Автоматные грамматики и конечные автоматы

Содержание

Понятие регулярной грамматики

Как известно из прошлой лекции, наиболее простой тип грамматик - *автоматные грамматики*. Их правила определяются продукциями вида $A \to a|aB|\epsilon$. Здесь A, B - это один какой-либо нетерминальный символ из нетерминального алфавита, a - один какой-либо терминальный символ из терминального алфавита.
В более общем случае говорят о *регулярных грамматиках*, в которых символ a может состоять не из одного терминального символа, а быть цепочкой терминальных символов (обозначим его $\gamma \in \Sigma^*$). В свою очередь выделяют два класса регулярных грамматик: *праволинейная регулярная грамматика* (в правиле $A \to \gamma B$ нетерминальный символ B стоит справа); *леволинейную регулярную грамматику*, если в правиле $A \to B\gamma$ нетерминальный символ B стоит слева.
Доказано, что эти два класса грамматик эквивалентны: для любого регулярного языка, заданного праволинейной грамматикой, можно построить леволинейную грамматику, задающую тот же язык, и наоборот.
Разница между леволинейными и праволинейными грамматиками заключается в основном в том, в каком порядке строятся предложения языка: слева направо для леволинейных и справа налево для праволинейных.

Понятие регулярной грамматики

Рассмотрим простой пример: $S \to \epsilon|aaS|abT, T \to abT|bbS$. Данная грамматика, как видно праволинейная регулярная грамматика. Посмотрим, возможен ли в этой грамматике вывод цепочки $aaababbb$. Чтобы восстановить этот вывод, мы должны двигаться справа налево:

$$aaababbb \to aaababbb\epsilon \to aaababbbS \to aaababT \to aaabT \to aaS \to S$$

Данная цепочка вывода - это основная задача так называемого лексического анализатора. Как мы видим, он вынужден двигаться в своем выводе справа налево (каждый раз он заменяет на нетерминальный символ цепочку терминальных символов справа). Это не всегда удобно, так как лексическому анализатору нужно анализировать цепочку с конца входной строки. Поэтому обычно используют леволинейные регулярные грамматики.

В тоже время еще более простая жизнь для лексического анализатора наступает в случае автоматных грамматик, которые, как следует из определений, являются частным случаем регулярных грамматик. В чистых автоматных грамматиках обычно еще убирают правило $A \to \epsilon$ и оставляют только правила $A \to a|aA$. Аналогично, автоматные грамматики могут быть право и леволинейными.

Преобразование регулярной грамматики к автоматному виду

Оказывается, класс автоматных грамматик почти эквивалентен регулярным грамматикам. Для полной эквивалентности необходимо в автоматную грамматику добавить еще одно правило $S \to \epsilon$, где S - целевой символ, при этом символ S не должен встречаться в правых частях других правил. На практике реально используемые языки не содержат, как правило, пустую цепочку, поэтому автоматные и регулярные грамматики обычно отождествляют.

Существует алгоритм преобразования регулярной грамматики к автоматному виду. Рассмотрим данный алгоритм на примере введеной праволинейной регулярной грамматики: $S \to \epsilon|aaS|abT, T \to abT|bbS$. Очевидно, что каждое правило $S \to aaS, S \to abT, T \to abT, T \to bbS$ нужно представить цепочкой правил:

$$S \to aaS \Rightarrow S \to aS_1, S_1 \to aS$$

Аналогично поступаем в остальных случаях:

$$S \to abT \Rightarrow S \to aT_1, T_1 \to bT$$

$$T \to abT \Rightarrow T \to aT_2, T_2 \to bT$$

$$T \to bbS \Rightarrow T \to bT_3, T_3 \to bS$$

Чтобы устранить правило $S \to \epsilon$, нужно реализовать возможность получить $S \to aa$ и $T \to bb$, что также легко сделать: $S \to aS_2, S_2 \to a, T \to bT_4, T_4 \to b$

Преобразование регулярной грамматики к автоматному виду

Окончательно, грамматика с набором правил: $S \to aS_1, S_1 \to aS, S \to aT_1, T_1 \to bT, T \to aT_2, T_2 \to bT, T \to bT_3, T_3 \to bS, S \to aS_2, S_2 \to a, T \to bT_4, T_4 \to b$ является праволинейной автоматной грамматикой, которая порождает тот же язык, что и регулярная грамматика $S \to \epsilon | aaS | abT, T \to abT | bbS$.

С автоматными грамматиками жить становится действительно легче, поскольку порождаемые ими цепочки можно распознавать специальными устройствами-распознавателями, которые получили название **конечные автоматы**.

Неформальное определение конечного автомата

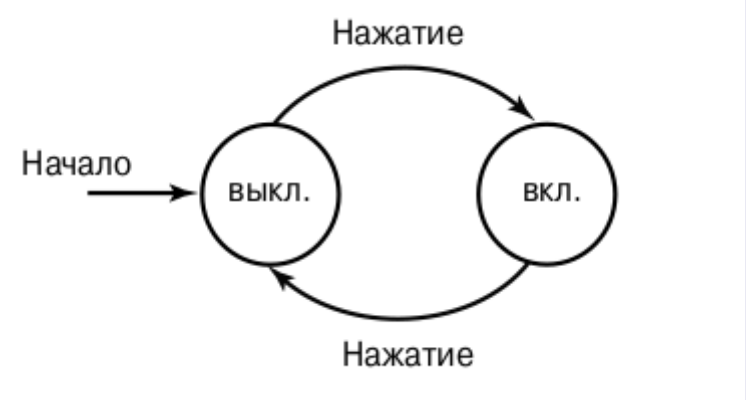
Конечные автоматы являются моделью для многих компонентов аппаратного и программного обеспечения. Ниже представлены наиболее важные примеры их использования:

1. Программное обеспечение, используемое для разработки и проверки цифровых схем.
2. Лексический анализатор стандартного компилятора, т.е. тот компонент компилятора, который отвечает за разбивку исходного текста на такие логические единицы, как идентификаторы, ключевые слова и знаки пунктуации.
3. Программное обеспечение для сканирования таких больших текстовых массивов, как наборы Web-страниц, с целью поиска заданных слов, фраз или других последовательностей символов (шаблонов).
4. Программное обеспечение для проверки различного рода систем (протоколы связи или протоколы для защищенного обмена информацией), которые могут находиться в конечном числе различных состояний.

