Введение в программирование: Марковский алгорифм, грамматики и конечные автоматы

Д.Г. Штукенберг

10 июля 2023 г.

Аннотация

Данный курс в той или иной форме рассказывается новичкам, поступающим в ЛНМО; обычно это происходит в летней школе. Курс предназначен для первоначального ознакомления с предметом. Данный документ содержит краткий конспект теоретической части курса.

1 Алгорифм Маркова

Для начала, прежде чем мы перейдем к языкам программирования, использующимся на практике, мы познакомимся с *Нормальным алгорифмом* — моделью вычислительной машины, предложенной в середине XX века советским математиком Андреем Андреевичем Марковым. Эту модель еще называют *Нормальным алгорифмом Маркова* или просто *Марковским алгорифмом*.

Модель — это некоторое упрощенное представление действительности, в котором убраны все несущественные подробности и оставлено только то, что реально важно. Модель вычислительной машины, соответственно — это крайне простая вычислительная машина, обычно существующая только на бумаге и в воображении, не очень удобная для реальных применений, но незаменимая для исследования программ «в общем». Мы будем использовать Марковский алгорифм для в чем-то похожей цели — для первого знакомства с языками программирования. Здесь тоже важно, чтобы мелкие частности не закрывали общей картины.

Синтаксисом языка мы будем называть правила, определяющие допустимые тексты на этом языке. Слово «правила» мы пока понимаем в самом общем смысле, никак не ограничивая.

Семантика языка — это способ понимания того, что в тексте на данном языке записано, ведь одно и то же слово может на разных языках значить разное. Скажем, слово «яма» принадлежит и русскому и японскому языку. Тем не менее, смысл (семантика) этих слов различен: «яма» по-русски означает яму, а по-японски — гору.

1.1 Синтаксис Марковского алгорифма

Рассмотрим синтаксис Марковского алгорифма, для чего дадим несколько определений.

Определение 1.1. *Алфавит* — конечное множество символов.

Пример 1.1. *{a,b,c} или {0,1,2,3,4,5,6,7,8,9}.*

Определение 1.2. *Слово* или *строка* — конечная последовательность символов. Строка в алфавите A — конечная последовательность символов из алфавита A. Строка может быть *пустой* — то есть, не содержащей ни одного символа. Термины слово и строка равнозначны.

Пример 1.2. *abba, aaaab — слова в алфавите {a, b}.*

Слово в данном контексте — это очень формальный объект. Оно не обязано быть словом русского или какого-то другого естественного языка и иметь какой-либо смысл. Оно может содержать пробелы или знаки пунктуации, если эти знаки входят в алфавит. Словом может быть даже целая страница текста на русском языке.

Слово, не содержащее никаких символов, мы назовём *пустым*. Для обозначения пустой строки часто используется символ ε. Понятно, что слово, состоящее только из пробелов, и пустое слово различны. Также различны слова из двух и из пяти пробелов.

Определение 1.3. *Правило* — это запись вида

$$\alpha \quad \longrightarrow \quad \beta$$

После стрелки может быть указана точка, такое правило мы назовем *правилом с точкой*:

$$\alpha \quad \longrightarrow . \quad \beta$$

Здесь α и β — слова, состоящие из символов алфавита A. Слово α будем называть левой частью правила, а слово β — правой частью.

Пример 1.3. *В алфавите* $\{a,b,c\}$ *следующие записи являются примерами правил:*

$$\begin{aligned} a \quad &\longrightarrow \quad b \\ aaa \quad &\longrightarrow . \quad bbb \end{aligned}$$

Естественно, слова могут быть пустыми, поэтому следующие правила вполне допустимы:

$$\begin{aligned} a \quad &\longrightarrow \\ &\longrightarrow . \quad a \\ &\longrightarrow \end{aligned}$$

Чтобы понимать друг друга, мы должны различать предметный язык и метаязык. *Предметный язык* — это изучаемый язык; в случае алгоритмов Маркова это тексты программ на алгорифме Маркова. *Метаязык* — наши комментарии, общие замечания и т.п., служащие пониманию текстов программ. Отсюда два соглашения:

