — 2010. — с. 494-495.

6. Дейкстра, Э. Краткое введение в искусство программирования. / Э. Дейкстра. — EWD316. Пер. с англ. А.С. Деревянко. Web: http://khpi-iip.mipk.kharkiv.edu/library/extent/dijkstra/ewd316/index316.html

7. Котов, В.Е.; Сабельфельд, В.К. Теория схем программ. / В.Е. Котов, В.К. Сабельфельд. — М.: Наука. — 1991. — 248 с.

8. Евстигнеев, В.А. Применение теории графов в программировании. / В.А. Евстигнеев. — Под ред. А.П. Ершова. — М.: Наука. — 1985. — 352 с.

9. Касьянов, В.Н.; Евстигнеев, В.А. Графы в программировании: обработка, визуализация, применение. / В.Н. Касьянов, В.А. Евстигнеев. — СПб.: БХВ-Петербург. — 2003. — 1104 с.

10. Калентьев, А.А.; Тюгашев, А.А. ИПИ/CALS технологии в жизненном цикле комплексных программ управления. / А.А. Калентьев, А.А. Тюгашев. — Самара: Изд-во Самарского научного центра РАН. — 2006. — 285 с.

11. Тюгашев, А.А. Графические языки программирования и их применение в системах управления реального времени. / А.А. Тюгашев. — Самара: Изд-во Самарского научного центра РАН. — 2009.