В целом конечный автомат можно определить как устройство, которое под воздействием входного алфавита переходит в какое-либо состояние из некоторого конечного множества, учитывая, в каком состоянии это устройство находится в текущим момент времени.

Примеры конечных автоматов

Примером простейшего конечного автомата может быть кнопка включения, выключения - в зависимости от того, в каком состоянии кнопка, повторное нажатие может привести к включению или выключению света. Конечные автоматы удобно иллюстрировать в виде диаграммы переходов из состояния в состояние. Например, вот так можно представить автомат - «Кнопка».

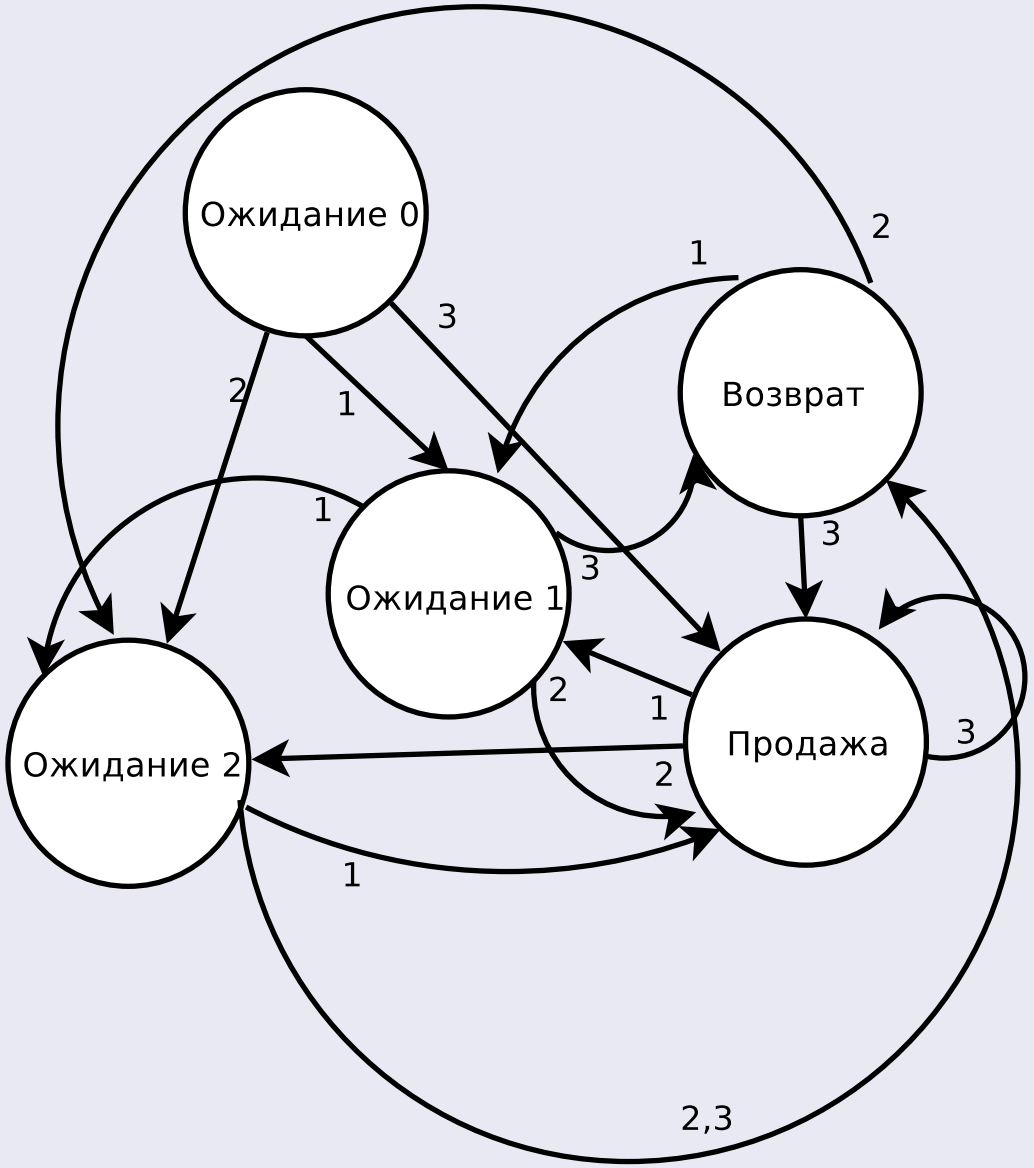


Более интересным примером является автомат по продаже газировки. Предположим, он работает по следующим правилам:

- Автомат распознает монеты достоинством 1, 2, 3 доллара.
- Если автомат получил в сумме 3 доллара, он продает газировку.
- Если автомат получил меньше 3 долларов, он ожидает.
- Если автомат получил больше 3 долларов, он возвращает деньги.