1. Нет никаких препятствий для вхожения точки и символа стрелки в алфавит, но, чтобы избежать путаницы с записью правил, мы будем этого избегать. В самом деле, если в качестве алфавита взять $\{\longrightarrow, .\}$, то следующую строку можно прочесть четырьмя разными способами (какими?)

$$\longrightarrow . \longrightarrow\longrightarrow$$

2. По умолчанию, мы будем использовать греческие буквы $\alpha, \beta, \gamma, \ldots$ для метаязыка, а для алфавита Марковского алгорифма использовать цифры, а также латинские и русские буквы. Например, повторим определения правила с точкой: это всякая запись вида

$$\alpha \quad \longrightarrow . \quad \beta$$

Определение 1.4. *Схемой Марковского алгорифма* назовем упорядоченную пару $\langle$A,R$\rangle$, где A — это алфавит, а R — упорядоченный набор правил. Всегда предполагается, что набор занумерован.

Пример 1.4. *Алфавит* $\{а,б,в\}$. *Правила:*

$$\begin{aligned} (1)\ а \quad &\longrightarrow \quad б \\ (2)\ б \quad &\longrightarrow \quad в \end{aligned}$$

Поскольку по упорядоченному списку нумерацию всегда легко восстановить, то при записи мы будем часто ее опускать, записывая вышеуказанные правила вот так:

$$\begin{aligned} \texttt{а} &\longrightarrow \texttt{б} \\ \texttt{б} &\longrightarrow \texttt{в} \end{aligned}$$

1.2 Семантика Марковского алгорифма

Марковский алгорифм всегда применяется к какой-то начальной строке. Работа Алгорифма состоит из последовательности шагов, каждый шаг изменяет исходную строку, и эта измененная строка служит исходной строкой для следующего шага.

Один шаг марковского алгорифма устроен так:

- Найдем правило с минимальным номером, такое, что его левая часть является подстрокой исходной строки.

  Пример 1.5. *Рассмотрим такой набор правил:*

  $$\begin{aligned} (1)\ \textit{р} &\longrightarrow \textit{с} \\ (2)\ \textit{ааа} &\longrightarrow \textit{б} \\ (3)\ \textit{а} &\longrightarrow \textit{в} \\ (4)\ \textit{к} &\longrightarrow \textit{л} \end{aligned}$$

  и такую входную строку:

  $$\textit{кбааб}$$

  Первое правило, которое может быть применено - это правило (3), поскольку строки $\textit{р}$ *(правило 1) и* $\textit{ааа}$ *(правило 2) в строку* $\textit{кбааб}$ *не входят.*

- Найдем в строке самое левое (то есть первое) вхождение левой части правила. Заменим это вхожение на правую часть правила.

  Пример 1.6. *Первое вхождение подстроки* $\textit{а}$ *в строку* $\textit{кбааб}$ *подчеркнуто:* $\textit{кб}\underline{\textit{а}}\textit{аб}$*. Результат замены - строка* $\textit{кбваб}$*.*

  Строка, получившаяся в результате замены и будет являться результатом одного шага Алгорифма.

На каждом шаге Алгорифм получает новую строку, после чего Алгорифм делает следующий шаг, преобразуя уже эту новую строку. Процесс повторяется, пока не произойдет одно из двух событий:

- Не нашлось ни одного правила, которое можно было бы применить. В этом случае Алгорифм останавливается, и результатом его работы будет исходная для данного неудачного шага строка.
- Применено правило с точкой. В этом случае Алгорифм также прекращает работу, и результатом его работы будет результат применения этого правила. Это единственное отличие правил с точкой от обычных правил.