Примеры конечных автоматов

Ниже на рисунке показана диаграмма работы автомата по продаже газировки.



Формальное определение конечного автомата

Определение

Конечным автоматом *называют кортеж из пяти элементов:*

$$A =< Q, \Sigma, \delta, q_0, F >$$

Q *- конечное множество состояний автомата.*
Σ *- конечное множество входных символов (алфавит автомата).*
δ *- функция перехода из одного состояния в другое состояние под действием символа из* Σ.
q_0 *- начальное состояние автомата, в котором он находится перед началом работы* $q_0 \in Q$.
F *- непустое множество конечных состояний автомата, в которых он должен находиться по окончанию работы* $F \subseteq Q, F \neq \varnothing$

Автомат является полностью определенным, если в каждом его состоянии существует функция перехода для всех возможных входных символов.

Формальное определение конечного автомата

Работа конечного автомата представляет собой последовательность шагов. На каждом таким шаге автомат находится в одном из своих состояний, на следующем шаге он переходит в следующее состояние (может остаться в этом же состоянии) под действием входного символа. Такой переход задается функцией переходов δ. Работа автомата продолжается до тех пор, пока на его вход поступают символы. Если по окончанию входных символов автомат находится в одном из конечных состояний множества F, то говорят, что автомат распознал входную цепочку символов - она допустима в том языке, который понимает конечный автомат, в противном случае такая цепочка не распознается и считается недопустимой цепочкой этого языка.
Рассмотрим пример автомата по продаже газировки. Обозначим для краткости «Ожидание 0» состоянием A, «Ожидание 1» состоянием B, «Ожидание 2» состоянием C, «Возврат» состоянием D, «Продажа» состоянием F. Имеем:

$$\delta(A,1)=B, \delta(A,2)=C, \delta(A,3)=F, \delta(B,1)=C, \delta(B,2)=F, \delta(B,3)=D$$

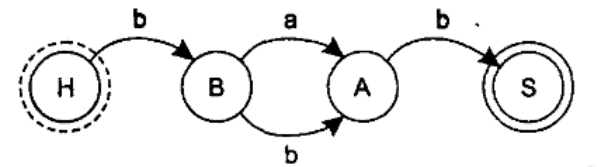
$$\delta(C,1)=F, \delta(C,2)=D, \delta(C,3)=D, \delta(D,1)=B, \delta(D,2)=C, \delta(D,3)=F$$

$$\delta(F,1)=B, \delta(F,2)=C, \delta(F,3)=F$$

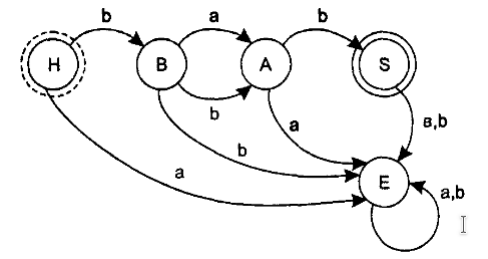
Как видно, автомат полностью определен,при этом $q_0=A$, а состоние F является единственным конечным состоянием, $\Sigma=\{1,2,3\}$.

Формальное определение конечного автомата

Конечный автомат часто представляют в виде диаграммы или графа переходов по аналогии с тем, как мы рисовали автоматы. Граф переходов конечного автомата - это направленный граф, в котором вершины помечены символами состояний автомата, а дуги помечены символами входного алфавита. Начальное состояние помечается дополнительной пунктирной линией, конечное состояние - дополнительной сплошной линией. На рисунке представлен пример.



Ясно, что этой диаграмме соответствует конечный автомат $A =< \{H, A, B, S\}, \{a, b\}, \delta, H, \{S\} >, \delta : \delta(H, b) = B, \delta(B, a) = A, \delta(B, b) = A, \delta(A, b) = S$
Однако, как видно данный автомат не полностью определен. На практике для полной определенности добавляют еще одно состояние, которое можно назвать «Ошибка». На это состояние замыкают все неопределенные переходы. Пример такой модификации представлен ниже:



Детерминированные и недетерминированные конечные автоматы

Важной особенностью функции переходов является ее однозначность - когда для одного и того же состояния под действием одного и того же символа автомат переходит в одно единственное состояние. В этом случае автомат называется детерминированным, в противном случае он называется недетерминированным. Для недетерминированных автоматов функция переходов задает множество возможных состояний, в которые может перейти автомат. Например, автомат с функцией переходов $\delta(H,b) = B, \delta(B,a) = A, \delta(A,b) = \{B,S\}$ является недетерминированным из-за перехода вида $\delta(A,b) = \{B,S\}$.

В принципе на практике часто используют недетерминированные конечные автоматы. Они являются более выразительными, однако для автоматизации операций синтеза анализаторов целесообразно переходить от недетерминированных конечных автоматов к детерминированным. Данный алгоритм мы рассмотрим позже, а сейчас рассмотрим, как по произвольному автомату получить порождающую грамматику.

Построение автоматной грамматики по конечному автомату

Задача формулируется таким образом: имеется конечный автомат $A =< Q, \Sigma, \delta, q_o, F >$, необходимо построить эквивалентную ему леволинейную грамматику $G(T, N, S, R)$.

1. Принимаем $T = \Sigma$.
2. Принимает $N = Q \setminus \{q_0\}$ - множество нетерминальных символов совпадает с множеством состояний автомата, исключая начальное состояние.
3. Для каждого перехода конечного автомата $\delta(A, t) = \{B_1, B_2, \ldots, B_n\}$ добавляем следующие правила:
   - Если $A = q_o$, то добавляем правила $B_i \to t, i = 1, \ldots, n$
   - Если $A \neq q_o$, то добавляем правила $B_i \to At, i = 1, \ldots, n$
   - Если $B_i = q_o$, то добавляем правила $S \to At, i = 1, \ldots, n$
4. Если множество конечных состояний содержит только одно состояние $F = \{F_0\}$, то целевым символом S принимаем F_0. Если $F = \{F_1, F_2, \ldots, F_n\}$, то целевым символом принимаем S и добавляем правило $S \to F_1, |F_2| \ldots |F_n$.

Пример построения леволинейной автоматной грамматики по конечному автомату

Преобразуем в качестве примера автомат по продаже газировки в автоматную грамматику:

$$\delta(A,1)=B,\delta(A,2)=C,\delta(A,3)=F,\delta(B,1)=C,\delta(B,2)=F,\delta(B,3)=D$$

$$\delta(C,1)=F,\delta(C,2)=D,\delta(C,3)=D,\delta(D,1)=B,\delta(D,2)=C,\delta(D,3)=F$$

$$\delta(F,1)=B,\delta(F,2)=C,\delta(F,3)=F$$

Имеем: $T=\{1,2,3\}, N=\{B,C,D,F\}, S=F$, у автомата состояние A принято за начальное.

- $B\to 1, C\to B1, S\to C1, B\to D1$
- $C\to 2, S\to B2, D\to C2, C\to D2$
- $S\to 3, D\to B3, D\to C3, S\to D3$
- $B\to S1, C\to S2, S\to S3$

Пример построения леволинейной автоматной грамматики по конечному автомату

Запишем полученные правила более компактно и попробуем получать цепочки данного языка:

$$S \to 3|B2|C1|D3|S3$$

$$B \to 1|D1|S1$$

$$C \to 2|B1|D2|S2$$

$$D \to B3|C2|C3$$

Первый пример синтеза цепочки: $S \to B2 \to D12 \to B312 \to 1312$

Второй пример синтеза цепочки:$S \to D3 \to B33 \to S133 \to S3133 \to C13133 \to 213133$

Построение конечного автомата по автоматной леволинейной грамматике

Возможен и обратный алгоритм:

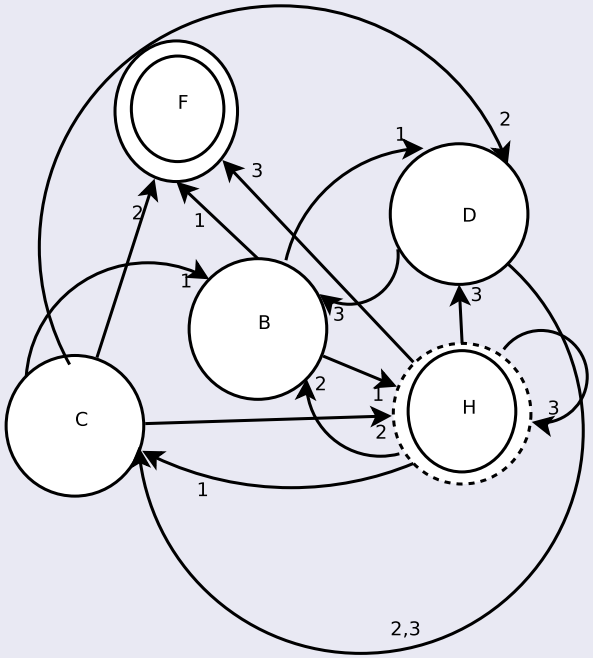
1. $Q = N \cup \{H\}$ - множество состояний совпадает с множеством нетерминалов с добавлением начального состояния.
2. $\Sigma = T$.
3. Если встречается правило $A \to t$, то в функцию переходов $\delta(H, t)$ добавляем состояние A.
4. Если встречается правило $A \to Bt$, то в функцию переходов $\delta(B, t)$ добавляем состояние A.
5. $q_0 = H$.
6. $F = \{S\}$.

Преобразование леволинейной автоматной грамматики в праволинейную

Как мы убедились, конечные автоматы, построенные по леволинейной грамматике, осуществляют разбор цепочки входного языка слева направо. Тогда, очевидно, что конечные автоматы, построенные по праволинейной грамматике, осуществлят разбор цепочки справо налево (с конца в начало). Автомат для праволинейной грамматики легко синтезировать имея автомат для соответствующей леволинейной грамматики. Для этого следуем поменять местами начальные и конечные состояния и направить стрелки в обратную сторону.

Преобразование леволинейной автоматной грамматики в праволинейную

Например, автомат по продаже газировки, разпознающий входную цепочку справа налево будет выглядеть следующим образом:



Как видно, в этом случае автомат стал недетерминированным. Функции переходов имеют вид:

$$\delta(H,1)=C, \delta(H,2)=B, \delta(H,3)=\{F,D,H\}, \delta(B,1)=\{F,D,H\}, \delta(C,1)=B,$$

$$\delta(C,2)=\{F,H,D\}, \delta(D,2)=C, \delta(D,3)=\{B,C\}$$

Преобразование леволинейной автоматной грамматики в праволинейную

Распознавание цепочки входного языка на недетерминированном автомате усложняется. Проведем в качестве примера распознавание цепочки 1312 на данном автомате. При этом цепочка считается распознана, если в множестве состояний находится хотя бы один раз финальное состояние по окончанию этой цепочки.
Итак, анализ первого символа: $\delta(H,2) = B$, анализ второго символа: $\delta(B,1) = \{F,D,H\}$, анализ третьего символа: $\delta(F,3) = \varnothing, \delta(D,3) = \{B,C\}, \delta(H,3) = \{F,D,H\}$, анализ четвертого символа: $\delta(B,1) = \{F,D,H\}, \delta(C,1) = B, \delta(F,1) = \varnothing, \delta(D,1) = \varnothing, \delta(H,1) = C$. Поскольку в окончательном выводе встретился символ F, то цепочка распознана. Можно проследить, в какие состояния нужно переходить при прямом синтезе на автомате леволинейной грамматики:

$$F \overset{1}{\leftarrow} B \overset{3}{\leftarrow} D \overset{1}{\leftarrow} B \overset{2}{\leftarrow} H$$