Пример 1.7. *Рассмотрим приведенный выше набор правил и исходную строку* **кбааб***. На первом шаге применяем правило (3), получаем строку* **кбваб***, на втором — снова правило (3), получаем строку* **кбввб***, на третьем — правило (4), получаем строку* **лбввб***. Поскольку ни одно из четырех правил к строке* **лбввб** *не применить, то Алгорифм остановится и эта строка и будет результатом его работы.*

Пример 1.8. *Алфавит {а,б,в}, правила:*

$$\begin{aligned} а &\longrightarrow б \\ в &\longrightarrow . \\ бб &\longrightarrow в \end{aligned}$$

На вход подается слово **абба***. Алгорифм последовательно получит следующие слова и остановится, выполнив правило с точкой:*

$$абба \Rightarrow ббба \Rightarrow бббб \Rightarrow вбб \Rightarrow бб$$

Обратите внимание на символ $\Rightarrow$ — он означает «получается из», и предназначен для записи результата выполнения одного шага Алгорифма. Если же мы хотим указать, что строка P через какое-то количество шагов превратилась в строку Q, мы будем записывать это с использованием троеточия:

$$\texttt{абба} \Rightarrow \ldots \Rightarrow \texttt{в}$$

Не путайте эти стрелки со стрелками для записи правил!

Совершенно необязательно, что Алгорифм в принципе закончит свою работу. Например, если в качестве алфавита взять {0,1}, в качестве правил:

$$\begin{aligned} 0 &\longrightarrow 1 \\ 1 &\longrightarrow 0 \end{aligned}$$

и применить такой Алгорифм к строке 11, то получится бесконечная последовательность шагов

$$11 \Rightarrow 01 \Rightarrow 11 \Rightarrow 01 \Rightarrow 11 \Rightarrow \ldots$$

На этом описание Марковского алгорифма заканчивается, в нем больше нет никаких подробностей.

1.3 Задачи

Вы можете дополнить алфавит, указанный в условии задачи, любыми буквами, которые вам потребуются. В более сложных задачах алфавит вам потребуется полностью определить самим.

1. Сделать в алфавите {А,Л,М,Н,О,С,У,Х} из слова МУХА слово СЛОН. На других словах поведение алгорифма может быть любым.

2. Сделать из слова МЫШКА слово КОШКА.

3. В строке из алфавита {а,б} заменить все буквы а на б.

4. В алфавите {1,2,3} из пустой строки сделать строку 123. На других входных строках работа алгорифма может быть любой.

5. В алфавите {0,1} сделать из любой строки строку 0110.

6. Дан алфавит из двух цифр {0,1}. Отсортировать цифры во входной строке по возрастанию. Например: 1000100111 $\Rightarrow \ldots \Rightarrow$ 0000011111.

7. Дан алфавит из четырех цифр {0,1,2,3}. Отсортировать цифры во входной строке по возрастанию. Например: 12030011 ⇒ ... ⇒ 00011123.

8. Дана строка из скобок [и]. Заменить все скобки на соответствующие им парные. Пример: [][[[[⇒ ... ⇒]][]]]]

   Подсказка: алгорифм

   $$[\longrightarrow]$$
   $$] \longrightarrow [$$

   работать не будет, поскольку никогда не окончит своей работы.

9. Из строки, состоящей из повторяющейся последовательности букв БА (конечной, но неизвестной заранее длины) сделать строку с тем же самым количеством последовательностей АБ. Например: БАБАБАБА ⇒ ... ⇒ АБАБАБАБ

10. По строке из алфавита {0} проверить, имеет ли строка четную длину. Если да — то результатом работы алгорифма должна быть строка четная, если нет — строка нечетная.

11. *Правильная скобочная запись* — такая строка из символов (и), которую можно дополнить до корректного арифметического выражения с помощью цифр и знаков арифметических операций (добавлять скобки нельзя!). Например, запись (()()) — правильная, поскольку ее можно дополнить до корректного выражения таким образом: ((0+0)+(0+0)). Запись)(правильной не является, поскольку ни в каком арифметическом выражении скобки в таком порядке идти не могут. Также, пустая строка - правильная запись, поскольку 1 - соответствующее корректное арифметическое выражение.

    Задача: проверить, является ли данная на входе строка из алфавита {(,)} правильной скобочной записью. Если запись правильная, результатом работы алгорифма должна быть строка правильная, а если запись - неправильная, то - строка неправильная.