Преобразование леволинейной автоматной грамматики в праволинейную

Сама праволинейная грамматика строится по полученному конечному автомату аналогичным образом: Для функций перехода вида $\delta(A, t) = \{B_1, B_2, \ldots, B_n\}$

- Если $A = q_o$, то добавляем правила $B_i \to t, i = 1, \ldots, n$
- Если $A \neq q_o$, то добавляем правила $B_i \to tA, i = 1, \ldots, n$
- Если $B_i = q_o$, то добавляем правила $S \to tA, i = 1, \ldots, n$

В качестве примера получим праволинейную грамматику по соотвествующему конечному автомату (принято $F = S$):

$$C \to 1, B \to 2, S \to 3, D \to 3, S \to 1B, D \to 1B, B \to 1C$$

$$S \to 2C, D \to 2C, C \to 2D, B \to 3D, C \to 3D$$

Запишем эти правила более компактно и продемонстрируем вывод:

$$S \to 3|1B|2C, C \to 1|2D|3D, B \to 2|1C|3D, D \to 3|1B|2C$$

Демонстрация вывода: $S \to 2C \to 22D \to 222C \to 2223D \to 22231B \to 222312$.
Можно проверить, что эта цепочка распознается справа налево на недерминированном автомате, и слева направа на первичном детерминированном.

Задача

Придумать автоматную грамматику работы лифта (он считается трехэтажным, с первого этажа можно подняться на любой, с любого можно только спуститься на первый), построить по ней конечный автомат или решить обратную задачу.

Пример построения программного синтезатора на основе порождающей грамматики

```
import random
class gramm:
    __rules=[]
    def add_rule(self, S):
        self.__rules.append(S)
    def look(self):
        return self.__rules
    def deduct(self, L, n):
        while (n!=0):
            r=random.randint(0, len(self.__rules)-1)
            st=self.__rules[r]
            if L.find(st[0])!=-1:
                print(st)
                L=L.replace(st[0],st[1])
            n-=1
        return L
```

Алгоритм преобразования недетерминированного конечного автомата к детерминированному виду

Как мы увидели, во многих случаях иметь дело с недетерминированным автоматом не всегда удобно. Поэтому рассмотрим алгоритм преобразования недетерминированного автомата в детерминированный.

Пусть $A =< Q, \Sigma, \delta, q_0, F >$ - искомый недетерминированный автомат, требуется построить эквивалентный детерминированный $A^{'} =< Q^{'}, \Sigma^{'}, \delta^{'}, q_0^{'}, F^{'} >$.

1. Множество состояний Q' строим из всех комбинаций элементов множества Q - количество таких комбинаций равно $2^n - 1$, где n - количество элеметов в множестве Q. Будем обозначать $q' = [q_1, q_2, \ldots, q_m], 0 < m \leq n$.
2. Функция переходов $\delta^{'}(a, [q_1, q_2, \ldots, q_m]) = [r_1, r_2, \ldots, r_k],$, если существует $\delta(a, q_i) = r_j$.
3. $q_0^{'} = q_0$.
4. Если $f_1, f_2, \ldots, f_l$ - это конечные состояния автомата A, то $F^{'}$ строится из всех состояний $Q^{'}$, имеющих вид $[\ldots, f_i, \ldots]$

Рассмотрим данный алгоритм на примере нашего недетерминированного автомата.

Пример преобразования недетерминированного конечного автомата к детерминированному виду

Итак, у нас имеется 5 состояний:H, B, C, D, F. Необходимо перебрать все их сочетания по 1, 2, 3, 4, 5.
Сочетания по одному: H, B, C, D, F
Сочетания по два: $HB, HC, HD, HF, BC, BD, BF, CD, CF, DF$ (их количество $\frac{5!}{2!(5-2)!} = 10$
Сочетания по три: $HBC, HBD, HBF, HCD, HCF, HDF, BCD, BCF, BDF, CDF$ (их количество $\frac{5!}{3!(5-3)!} = 10$
Сочетания по четыре: $HBCD, HBCF, HCDF, BCDF, HBDF$ (их количество $\frac{5!}{4!(5-4)!} = 5$
Сочетания по пять: $HBCDF$
Переберем для всех этих состояний все их функции переходов.

Пример преобразования недетерминированного конечного автомата к детерминированному виду

$$\delta'(H,1)=C,\delta'(H,2)=B,\delta'(H,3)=HDF,\delta'(C,1)=B,\delta'(C,2)=HDF,$$
$$\delta'(B,1)=HDF,\delta'(D,2)=C,\delta'(D,3)=BC,$$
$$\delta'(HB,1)=HCDF,\delta'(HB,2)=B,\delta'(HB,3)=HDF,\delta'(HC,1)=BC,$$
$$\delta'(HC,2)=HCDF,\delta'(HC,3)=HDF,\delta'(HD,1)=C,\delta'(HD,2)=BC$$
$$\delta'(HD,3)=HBCDF,\delta'(HF,1)=C,\delta'(HF,2)=B,\delta'(HF,3)=HDF,$$
$$\delta'(BC,1)=HBDF,\delta'(BC,2)=HDF,\delta'(BC,3)=[],\delta'(BD,1)=HDF,$$
$$\delta'(BD,2)=C,\delta'(BD,3)=BC,\delta'(BF,1)=HDF,\delta'(BF,2)=[],\delta'(BF,3)=[],$$
$$\delta'(CD,1)=B,\delta'(CD,2)=HCDF,\delta'(CD,3)=BC,\delta'(CF,1)=B,\delta'(CF,2)=HDF,$$
$$\delta'(CF,3)=[],\delta'(DF,1)=[],\delta'(DF,2)=C,\delta'(DF,3)=BC,\delta'(HCD,1)=BC,$$
$$\delta'(HCD,2)=HBCDF,\delta'(HCD,3)=HBCDF,\delta'(HDF,1)=C,\delta'(HDF,2)=BC,$$
$$\delta'(HDF,3)=HBCDF,\delta'(HBC,1)=HBCDF,\delta'(HBC,2)=HBDF,$$
$$\delta'(HBC,3)=HDF$$

Пример преобразования недетерминированного конечного автомата к детерминированному виду

$$\delta'(HBD,1)=HCDF, \delta'(HBD,2)=BC, \delta'(HBD,3)=HBCDF, \delta'(HBF,1)=HCDF,$$
$$\delta'(HBF,2)=B, \delta'(HBF,3)=HDF, \delta'(HCF,1)=BC, \delta'(HCF,2)=HBDF,$$
$$\delta'(HCF,3)=HDF, \delta'(BCD,1)=HBDF, \delta'(BCD,2)=HCDF, \delta'(BCD,3)=BC,$$
$$\delta'(BCF,1)=HBDF, \delta'(BCF,2)=HDF, \delta'(BCF,3)=[], \delta'(BDF,1)=HDF,$$
$$\delta'(BDF,2)=C, \delta'(BDF,3)=BC, \delta'(CDF,1)=B, \delta'(CDF,2)=HCDF,$$
$$\delta'(CDF,3)=BC, \delta'(HBCD,1)=HBCDF, \delta'(HBCD,2)=HBCDF,$$
$$\delta'(HBCD,3)=HBCDF, \delta'(HCDF,1)=BC, \delta'(HCDF,2)=HBCDF,$$
$$\delta'(HCDF,3)=HBCDF, \delta'(HBDF,1)=HCDF, \delta'(HBDF,2)=BC,$$
$$\delta'(HBDF,3)=HBCDF, \delta'(HBCF,1)=HBCDF, \delta'(HBCF,2)=HBDF,$$
$$\delta'(HBCF,3)=HDF, \delta'(BCDF,1)=HBDF, \delta'(BCDF,2)=HCDF,$$
$$\delta'(BCDF,3)=BC, \delta'(HBCDF,1)=HBCDF, \delta'(HBCDF,2)=HBCDF,$$
$$\delta'(HBCDF,3)=HBCDF$$

Пример преобразования недетерминированного конечного автомата к детерминированному виду

Следующим важным этапом синтеза детерминированного автомата является исключение недостижимых состояний - в них никак нельзя перейти из начального состояния.
Начальным состоянием нашего автомата является состояние H, из него можно перейти в состояния C, B, HDF. Из состояния C можно перейти в B, HDF, из B можно перейти в HDF. Из HDF можно перейти в $C, BC, HBCDF$, из BC можно перейти только в HDF, из $HBCDF$ можно перейти только в само $HBCDF$. Таким образом, круг замкнулся, перечень состояний нашего автомата:

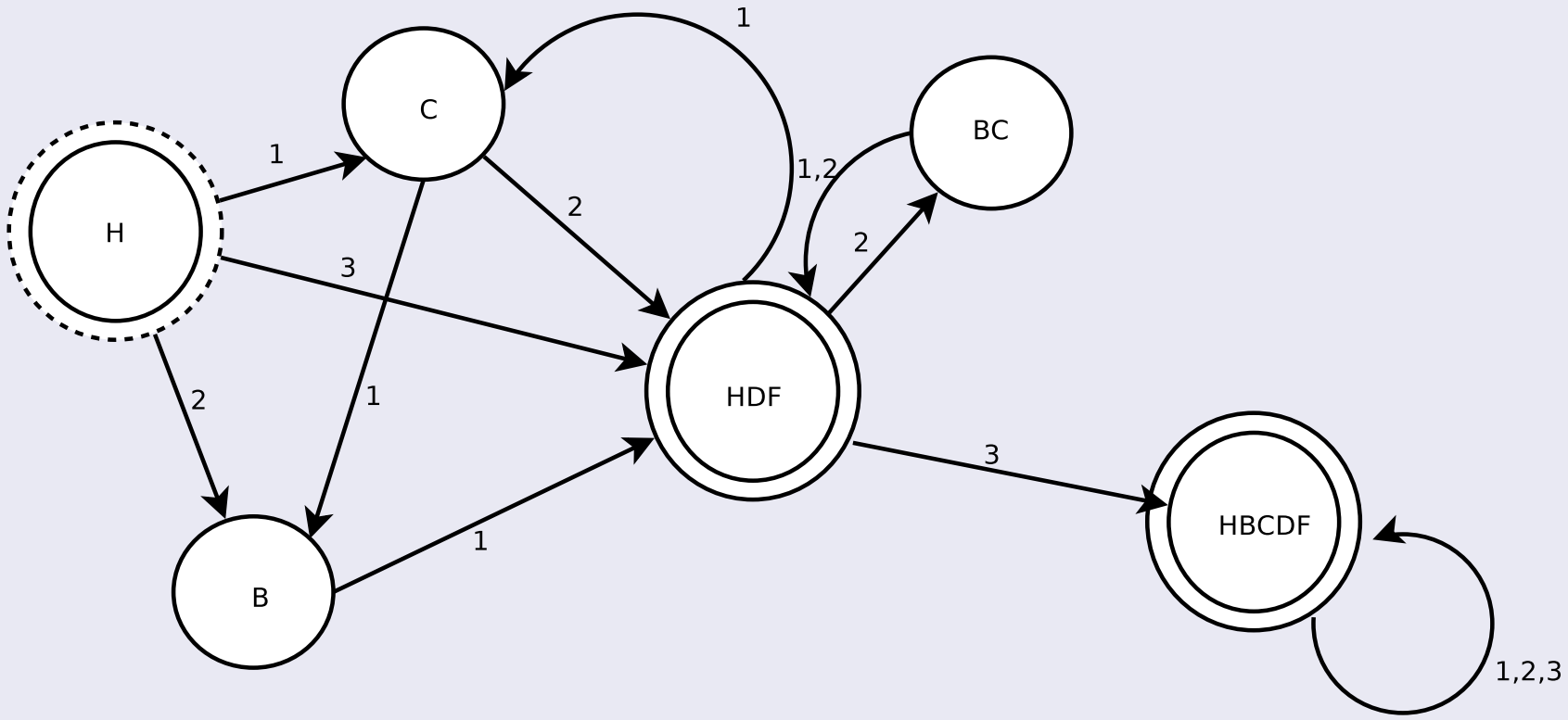
$$H, B, C, BC, HDF, HDCDF$$

Начальным состоянием является H, конечные состояния - это все, которые включают в себя символ F: $HDF, HDCDF$. Таким образом, имеем 2 конечных состояния. Функция переходов имеет следующий вид:

$$\delta'(H,1) = C, \delta'(H,2) = B, \delta'(H,3) = HDF, \delta'(C,1) = B, \delta'(C,2) = HDF,$$
$$\delta'(B,1) = HDF, \delta'(HDF,1) = C, \delta'(HDF,2) = BC, \delta'(HDF,3) = HBCDF,$$
$$\delta'(BC,1) = HDF, \delta'(BC,2) = HDF, \delta'(HBCDF,1) = HBCDF,$$
$$\delta'(HBCDF,2) = HBCDF, \delta'(HBCDF,3) = HBCDF$$

Пример преобразования недетерминированного конечного автомата к детерминированному виду

В соответствии с функцией переходов, автомат примет следующий вид:



В соответствии с этой схемой можно попробовать распознать правильные цепочки языка $222312, 1312$ автомата по продаже газировки, анализируя их справа налево.

Задание

1. Построить по заданной регулярной грамматике детерминированный конечный автомат:

   G=({X, Y, Z, W, V}, {0, 1, ~, #, &}, P, X), где P:

   X → 0Y | 1Z | 1 Y → 0Z | ~W | #

   Z→ 1Y | 1W | 0V W→0W | 1W | #

   V → &Z

2. Построить по заданной регулярной грамматике детерминированный конечный автомат:

   G=({K, L, M, N, P}, {0, 1, &, %, a, b}, C, K), где C:

   K→1M | 1 M→0L | &N | &P

   L→1L | 0L | %P N→aN | bN | %P

   P→1P | aP | 0

Минимизация состояний конечного автомата

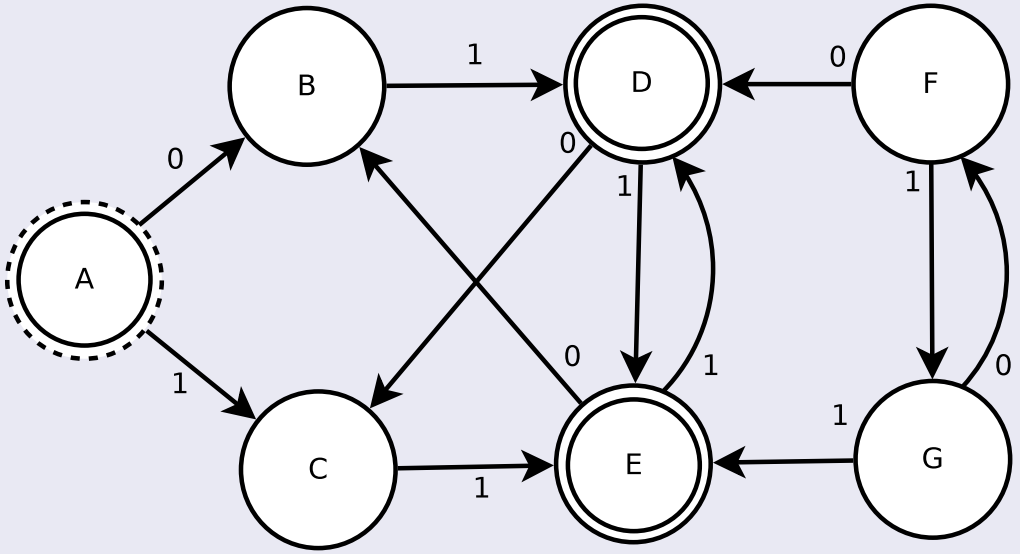
В ряде случаев можно уменьшить количество состояний исходного автомата и получить эквивалентный автомат. Такой автомат фактически будет распознавателем того же языка, но оказывается проще исходного.
Для минимизации количества состояний используется алгоритм выделения эквивалентных состояний.
Два состояния $q_1, q_2 \in Q$ называются n-эквивалентными, если из q_1, q_2 под действием входной цепочки символов длины не более n автомат может перейти в одно и тоже множество конечных состояний.
Когда входная цепочка не подается, уже существует два 0-эквивалентных множества состояний: $F, Q \setminus F$.
Множества эквивалентных состояний автомата называются **классами эквивалентности**, а всю их совокупность называют множеством классов эквивалентности, будем его обозначать через $R(n)$.
Например, $R(0) = \{F, Q \setminus F\}$
Рассмотрим алгоримт нахождения классов эквивалентных состояний.