    Подсказка: алгорифм совсем не обязательно должен в лоб следовать определению. Вам, возможно, будет удобно найти более подходящее свойство скобочной записи, которое может быть легко проверено.

    Пример: (()) ⇒ ... ⇒ правильная.

12. Заменить число, записанное в двоичной системе счисления на соответствующее количество символов |. Например: 110 ⇒ ... ⇒ ||||||.

13. Поставить символ | посередине слова четной длины (слово в алфавите {а,б}). Например: абаб ⇒ ... ⇒ аб|аб

14. Переставить буквы слова в алфавите {а,б} в обратном порядке.

15. Слово называется *палиндромом*, если оно читается одинаково как справа налево, так и слева направо. Например, слова ТОПОТ и АННА палиндромы, а слово ТОПОР — нет.

    Проверить, является ли данное слово в алфавите {а,б} палиндромом, и если да — результатом должно быть слово палиндром, а если нет — слово не палиндром.

16. Проверить, что число в двоичной записи равно 0.

17. Проверить число в двоичной записи на чётность.

18. Проверить число в четверичной записи на делимость на 3 (222_4 делится на 3, но 221_4 — нет).

19. Прибавить 1 к двоичному числу.

20. Сложить два двоичных числа.

2 Грамматика

Определение 2.1. Чтобы задать грамматику, необходимо задать:

- Алфавит (A), состоящий из двух непересекающихся конечных множества символов, называемых алфавитами терминальных (A_T) и нетерминальных (A_H) символов:

$$A = A_T \cup A_H, \quad A_T \cap A_H = \varnothing$$

- Набор правил без точек в алфавите A, в каждом из которых которых левая часть содержит по крайней мере один нетерминальный символ
- Начальный символ — некоторый символ из A_H.

Определение 2.2. Будем говорить, что слово s задаётся грамматикой, если выполнены следующие два утверждения:

- Оно целиком состоит из терминальных символов данной грамматики;
- Существует последовательность применения правил, которая преобразует начальный символ в слово s. В данном случае мы разрешаем применять правила в произвольном порядке к произвольным вхождениям в строке, не требуя применения самого первого правила к самому левому вхождению.

Определение 2.3. Будем говорить, что язык задается грамматикой, если грамматика задает те и только те слова, которые входят в язык.

3 Сокращения записи

Для упрощения записи мы будем пользоваться следующими сокращениями, которые можно раскрыть в традиционную грамматику.

3.1 Альтернатива

Будем писать $\alpha \to \beta|\gamma$ вместо двух строчек

$$\begin{aligned} \alpha &\to \beta \\ \alpha &\to \gamma \end{aligned}$$

Данная запись означает «строчка α может быть преобразована либо в β, либо в γ».

3.2 Необязательная часть

Будем писать $\alpha \to [\beta]$ вместо двух строчек

$$\begin{aligned} \alpha &\to \\ \alpha &\to \beta \end{aligned}$$

Данная запись означает «строчка α может быть преобразована либо в β, либо в пустую строку».

3.3 Повторение

Будем писать $\alpha \to \{\beta\}^*$ вместо таких строчек

$$\begin{aligned} \alpha &\to S \\ S &\to \beta S \\ S &\to \varepsilon \end{aligned}$$

Здесь S — некоторый новый, ранее не встречавшийся в данной грамматике нетерминальный символ.

Данная запись означает повторение строчки β ноль или более раз.

Аналогично мы можем рассмотреть повторение строки один или более раз: $\alpha \to \{\beta\}^+$ вместо строчек

$$\begin{aligned} \alpha &\to S \\ S &\to \beta S \\ S &\to \beta \end{aligned}$$

Эти два типа повторения легко выражаются один через другой: как $\{\alpha\}^+$ можно представить как $\alpha\{\alpha\}^*$, так и $\{\alpha\}^*$ — как $\varepsilon|\{\alpha\}^+$.

4 Иерархия Хомского

Определение 4.1. Назовем грамматику неукорачивающей, если все правила в этой грамматике имеют в левой части не больше символов, чем в соответствующей правой части. Грамматика также может содержать правило вида $S \to \varepsilon$, но только если нетерминал S не содержится в правых частях правил.

Определение 4.2. Назовем грамматику бесконтекстной (контекстно-свободной), если все правила в этой грамматике имеют в левой части в точности один нетерминальный символ и ничего, кроме него.

Определение 4.3. Назовем грамматику построенной по регулярному выражению, если:

- Алфавит нетерминальных символов состоит в точности из одного символа
- Грамматика имеет единственное правило, левая часть которого — нетерминальный символ, а правая состоит только из терминальных символов. Также, в правой части разрешено использовать сокращения записи (операции «альтернатива», «необязательная часть» и «повторение»).

Определение 4.4. Будем говорить, что язык — неукорачивающий (бесконтекстный, автоматный), если он может быть задан соответствующей грамматикой.

Определение 4.5. Иерархия Хомского имеет 4 уровня:

Уровень в иерархии	Тип грамматики
0	Грамматика общего вида
1	Неукорачивающая
2	Бесконтекстная
3	Регулярная

Легко заметить, что если грамматика (язык) принадлежит уровню k иерархии Хомского, то он принадлежит и всем уровням, меньшим k.

5 Регулярные выражения и конечные автоматы

Определение 5.1. Конечным автоматом мы назовем совокупность:

- двух алфавитов — алфавита терминальных символов A_T и алфавита состояний A_C;
- таблицы переходов, сопоставляющей паре символов — терминальному и состоянию — некоторое новое состояние: $T : A_T \times A_C \to A_C$;
- начального состояния и множества допускающих состояний.

Определение 5.2. Шаг применения конечного автомата. Если автомат находится в некотором состоянии $s \in A_C$, то результатом применения автомата к некоторому символу $a \in A_T$ будет новое состояние $T[s, a]$.

Определение 5.3. Будем говорить, что автомат допускает некоторую строку, если последовательное применение автомата ко всем символам из строки в порядке их записи в слове приводит автомат из начального в допускающее состояние.

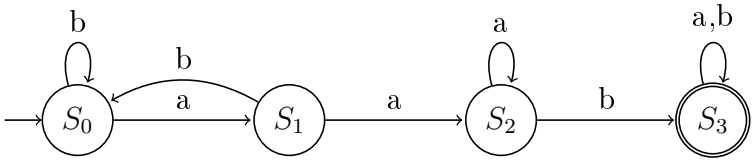
Определение 5.4. Будем говорить, что автомат задаёт язык, если он допускает те и только те слова, которые принадлежат языку.

5.1 Недетерминированные конечные автоматы

Введем альтернативный способ изображения конечных автоматов — в виде графа. Будем сопоставлять состояниям автомата точки, а клеткам таблицы переходов — дуги. Допускающие состояния будем обозначать двойным кружком, начальное состояние — входящей стрелкой. Если дуги для символа для какого-то состояния не указано, то появление такого символа ведёт к недопуску строки.

Пример задания одного и того же автомата с помощью таблицы и с помощью графа:

	a	b
S_0	S_1	S_0
S_1	S_2	S_0
S_2	S_2	S_3
S_3^*	S_3	S_3



Определение 5.5. Недетерминированный конечный автомат — это конечный автомат, в котором в каждой клетке таблицы переходов записано не одно состояние, а множество состояний. То есть, функция переходов задаётся иначе: $T : A_T \times A_C \to 2^{A_C}$.

Определение 5.6. Мы будем говорить, что недетерминированный конечный автомат допускает строку, если на каждом шаге применения автомата к символам входной строки (то есть, на символе $t \in A_T$ и в состоянии $s \in A_C$) найдётся такой переход (из множества допустимых переходов $T[t, s]$), что после последнего шага автомата он окажется в допускающем состоянии.

То же самое можно выразить несколько парадоксальной фразой: недетерминированный автомат на каждом шагу «угадывает», какой из вариантов перехода выбрать, чтобы достигнуть допускающего состояния.

Определение 5.7. Недетерминированный конечный автомат с эпсилон-переходами — конечный автомат, в котором к алфавиту терминальных символов добавлен «пустой» символ ε, не встречающийся во входной строке.

Определение 5.8. Назовём состояние r ε-достижимым из состояния s, если существует последовательность ε-переходов, позволяющая достичь r из s.

Определение 5.9. Мы будем говорить, что недетерминированный конечный автомат с ε-переходами допускает строку, если на каждом шаге применения автомата к символам входной строки (то есть, на символе $t \in A_T$ и в состоянии $s \in A_C$) найдётся такое ε-достижимое состояние r и такой переход из него (из множества допустимых переходов $T[t, r]$), что после последнего шага автомата допускающее состояние окажется ε-достижимо.

Иными словами, недетерминированный конечный автомат с ε-переходами может в любой момент самопроизвольно выполнить любое количество ε-переходов, если это ведёт к цели (к допуску строчки).

Теорема 5.1. *По любому недетерминированному автомату с ε-переходами можно построить детерминированный, задающий тот же язык*

Теорема доказывается путём рассмотрения автомата с алфавитом из 2^{A_C} состояний. Из этой теоремы следует, что классы языков, задаваемые и детерминированными и недетерминированными автоматами, одинаковы. Однако, в разных задачах удобен разный тип автоматов.

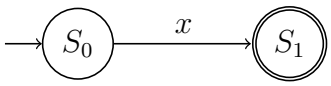
5.2 Соответствие регулярных выражений конечным автоматам

Известно, что грамматики, построенные по регулярным выражениям, задают в точности то же множество языков, что и конечные автоматы. Мы можем это показать, например, двумя включениями: показав, как заданный грамматикой язык задать конечным автоматом — и как язык, заданный конечным автоматом, задать с помощью грамматики, построенной по регулярному выражению.

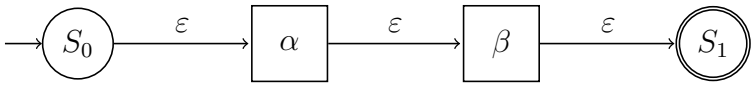
Мы покажем, как преобразовать регулярное выражение в недетерминированный конечный автомат, задающий тот же язык. Обратное преобразование также можно произвести, но оно останется за рамками данного курса.

В регулярном выражении возможны 4 типа конструкций: литерал (терминальный символ), конкатенация ($\alpha\beta$), альтернатива ($\alpha|\beta$) и повторение ($\{\alpha\}^+$). Нам достаточно предложить способ построения автомата для литералов и способы построения более сложных автоматов на основе уже имеющихся в остальных трёх типах конструкций.

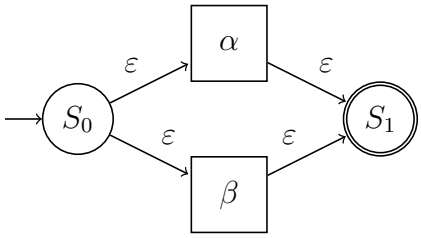
1. Литерал. Автомат, допускающий символ x (и только его), выглядит так:



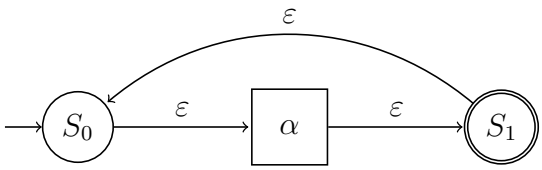
2. Конкатенация. Пусть даны два автомата, задающие α и β (они схематично изображены квадратами на графе). Тогда автоматом, задающим $\alpha\beta$, будет



3. Альтернатива. Пусть даны два автомата, задающие α и β. Тогда автоматом, задающим $\alpha|\beta$, будет



4. Повторение. Пусть дан автомат, задающий α. Тогда автоматом, задающим $\{\alpha\}^+$, будет



При помощи этих четырёх примитивов можно последовательно построить автомат, соответствующий любому регулярному выражению.