Алгоритм минимизация состояний конечного автомата

1. Удаляем из автомата все недостижимые состояния. Принимаем $n = 0$ и формируем $R(0) = \{F, Q \setminus F\}$.
2. Присваиваем $n = n + 1$, формируем классы эквивалентности: в один класс эквивалентности на n шаге попадают те состояния, которые по всем одинаковым входным символам переходят в одинаковые $n-1$-эквивалентные состояния.
3. Если $R(n) = R(n-1)$, то алгоритм закончен, иначе повторить пункт 2.

Доказано, что данный алгоритм завершается не позже за $n = |Q| - 2$ шагов, где $|Q|$ - суммарное количество состояний исходного автомата.
Рассмотрим пример использования данного алгоритма.

Пример использования алгоритма минимизация состояний конечного автомата

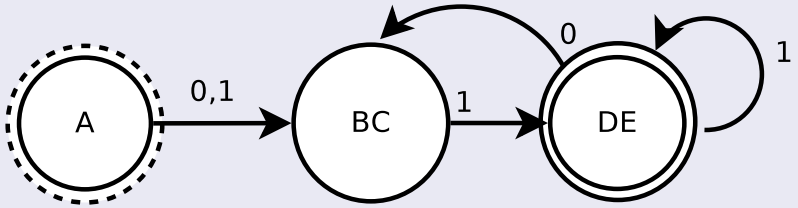


Для данного автомата состояния F, G являются недостижимыми, поэтому их можно исключить из рассмотрения.

$$n = 0, R(0) = \{\{A, B, C\}, \{D, E\}\}$$
$$n = 1, \delta(A, 0) = R(0)_1, \delta(A, 1) = R(0)_1, \delta(B, 0) = \varnothing, \delta(B, 1) = R(0)_2,$$
$$\delta(C, 0) = \varnothing, \delta(C, 1) = R(0)_2$$
$$\delta(D, 0) = R(0)_1, \delta(D, 1) = R(0)_2, \delta(E, 0) = R(0)_1, \delta(E, 1) = R(0)_2$$
$$R(1) = \{\{A\}, \{B, C\}, \{D, E\}\} = R(2)$$

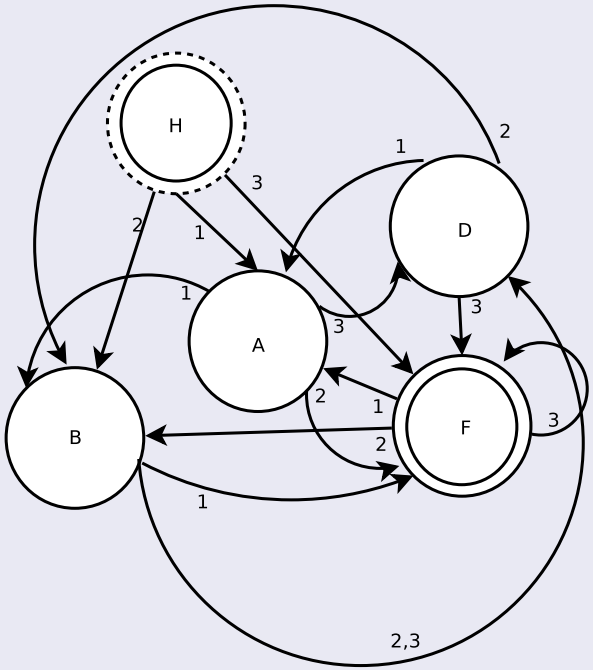
Пример использования алгоритма минимизация состояний конечного автомата

Для получения минимизированного конечного автомата объединим состояния D, E, и состояния B, C. Построим для них функцию переходов: $\delta(BC, 1) = DE, \delta(BC, 0) = \varnothing, \delta(DE, 0) = BC, \delta(DE, 1) = DE$. Окончательно минимизированный автомат без недостижимых состояний примет следующий вид:



Ясно, что получение автомата с меньшим количеством состояний позволяет строить наиболее простые распознаватели.
Рассмотрим еще пример минимизации состояний автомата по продаже газировки.

Минимизация состояний автомата по продаже газировки



$$n = 0, R(0) = \{\{H, A, B, D\}, \{F\}\}$$

$$n = 1, \delta(H, 1) = R(0)_1, \delta(H, 2) = R(0)_1, \delta(H, 3) = R(0)_2$$

$$\delta(D, 1) = R(0)_1, \delta(D, 2) = R(0)_1, \delta(D, 3) = R(0)_2$$

$$\delta(F, 1) = R(0)_1, \delta(F, 2) = R(0)_1, \delta(F, 3) = R(0)_2$$

$$\delta(A, 1) = R(0)_1, \delta(A, 2) = R(0)_2, \delta(A, 3) = R(0)_1$$

$$\delta(B, 1) = R(0)_2, \delta(B, 2) = R(0)_1, \delta(B, 3) = R(0)_1$$

$$R(1) = \{\{H, D\}, \{A\}, \{B\}, \{F\}\} = R(2)$$

Минимизация состояний автомата по продаже газировки

Окончательно получаем: