§1. Дискретная машина

Определение 1.1. *Дискретная машина* (или *цифровая машина*) – это устройство, для которого справедливо следующее:

1. устройство обладает входом, на который могут подаваться входные сигналы, принадлежащие некоторому конечному множеству X;

2. устройство обладает выходом, на котором получаются выходные сигналы, принадлежащие некоторому конечному множеству Y;

3. входные сигналы поступают в устройство, а выходные выходят из устройства в дискретные моменты времени, которые можно пронумеровать целыми числами: $t = 0, 1, 2, \ldots$;

4. сигнал, образующийся на выходе устройства в момент времени t, определяется исключительно только входными сигналами, полученными устройством до момента времени t.

Машины, не являющиеся дискретными, называются *непрерывными* (или *аналоговыми*).

В реальных дискретных машинах входные сигналы могут поступать не в каждый момент времени. То же самое справедливо и для выходных сигналов. Однако отсутствие сигнала можно само по себе считать сигналом, поэтому для таких машин добавим в множество X и/или Y специальный сигнал λ, обозначающий отсутствие сигнала. Это даст нам возможность считать, что входной и выходной сигналы присутствуют в каждый момент времени.

Мы будем обозначать входной сигнал, поступивший в машину в момент времени t как $s\,(t)$, а выходной сигнал, образовавшийся на выходе машины в момент времени t как $r\,(t)$.

Определение 1.2. *Предыстория* момента времени t – это последовательность входных сигналов, поступившая на вход дискретной машины к моменту времени t:
$h(t) = \langle s(0), s(1), s(2), \ldots, s(t-1) \rangle$.
Для общности предысторию момента времени 0 будем считать пустой последовательностью $\langle\rangle$.

Имея понятие предыстории, работу дискретной машины можно описать некоторой функцией F такой, что $r(t+1) = F(h(t), s(t))$.

Пример. (Машина с задержкой на два такта)
Пусть $X = \{0, 1\}$, $Y = \{\lambda, 0, 1\}$,

$$F(h, x) = \begin{cases} \lambda, & \text{если } h = \langle\rangle; \\ x', & \text{если } h = \langle \ldots x' \rangle. \end{cases}$$

Тогда при подаче на вход машины последовательности
$1, 0, 0, 1, 1, 1, 0, \ldots$
на выходе образуется последовательность
$\lambda, \lambda, 1, 0, 0, 1, 1, \ldots$.

Мы будем обозначать множество всех предысторий момента времени t как $H(t)$. Понятно, что $|H(t)| = n^t$, где $n = |X|$.

Множество всех предысторий ДО момента времени t будем записывать как $H^\star(t) = H(0) \cup H(1) \cup H(2) \cup \ldots \cup H(t)$.

Нетрудно доказать, что $|H^\star(t)| = \sum\limits_{i=0}^{t} n^i = \begin{cases} \dfrac{n^{t+1}-1}{n-1}, & \text{если } n > 1; \\ n+1, & \text{если } n = 1. \end{cases}$

Пример. Пусть $n = |X| = 10$, тогда $|H^\star(3)| = \dfrac{10^4-1}{10-1} = 1111$.

Действительно, $10^0 + 10^1 + 10^2 + 10^3 = 1111$.

Определение 1.3. Пусть M_1 и M_2 – две идентичные копии одной и той же дискретной машины, а h_1 и h_2 – это предыстории моментов времени t_1 и t_2 машин M_1 и M_2, соответственно.

Будем подавать одну и ту же последовательность входных сигналов g на машину M_1, начиная с момета времени t_1, и на машину M_2, начиная с момента времени t_2.

Если для любой последовательности g выходные сигналы, выдаваемые обеими машинами, совпадают, то h_1 и h_2 – *эквивалентные предыстории*.

Пример. (Машина с задержкой на два такта)

Рассмотрим две предыстории машины с задержкой на два такта: $\langle 0, 0, 1, 1, 0\rangle$ и $\langle 1, 0, 0\rangle$.

Ясно, что они эквивалентны, так как работа машины определяется последним входным сигналом, который у этих предысторий одинаковый.

Нетрудно убедиться в рефлексивности, симметричности и транзитивности введённого отношения эквивалентности. Тогда оно должно разбивать множество всех предысторий данной дискретной машины на непересекающиеся классы эквивалентности.

Определение 1.4. Класс эквивалентности предысторий дискретной машины называется *состоянием*.

Пример. (Машина с задержкой на два такта)

Нетрудно убедиться, что множество предысторий машины с задержкой на два такта разбивается на три состояния:

$q_1 = \{\langle\rangle\}$ – единственная пустая предыстория;

$q_2 = \{\langle \ldots, 0\rangle\}$ – все предыстории, оканчивающиеся на 0;

$q_3 = \{\langle \ldots, 1\rangle\}$ – все предыстории, оканчивающиеся на 1.

Если в момент времени t машина имеет предысторию $h(t)$, и q – состояние, которому принадлежит $h(t)$, то говорят, что в момент времени t машина *находится в состоянии* q.

Если в момент времени t машина находится в состоянии q_1 и получает входной сигнал s, а в момент времени $t+1$ машина находится в состоянии q_2, то говорят, что машина *переходит из состояния* q_1 *в состояние* q_2 *по входному сигналу* s.

Утверждение 1.1. Если дискретная машина находилась в состоянии q, то состояние, в которое она перешла по сигналу s, не зависит от конкретной предыстории, в результате которой машина находилась в состоянии q.

$\triangleright$ Доказательство непосредственно следует из определения эквивалентных предысторий. $\triangleleft$

Пусть Q – множество состояний некоторой дискретной машины.

Из утверждения 1.1 следует, что дискретная машина описывается двумя функциями:
$\delta : Q \times X \to Q$ – функция переходов;
$\varphi : Q \times X \to Y$ – функция выходов.

С помощью этих функций можно записать рекуррентные соотношения, определяющие работу дискретной машины:
$q(t+1) = \delta(q(t), s(t))$,
$r(t+1) = \varphi(q(t), s(t))$.

Пример. (Машина с задержкой на два такта)

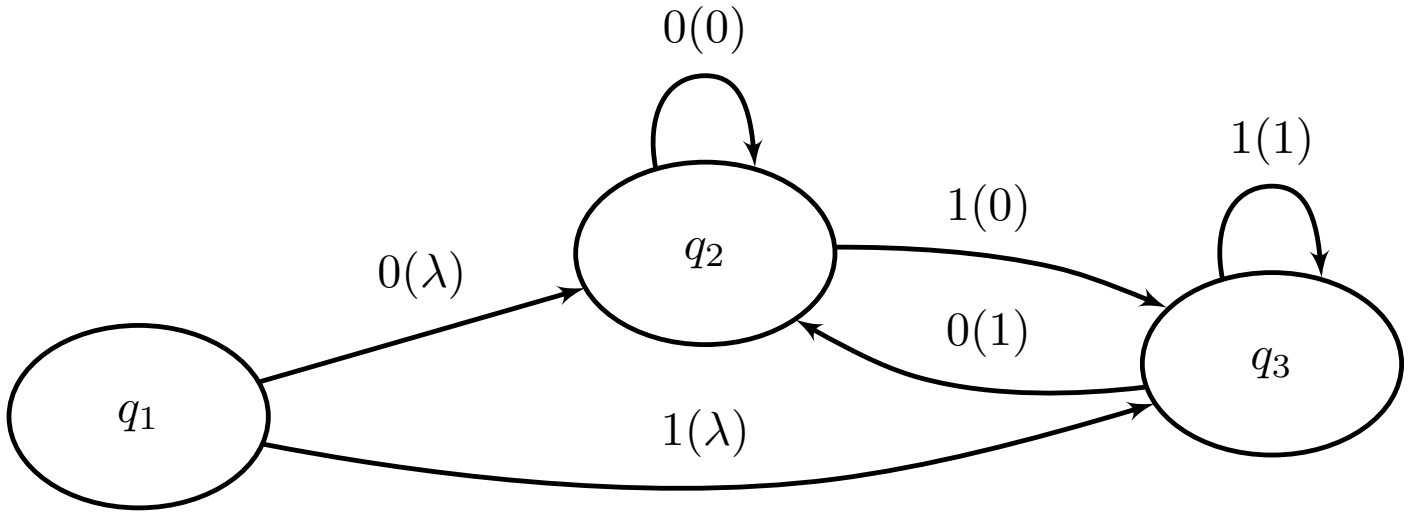
$X = \{0, 1\}$, $Y = \{\lambda, 0, 1\}$, $Q = \{q_1, q_2, q_3\}$.

$$\delta(q_1, 0) = q_2, \quad \delta(q_1, 1) = q_3, \quad \varphi(q_1, 0) = \lambda, \quad \varphi(q_1, 1) = \lambda,$$
$$\delta(q_2, 0) = q_2, \quad \delta(q_2, 1) = q_3, \quad \varphi(q_2, 0) = 0, \quad \varphi(q_2, 1) = 0,$$
$$\delta(q_3, 0) = q_2, \quad \delta(q_3, 1) = q_3, \quad \varphi(q_3, 0) = 1, \quad \varphi(q_3, 1) = 1.$$

Функции переходов и выходов дискретной машины с конечным множеством состояний удобно изображать в виде *диаграммы* – размеченного ориентированного мультиграфа, вершинами которого являются состояния, а дуги обозначают переходы между состояниями.

При этом каждая дуга помечается входным сигналом, вызывающим переход по этой дуге, и выходным сигналом, который окажется на выходе машины, когда она перейдёт в следующее состояние.

Пример. (Машина с задержкой на два такта)

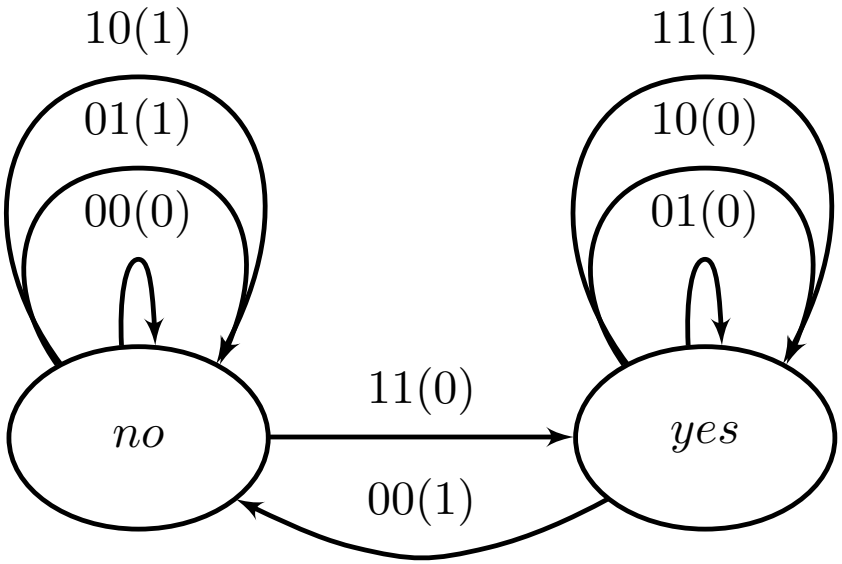


(Выходные сигналы записаны в скобках.)

Пример. (Двоичный последовательный сумматор)

На вход сумматора подаётся сразу две последовательности двоичных цифр (в low-endian). На выходе сумматора получается последовательность двоичных цифр, представляющая собой сумму двух чисел, подаваемых на вход.

$X = \{00, 01, 10, 11\}$, $Y = \{0, 1\}$, $Q = \{yes, no\}$.



Состояние yes содержит все предыстории, в результате которых получился «перенос», состояние no – все остальные предыстории.

§2. Абстрактный автомат как модель дискретной машины

Определение 2.1. *Абстрактный автомат* – это пятёрка $A = \langle Q, X, Y, \delta, \varphi \rangle$, в которой:

Q – множество состояний;

X и Y – конечные входной и выходной алфавиты;

$\delta : Q \times X \to Q$ – функция переходов;

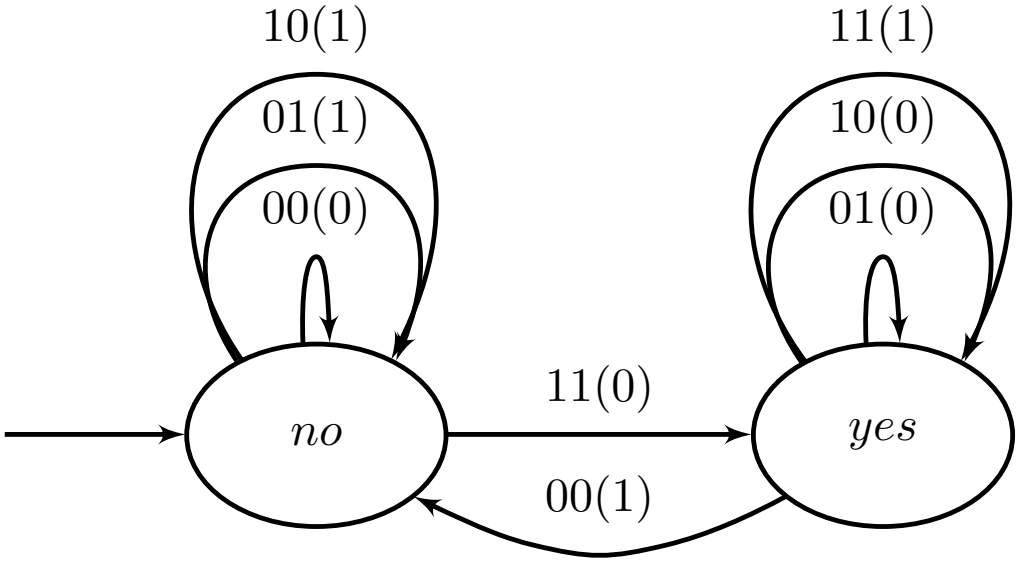
$\varphi : Q \times X \to Y$ – функция выходов.

Если дискретная машина в общем случае представляет собой реальное устройство (механизм, электронная схема) сложной конструкции, то абстрактный автомат – это математическая модель этого устройства, позволяющая абстрагироваться от деталей его конструкции. То есть абстрактный автомат описывает наблюдаемое поведение дискретной машины, но не несёт никакой информации о том, как это поведение реализовано.

Определение 2.2. *Инициальный абстрактный автомат* – это абстрактный автомат с выделенным начальным состоянием q_0, в котором он обязан находиться до приёма первого входного сигнала.

В дальнейшем нам, как правило, придётся работать именно с инициальными автоматами, поэтому условимся на диаграмме автомата изображать начальное состояние входящей стрелкой «из ниоткуда».

Пример. (Инициальный двоичный последовательный сумматор)



Заметим, что по определению дискретной машины сигнал $r(t)$ зависит только от предыстории $h(t)$. Это значит, что в общем случае машина должна запоминать где-то у себя внутри по крайней мере ту часть входных сигналов, которая определяет её дальнейшую работу.

Состояние абстрактного автомата моделирует «слепок» памяти дискретной машины.

Лемма 2.1. Если количество состояний некоторой дискретной машины бесконечно, то для любого времени t_0 найдётся такое время $t \geq t_0$, что существует предыстория $h(t)$, не эквивалентная ни одной предыстории из $H^\star(t_0)$.

$\triangleright$ Предположим, что существует время t_0 такое, что для любого $t \geq t_0$ любая предыстория $h(t)$ эквивалентна какой-то предыстории из $H^\star(t_0)$.

Тогда, даже если все предыстории в $H^\star(t_0)$ попарно неэквивалентны, количество состояний не может превышать $|H^\star(t_0)|$, то есть оно конечно. $\triangleleft$

Утверждение 2.1. Количество состояний дискретной машины с конечной памятью конечно.

$\triangleright$ Предположим, что количество состояний некоторой дискретной машины с конечной памятью бесконечно.

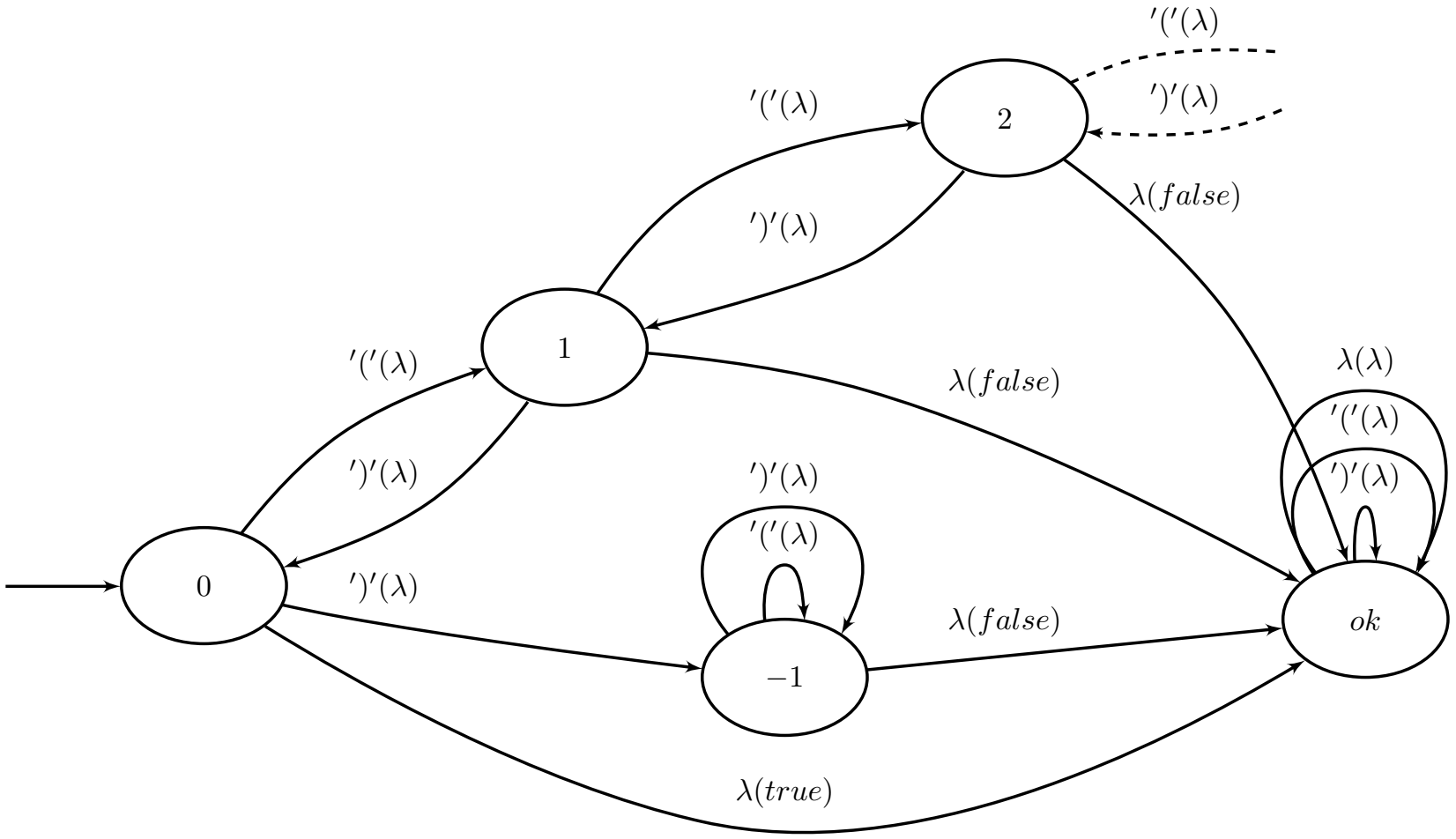
Тогда, согласно лемме 2.1, для любого сколь угодно большого времени t_0 найдётся такое время $t \geq t_0$, что существует предыстория $h(t)$, не эквивалентная ни одной предыстории из $H^\star(t_0)$.

Это означает, что машина запомнила достаточное количество входных сигналов из $h(t)$, чтобы предыстория $h(t)$ отличалась от любой другой предыстории из $H^\star(t_0)$.

Очевидно, что чем больше t_0, тем больше входных сигналов должна запомнить машина. Однако память машины конечна, и при каком-то t_0 она будет исчерпана. $\triangleleft$

На практике автоматы с бесконечным множеством состояний напрямую не применяются, но являются базой для теоретических постороений.

Пример. (Автомат, проверяющий сбалансированность скобок)
$X = \{\lambda, '(', ')'\}$, $Y = \{\lambda, true, false\}$, $Q = \{ok, -1, 0, 1, 2, \ldots\}$.



Определение 2.3. *Автомат Мили* – это абстрактный автомат с конечным множеством состояний.

В заключение параграфа необходимо акцентировать внимание на одном важном свойстве абстрактного автомата.

Замечание. Так как δ и φ в определении абстрактного автомата – функции, и они определены для любого элемента множества $Q \times X$, то из любого состояния абстрактного автомата существует переход по любому входному сигналу. Поэтому говорят, что автомат – *детерминированный*.

§3. Представление абстрактного автомата в памяти компьютера

Абстрактный автомат $A = \langle Q, X, Y, \delta, \varphi \rangle$ с конечным множеством состояний (автомат Мили) можно представить в программе в виде двух матриц: матрицы переходов Δ и матрицы выходов Φ.

Строки каждой из этих матриц соответствуют пронумерованным состояниям автомата, а столбцы – пронумерованным входным сигналам. В ячейке $\triangle_{ij}$ матрицы переходов находится номер состояния $\delta\left(q_i, x_j\right)$, в ячейке Φ_{ij} матрицы выходов находится номер выходного сигнала $\varphi\left(q_i, x_j\right)$.

Работа дискретной машины, моделируемой автоматом, в этом представлении реализуется следующим алгоритмом:

```
1 RunMealy(in Δ, in Φ, in q)
2       loop:
3             x ←очередной сигнал на входе
4             Отправить Φ[q, x] на выход
5             q ← Δ[q, x]
```

Определение 3.1. Входные сигналы x_1 и x_2 для автомата $A = \langle Q, X, Y, \delta, \varphi \rangle$ являются *эквивалентными*, если для любого состояния $q \in Q$ справедливы утверждения:
$\delta(q, x_1) = \delta(q, x_2)$,
$\varphi(q, x_1) = \varphi(q, x_2)$.

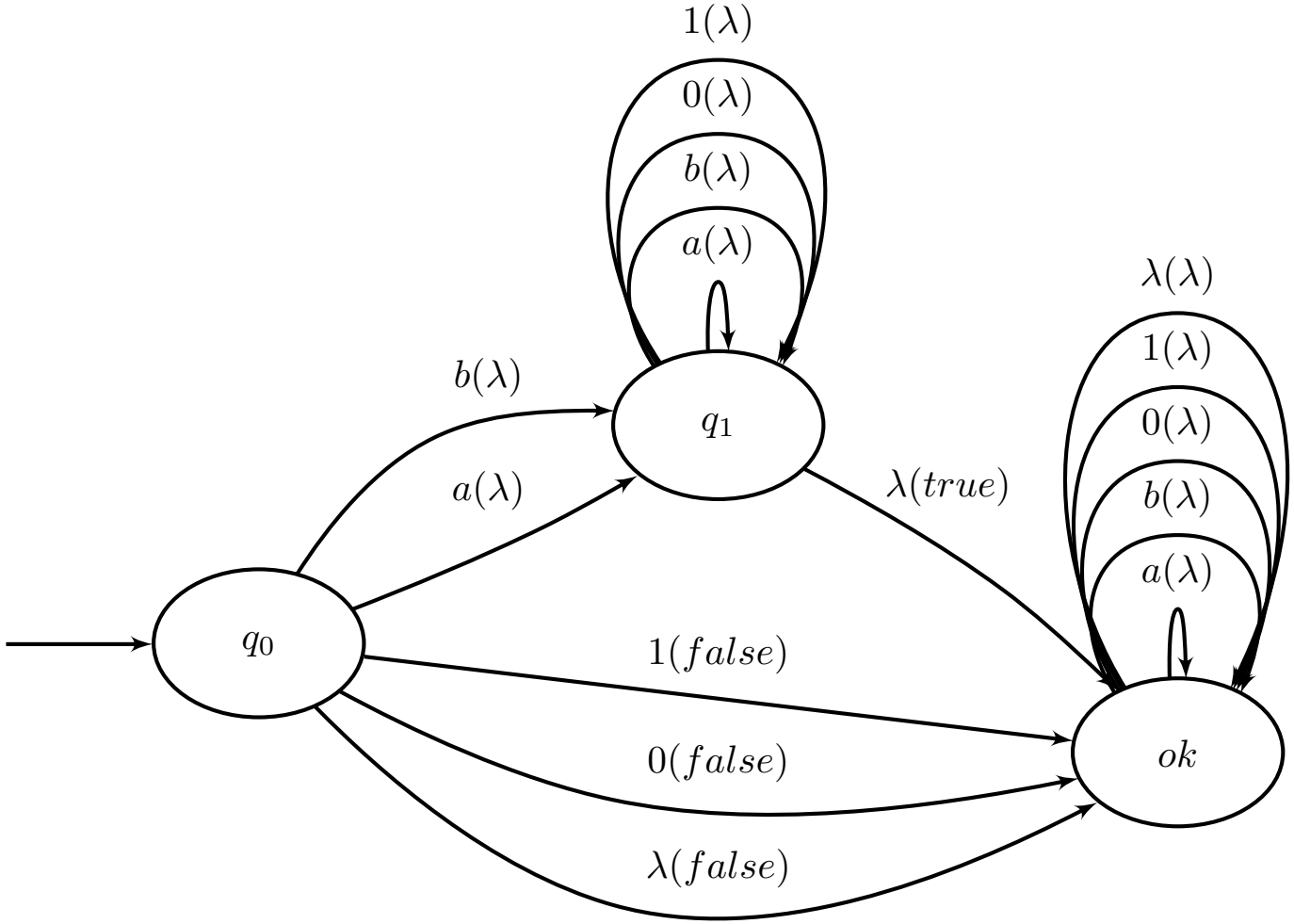
Столбцы матрицы переходов для эквивалентных сигналов совпадают. То же самое справедливо и для столбцов матрицы выходов.

Определение 3.2. *Обобщённый сигнал* – это класс эквивалентности входных сигналов.

Пример. (Автомат, распознающий идентификаторы)

Идентификатор – непустая последовательность букв и цифр, начинающаяся с буквы.

$X = \{\lambda, a, b, 0, 1\}$, $Y = \{\lambda, true, false\}$, $Q = \{ok, q_0, q_1\}$.



Сигналы a и b эквивалентны. Сигналы 0 и 1 эквивалентны.

В случае, если обобщённых сигналов заметно меньше, чем входных сигналов, имеет смысл поставить в соответствие столбцам матриц переходов и выходов номера обобщённых сигналов.

Тогда для перевода входных сигналов в обобщённые нам потребуется массив γ такой, что $\gamma[x] = \overline{x}$, где x – номер входного сигнала, а $\overline{x}$ – номер обобщённого сигнала, которому этот входной сигнал принадлежит.

Работа дискретной машины, моделируемой автоматом, представленном матрицами переходов и выходов с обобщёнными сигналами, реализуется следующим алгоритмом:

```
1 RunMealy2(in γ, in Δ, in Φ, in q)
2       loop:
3             x ←очередной сигнал на входе
4             x̄ ← γ[x]
5             Отправить Φ[q, x̄] на выход
6             q ← Δ[q, x̄]
```

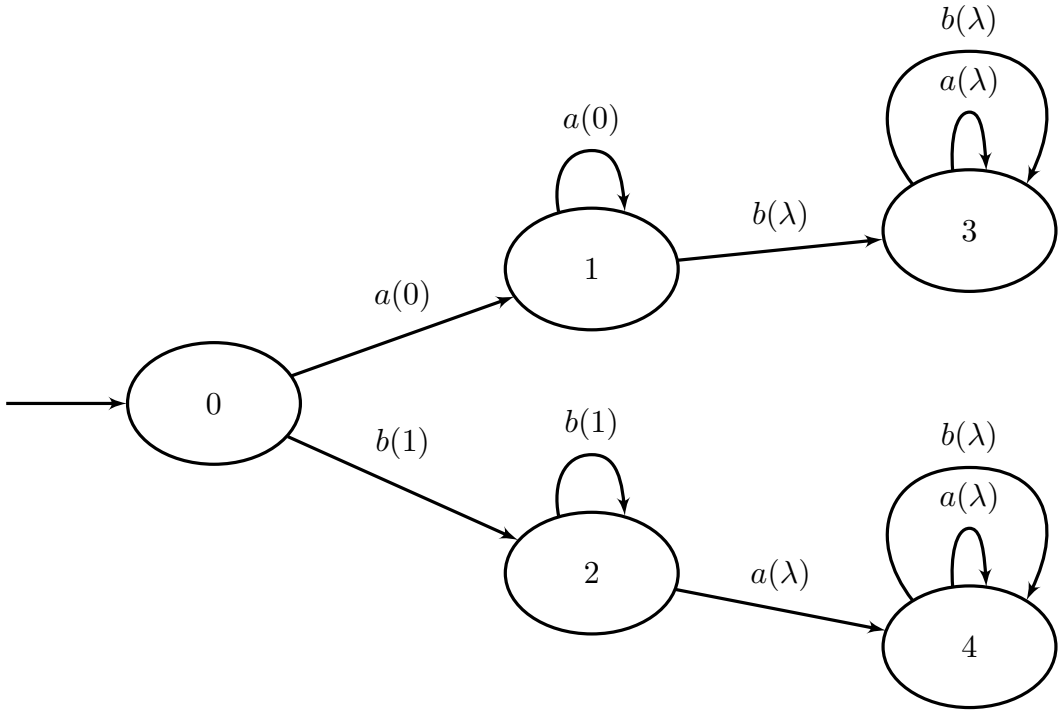
§4. Минимизация автомата Мили

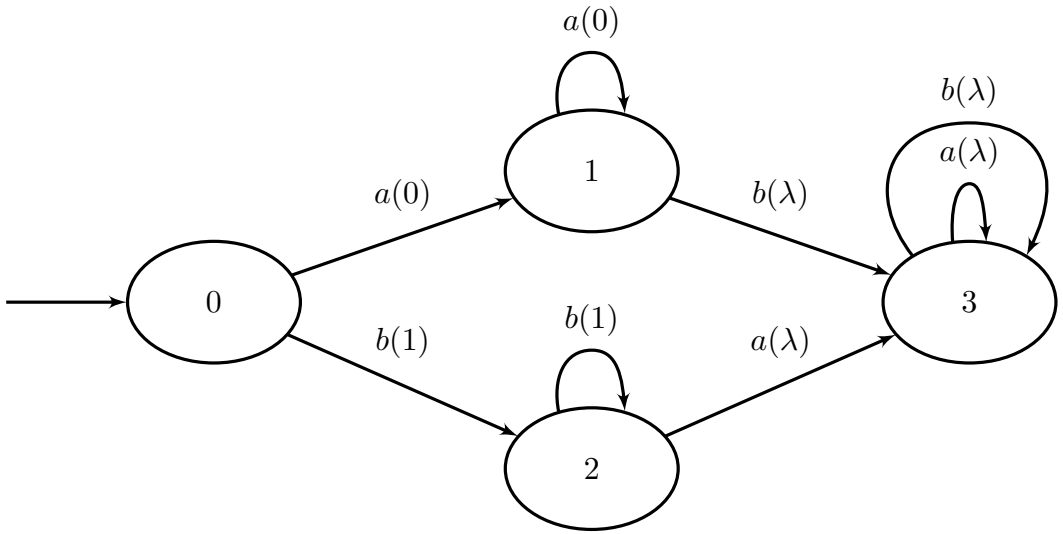
Так как состояние дискретной машины является классом эквивалентности её предысторий, то, очевидно, уменьшить количество её состояний невозможно (разбиение множества на классы эквивалентности единственно, поэтому для уменьшения количества состояний пришлось бы объединять классы эквивалентности).

Однако, применительно к абстрактному автомату это утверждение неверно, хотя он и является моделью дискретной машины.

Определение 4.1. Инициальные абстрактные автоматы A_1 и A_2 с совпадающими алфавитами ($X_1 = X_2$ и $Y_1 = Y_2$) называются *эквивалентными*, если они выдают одинаковые последовательности выходных сигналов для любой предыстории.

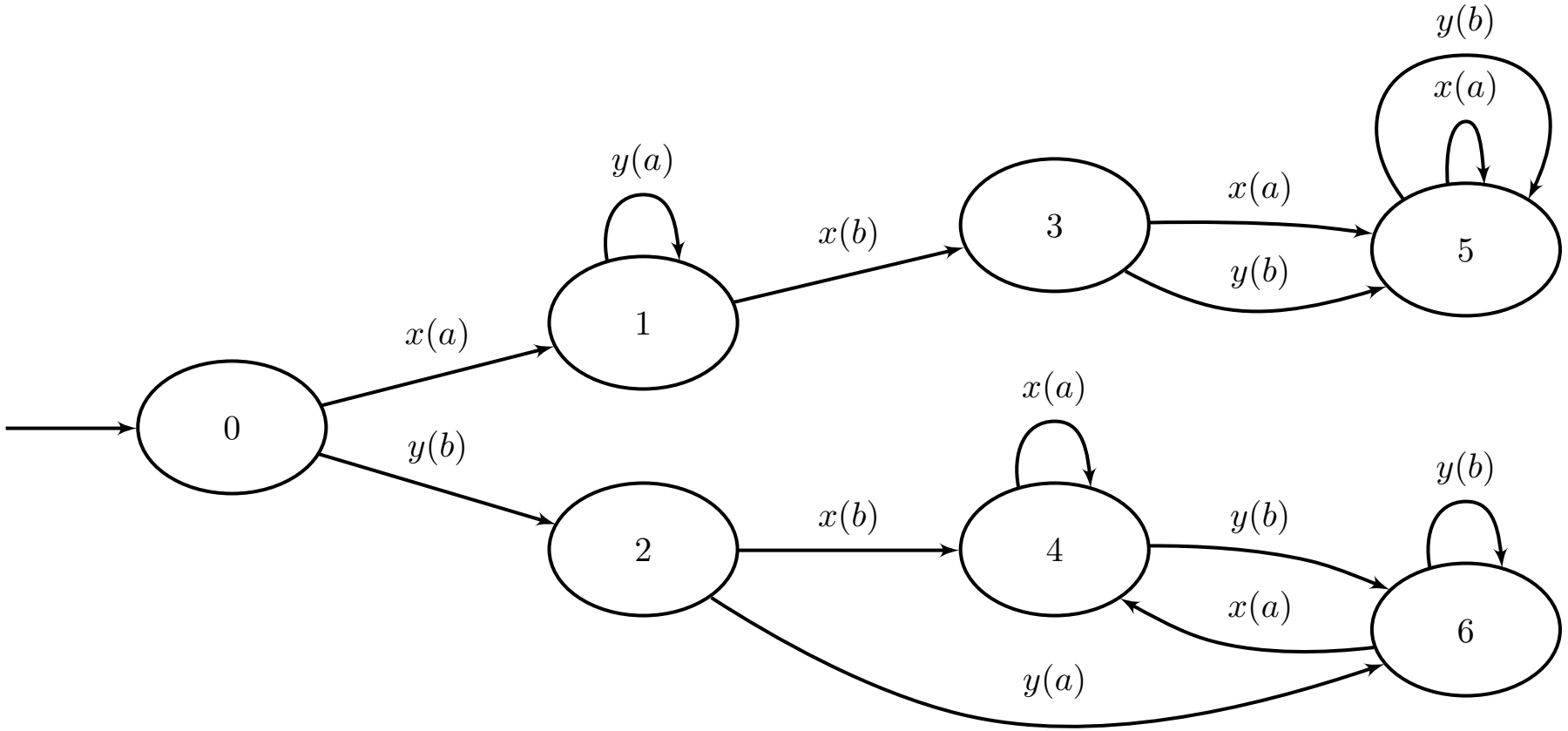
Пример. (Два эквивалентных автомата Мили)





Определение 4.2. Состояния q_1 и q_2 автомата A *k-эквивалентны* (записывается: $q_1 \overset{k}{\sim} q_2$), если для любой последовательности входных сигналов длины k автомат A, находящийся в состоянии q_1, выдаёт ту же самую последовательность выходных сигналов, что и автомат A, находящийся в состоянии q_2.

Пример. ($0 \overset{1}{\sim} 3 \overset{1}{\sim} 4 \overset{1}{\sim} 5 \overset{1}{\sim} 6$, $1 \overset{1}{\sim} 2$, $3 \overset{2}{\sim} 4 \overset{2}{\sim} 5 \overset{2}{\sim} 6$, ...)



Из определения следует, что $q_1 \overset{1}{\sim} q_2$, если $\varphi(q_1, x) = \varphi(q_2, x)$ для любого входного сигнала x.

Основная идея алгоритма Split1, разбивающнго множество состояний автомата Мили на классы 1-эквивалентности, состоит в следующем:
1. пусть сначала каждый «класс» состоит только из одного состояния;
2. попарно рассматриваем все состояния, и если для пары состояний q_1 и q_2 переходы по любому входному сигналу дают одинаковый выходной сигнал, объединяем «классы» состояний q_1 и q_2 в один «класс».

Для представления «классов» будем использовать лес непересекающихся множеств. При этом каждое состояние будет представлять собой структуру из трёх полей: номер состояния $i \in \mathbb{N}_n$, указатель на родителя $parent$ и глубина поддерева $depth$.

Алгоритм Split1 будет порождать массив π, который каждому состоянию из Q ставит в соответствие корень класса, которому оно принадлежит: $\pi[q.i] = \mathsf{Find}(q)$.

Кроме того, алгоритм будет возвращать общее количество классов m.

```
1  Split1(in A = ⟨Q, X, Y, δ, φ⟩, out m, out π)
2      m ← |Q|
3      for each q ∈ Q:
4          q.parent ← q
5          q.depth ← 0
6      for each {q1, q2} ⊆ Q:
7          if Find(q1) ≠ Find(q2):
8              eq ← true
9              for each x ∈ X:
10                 if φ(q1, x) ≠ φ(q2, x):
11                     eq ← false
12                     break
13             if eq:
14                 Union(q1, q2)
15                 m ← m − 1
16     for each q ∈ Q:
17         π[q.i] ← Find(q)
```

Сложность алгоритма: $O\left(n^2\left(k+\alpha\left(n\right)\right)\right)$, где $n = |Q|$, $k = |X|$.

Утверждение 4.1. Если $q_1 \overset{1}{\sim} q_2$, и для любого $x \in X$ справедливо утверждение $\delta(q_1, x) \overset{k}{\sim} \delta(q_2, x)$, то $q_1 \overset{k+1}{\sim} q_2$.

▷ Возьмём произвольную последовательность входных сигналов $\langle x_1, x_2, \ldots, x_{k+1} \rangle$ и подадим её на вход автомату из состояний q_1 и q_2.

Так как $q_1 \overset{1}{\sim} q_2$, то $\varphi(q_1, x_1) = \varphi(q_2, x_1)$. Значит первый выходной сигнал будет одинаковый вне зависимости от того, в каком состоянии находится автомат.

Перейдя по входному сигналу x_1, мы окажемся в состоянии $w_1 = \delta(q_1, x)$ или в состоянии $w_2 = \delta(q_2, x)$. Так как $w_1 \overset{k}{\sim} w_2$, то, подав на вход автомата остальные k входных сигналов $\langle x_2, \ldots, x_{k+1} \rangle$, мы получим одну и ту же последовательность выходных сигналов.
Поэтому $q_1 \overset{k+1}{\sim} q_2$. ◁

Утверждение 4.2. Если $q_1 \overset{k}{\sim} q_2$, то $q_1 \overset{k-1}{\sim} q_2$.

▷ Доказательство тривиально. ◁

Алгоритм Split, который мы рассмотрим на следующем слайде, получает разбиение π множества состояний автомата на классы k-эквивалентности, и меняет разбиение π таким образом, чтобы оно представляло классы $(k+1)$-эквивалентности.

Работа алгоритма основана на утверждениях 4.1 и 4.2 и состоит в следующем:
1. пусть сначала каждый «класс» состоит только из одного состояния;
2. попарно рассматриваем все состояния, и если два состояния q_1 и q_2 принадлежат одному классу k-эквивалентности, и кроме того, нет такого входного сигнала, по которому из q_1 и q_2 можно было бы перейти в состояния из разных классов k-эквивалентности, то объединяем «классы» состояний q_1 и q_2 в один «класс».

```
1  Split(in A = ⟨Q, X, Y, δ, φ⟩, out m, in/out π)
2      m ← |Q|
3      for each q ∈ Q:
4          q.parent ← q
5          q.depth ← 0
6      for each {q1, q2} ⊆ Q:
7          if π[q1.i] = π[q2.i] and Find(q1) ≠ Find(q2):
8              eq ← true
9              for each x ∈ X:
10                 w1 ← δ(q1, x),   w2 ← δ(q2, x)
11                 if π[w1] ≠ π[w2]:
12                     eq ← false
13                     break
14             if eq:
15                 Union(q1, q2)
16                 m ← m − 1
17     for each q ∈ Q:
18         π[q.i] ← Find(q)
```

Сложность алгоритма: $O\left(n^2\left(k+\alpha\left(n\right)\right)\right)$, где $n = |Q|$, $k = |X|$.

Утверждение 4.3. Классов k-эквивалентности состояний автомата не может быть меньше классов $(k-1)$-эквивалентности.

$\triangleright$ Предположим обратное.

Согласно утверждению 4.2 («Если $q_1 \overset{k}{\sim} q_2$, то $q_1 \overset{k-1}{\sim} q_2$») классы k-эквивалентности вкладываются в классы $(k-1)$-эквивалентности. Поэтому из нашего предположения следует, что существует такой класс $(k-1)$-эквивалентности C, что в него не вложен ни один класс k-эквивалентности.

Значит состояния из C не входят ни в один класс k-эквивалентности, что невозможно. $\triangleleft$

Определение 4.3. Состояния q_1 и q_2 автомата A *эквивалентны*, если они k-эквивалентны для сколь угодно большого k.

Из утверждения 4.3 и того факта, что классов k-эквивалентности не может быть больше, чем состояний автомата, следует, что найдётся такое k, начиная с которого классы k-эквивалентности совпадают с классами $(k+i)$-эквивалентности для любого положительного i.

Тем самым, последовательно запуская Split до неподвижной точки, мы найдём *классы эквивалентности* состояний автомата. Эта процедура называется *минимизацией автомата*.

В результате минимизации строится автомат, эквивалентный исходному, состояниями которого являются классы эквивалентности исходного автомата.

Алгоритм Ауфенкампа-Хона минимизации автомата Мили:

```
1  AufenkampHohn(in A = ⟨Q, X, Y, δ, φ⟩, out A′)
2      Split1(A, out m, out π)
3      loop:
4          Split(A, out m′, in/out π)
5          if m = m′:
6              break
7          m ← m′
8      Q′ ← ∅,   δ′ ← ∅,   φ′ ← ∅
9      for each q ∈ Q:
10         q′ ← π[q.i]
11         if q′ ∉ Q′:
12             Q′ ← Q′ ∪ {q′}
13             for each x ∈ X:
14                 δ′ ← [⟨q′, x⟩ → π[δ(q, x).i]] δ′
15                 φ′ ← [⟨q′, x⟩ → φ(q, x)] φ′
16     A′ ← ⟨Q′, X, Y, δ′, φ′⟩
```

Сложность алгоритма: $O\left(n^3\left(k+\alpha\left(n\right)\right)\right)$, где $n = |Q|$, $k = |X|$.

§5. Автомат Мура

Автомат Мура – это частный случай автомата Мили. В автомате Мура выходные сигналы являются атрибутами состояний, а не переходов между состояниями.

Определение 5.1. *Автомат Мура* – это пятёрка $A = \langle Q, X, Y, \delta, \psi \rangle$, в которой:
Q – конечное множество состояний;
X и Y – конечные входной и выходной алфавиты;
$\delta : Q \times X \to Q$ – функция переходов;
$\psi : Q \to Y$ – функция выходов.

Рекуррентные соотношения, определяющие работу автомата Мура, выглядят как
$q(t+1) = \delta(q(t), s(t))$,
$r(t+1) = \psi(q(t))$.

Напомним, что у абстрактного автомата последняя формула имеет вид
$r(t+1) = \varphi(q(t), s(t))$.

Представление автомата Мура в памяти компьютера отличается от представления автомата Мили тем, что матрица выходов заменяется на массив выходов.

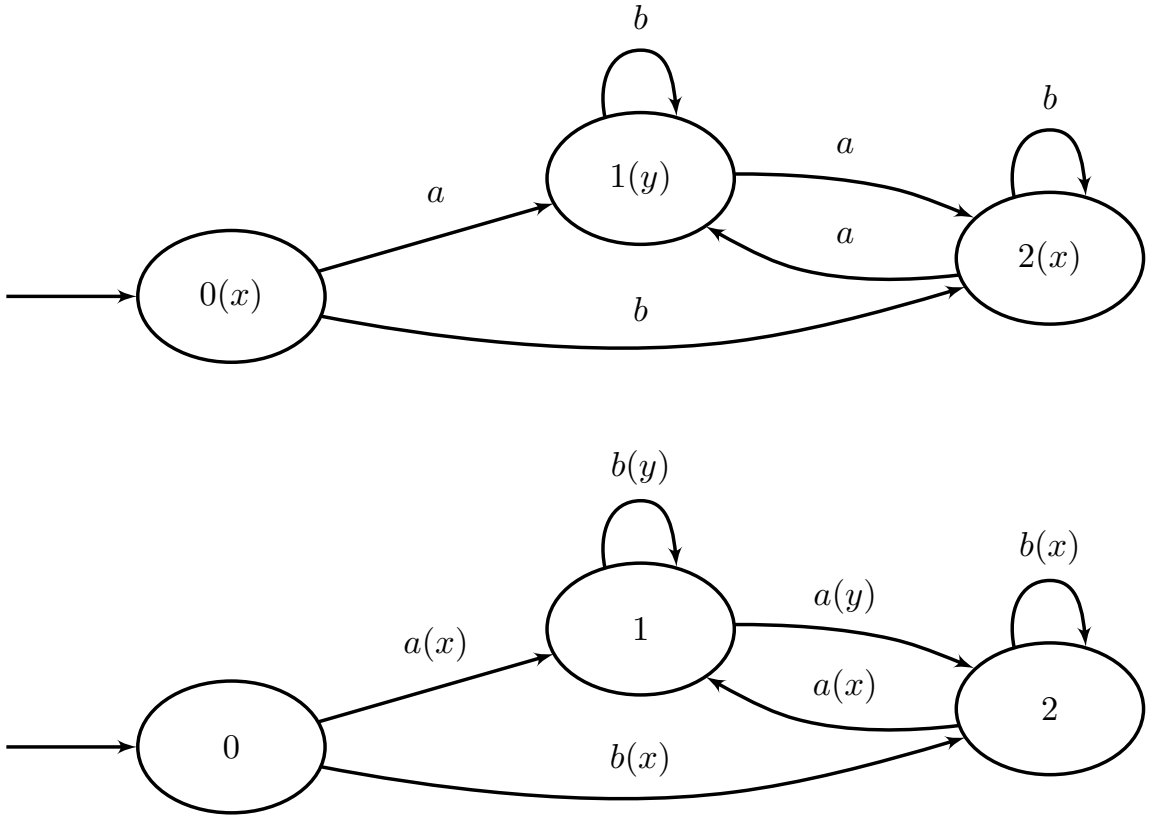
Элементы массива выходов Ψ соответствуют состояниям автомата Мура: Ψ_i – выходной сигнал автомата при проходе через i-тое состояние.

Работа дискретной машины, моделируемой автоматом Мура, реализуется следующим алгоритмом:

```
1 RunMoore(in Δ, in Ψ, in q)
2       loop:
3           x ←очередной сигнал на входе
4           Отправить Ψ[q] на выход
5           q ← Δ[q, x]
```

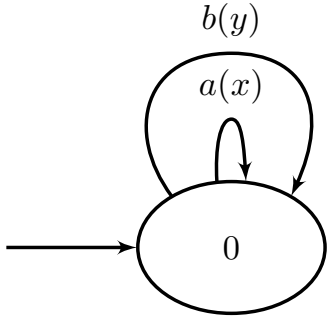
Если дан автомат Мура $A = \langle Q, X, Y, \delta, \psi \rangle$, то эквивалентный ему автомат Мили $A' = \langle Q', X, Y, \delta', \varphi \rangle$ строится следующим образом:
$Q' = Q$, $\delta' = \delta$,
и для любых $q \in Q, x \in X$: $\varphi(q, x) = \psi(q)$.
Нетрудно убедиться, что постороенный таким образом автомат Мили эквивалентен исходному автомату для любого начального состояния.

Пример. (автомат Мура $\Rightarrow$ автомат Мили)



В общем случае, когда в автомате Мили существует хотя бы одно состояние, при переходе из которого могут порождаться разные выходные сигналы, построить по этому автомату эквивалентный ему автомат Мура невозможно.

Проиллюстрируем это утверждение следующим автоматом Мили:



Предположим, что можно построить эквивалентный автомат Мура.

Пусть в момент времени 0 построенный нами автомат Мура находится в состоянии q_0. Так как функция выходов автомата не зависит от входного сигнала, то в момент времени 1 мы получим выходной сигнал $\psi(q_0)$, в то время как исходный автомат может выдавать разные сигналы (x или y). Значит наше предположение неверно.

Определение 5.2. Состояние q автомата Мили называется *преходящим*, если в него нельзя перейти ни из какого другого состояния.

Если инициальный автомат Мили $A = \langle Q, X, Y, \delta, \varphi, q_0 \rangle$ не имеет преходящих состояний, по нему можно построить инициальный автомат Мура $A' = \left\langle Q', X, Y, \delta', \psi, q'_0 \right\rangle$, который будет условно эквивалентен исходному: для одной последовательности входных сигналов $r_{A'}(t) = r_A(t-1)$ для любого $t \geq 2$ (задержка на один такт).

Состояния автомата Мура будут представлять собой пары $\langle q, y \rangle$, где $q \in Q$, $y \in Y$. При этом множество состояний с общим q мы будем называть *семейством* и определять как
$Z(q) = \{\langle q, y \rangle \mid \exists q_2 \in Q,\ \exists x \in X :\ \delta(q_2, x) = q$ и $\varphi(q_2, x) = y\}$.

То есть пара $\langle q, y \rangle$ входит в $Z(q)$, если существует переход в q с порождением выходного сигнала y.

Тогда $Q' = \bigcup\limits_{q \in Q} Z(q)$.

Функция переходов автомата Мура будет задаваться следующим образом:
$\delta'(\langle q, y\rangle, x) = \langle \delta(q, x), \varphi(q, x)\rangle$.

Функцию выходов автомата Мура определим как
$\psi(\langle q, y\rangle) = y$.

В качестве начального состояния q'_0 автомата Мура можно выбрать любое состояние вида $\langle q_0, y\rangle$.

§6. Распознающий автомат

Если рассмотренные ранее автоматы применяются главным образом для моделирования дискретных машин (схемотехника и т.п.), то в практике программирования наиболее полезны распознающие автоматы.

Определение 6.1. *Распознающий автомат* (или просто *конечный автомат*) – это пятёрка $A = \langle Q, X, \delta, F, q_0 \rangle$, в которой:
Q – конечное множество состояний;
X – конечный алфавит;
$\delta : Q \times X \to Q$ – функция переходов;
$F \subseteq Q$ – множество *принимающих* (или *заключительных*) состояний;
$q_0 \in Q$ – начальное состояние.

Элементы алфавита X называются *символами*. Говорят, что распознающий автомат *принимает* последовательность символов (строку), если в результате переходов по этим символам из начального состояния он оказывается в одном из принимающих состояний.

Представление распознающего автомата в памяти компьютера отличается от представления автомата Мили тем, что матрица выходов заменяется на булевский массив принимающих состояний.

Элементы массива принимающих состояний $Final$ соответствуют состояниям распознающего автомата: $Final_i$ = true, если i-тое состояние – принимающее, в противном случае $Final_i$ = false.

Алгоритм распознавания строки s возвращает true, если автомат принимает эту строку:

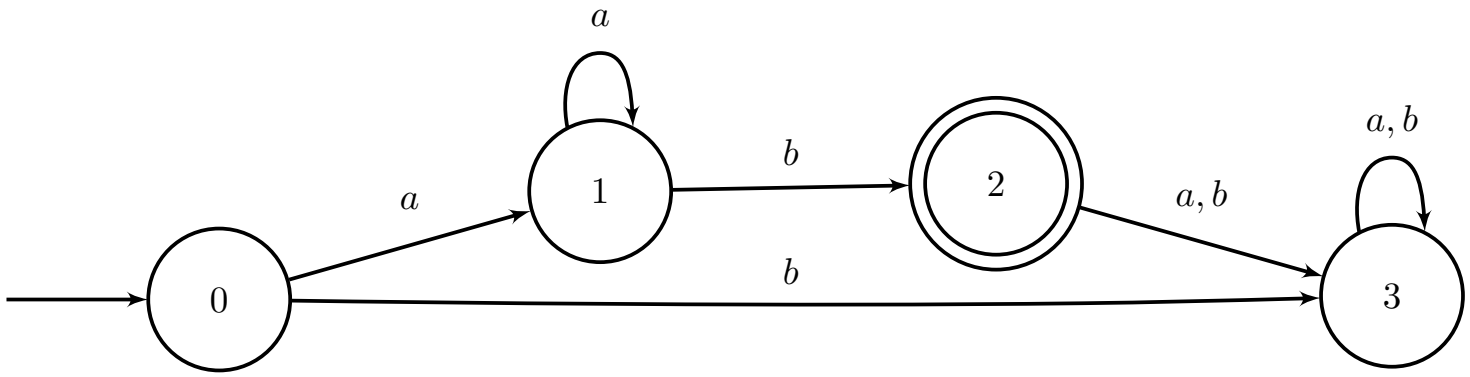
```
1 RunRecognizer(in Δ, in Final, in q, in s): accept
2       i ← 0
3       while i < len(S):
4             x ← s[i]
5             q ← Δ[q, x]
6             i ← i + 1
7       accept ← Final[q]
```

Определение 6.2. *Язык* распознающего автомата – это множество принимаемых им последовательностей символов.

Если считать, что распознающий автомат является моделью некоторой дискретной машины, то его состояния – классы эквивалентности предысторий этой дискретной машины.

Так как предыстория – не что иное, как последовательность символов, поступающая в распознающий автомат, то можно сказать, что язык автомата – это объединение его принимающих состояний.

Пример. (Язык распознающего автомата)

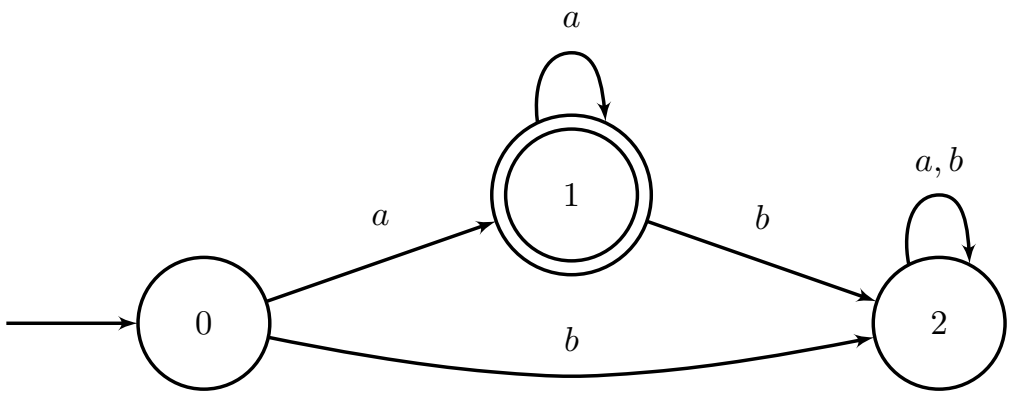


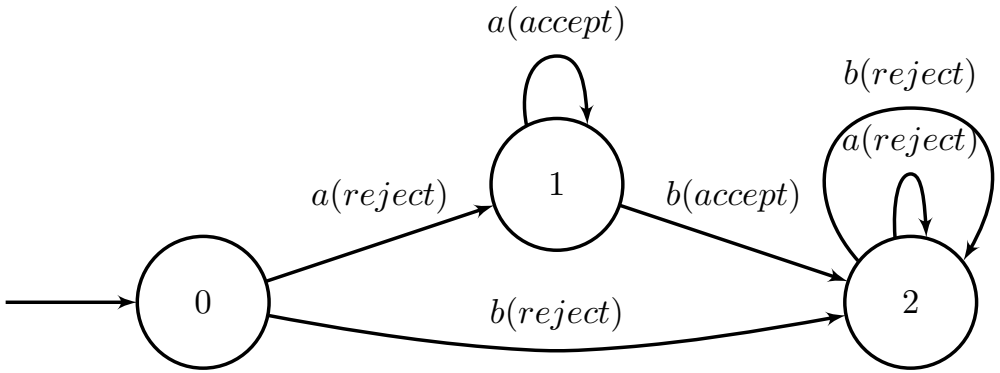

```
<0> ::= a <1>.
<1> ::= a <1> | b.
```

Если дан распознающий автомат $A = \langle Q, X, \delta, F, q_0 \rangle$, то эквивалентный ему (в некотором нестрогом смысле) инициальный автомат Мили $A' = \left\langle Q', X, Y, \delta', \varphi, q'_0 \right\rangle$ строится следующим образом:
$Q' = Q$, $Y = \{accept, reject\}$, $\delta' = \delta$, $q'_0 = q_0$,
и для любых $q \in Q, x \in X$:

$$\varphi(q, x) = \begin{cases} accept, & \text{если } q \in F, \\ reject, & \text{если } q \notin F. \end{cases}$$

Пример. (распознающий автомат $\Rightarrow$ автомат Мили)





Пусть q_1 и q_2 – состояния автомата Мили, эквивалентного некоторому распознающему автомату. Напомним, что $q_1 \overset{1}{\sim} q_2$, если $\varphi(q_1, x) = \varphi(q_2, x)$ для любого входного сигнала x.

Так как

$$\varphi(q, x) = \begin{cases} accept, & \text{если } q \in F, \\ reject, & \text{если } q \notin F, \end{cases}$$

то $q_1 \overset{1}{\sim} q_2$, если $\{q_1, q_2\} \subseteq F$ или $\{q_1, q_2\} \subseteq Q \setminus F$.

То есть состояния распознающего автомата делятся на два класса 1-эквивалентности: F и $Q \setminus F$.

Тогда алгоритм Split1 для распознающего автомата можно переписать следующим образом:

```
1 Split1_Recognizer(in A = ⟨Q, X, δ, F, q0⟩, out m, out π)
2       π ← Null_|Q|,   m ← 2
3       f, nf ← произвольные состояния из F и Q \ F
4       for each q ∈ Q:
5           if q ∈ F: π[q.i] ← f
6           else: π[q.i] ← nf
```

Алгоритм Split не зависит от функции выходов, поэтому он без изменений годится для распознающего автомата.

Запишем алгоритм Ауфенкампа-Хона для распознающего автомата:

```
1  AufenkampHohn_Recognizer(in A = ⟨Q, X, Y, δ, φ⟩, out A′)
2      Split1_Recognizer(A, out m, out π)
3      loop:
4          Split(A, out m′, in/out π)
5          if m = m′:
6              break
7          m ← m′
8      Q′ ← ∅,   δ′ ← ∅,   F′ ← ∅
9      for each q ∈ Q:
10         q′ ← π[q.i]
11         if q ∈ F:
12             F′ ← F′ ∪ {q′}
13         if q′ ∉ Q′:
14             Q′ ← Q′ ∪ {q′}
15             for each x ∈ X:
16                 δ′ ← [⟨q′, x⟩ → π[δ(q, x).i]]δ′
17     A′ ← ⟨Q′, X, δ′, F′, π[q0.i]⟩
```

§7. Регулярный язык

Пусть дан конечный алфавит X.

Определение 7.1. Назовём *словом* кортеж, составленный из символов алфавита X.

Кортеж $\langle a_0, a_1, \ldots, a_{n-1} \rangle$ по сути является строкой, и его принято записывать как $a_0 a_1 \ldots a_{n-1}$. При этом пустой кортеж обозначают буквой ε.

Определение 7.2. *Язык* – это множество слов.

Определение 7.3. Если $s_1 = a_0 a_1 \ldots a_{n-1}$ и $s_2 = b_0 b_1 \ldots b_{m-1}$ – слова в алфавите X, то операция *конкатенации* этих слов, записываемая как $s_1 \cdot s_2$, формирует новое слово
$s_1 \cdot s_2 = a_0 a_1 \ldots a_{n-1} b_0 b_1 \ldots b_{m-1}$.

Определение 7.4. Операция *конкатенации* языков L_1 и L_2, записываемая как $L_1 \cdot L_2$, формирует новый язык
$L_1 \cdot L_2 = \{s_1 \cdot s_2 \mid s_1 \in L_1 \text{ и } s_2 \in L_2\}$.

Если L – некоторый язык, то запись L^n, где $n > 0$, означает язык $\underbrace{L \cdot L \cdot \ldots \cdot L}_{n}$.

Для общности будем считать, что $L^0 = \{\varepsilon\}$.

Определение 7.5. Операция *замыкания* языка L, записываемая как $L^\star$, формирует новый язык
$L^\star = L^0 \cup L^1 \cup L^2 \cup \ldots$

Определение 7.5. Операция *позитивного замыкания* языка L, записываемая как L^+, формирует новый язык
$L^+ = L^1 \cup L^2 \cup \ldots$

Определение 7.6. Язык L в алфавите X называется *регулярным* тогда и только тогда, когда для него справедливо хотя бы одно из следующих утверждений:

1. $L = \emptyset$;
2. $L = \{\varepsilon\}$;
3. $L = \{a\}$, где $a \in X$;
4. $L = L_1 \cup L_2$, где L_1 и L_2 – регулярные языки;
5. $L = L_1 \cdot L_2$, где L_1 и L_2 – регулярные языки;
6. $L = L_1^\star$, где L_1 – регулярный язык.

Пример. (Регулярный язык)
Язык, состоящий из всех возможных слов в алфавите X.

Пример. (Регулярный язык)
Язык, состоящий из всех возможных слов вида $\underbrace{aaa\ldots a}_{n}$.

Пример. (Нерегулярный язык)
Язык, состоящий из всех возможных слов вида $\underbrace{aaa\ldots a}_{n}\underbrace{bbb\ldots b}_{n}$.

Утверждение 7.1. (Первая часть теоремы Клини) Язык распознающего автомата регулярен.

§8. Недетерминированный распознающий автомат

Недетерминированные автоматы используются главным образом в качестве вспомогательного инструмента для построения рассмотренных нами ранее детерминированных распознающих автоматов.

Определение 8.1. *Недетерминированный распознающий автомат* – это пятёрка $A = \langle Q, X, \delta, F, q_0 \rangle$, в которой:
Q – конечное множество состояний;
X – конечный алфавит;
$\delta : Q \times (X \cup \{\lambda\}) \hookrightarrow 2^Q$ – частичная функция переходов;
$F \subseteq Q$ – множество *принимающих* (или *заключительных*) состояний;
$q_0 \in Q$ – начальное состояние.

В недетерминированном автомате допустимы переходы по одному символу сразу в несколько состояний, а также разрешены λ-переходы. Кроме того, могут существовать такие q и a, что из q нет переходов по a.

Представление недетерминированного автомата в памяти компьютера отличается тем, что матрица переходов Δ имеет дополнительный столбец для λ, и каждая её ячейка содержат список (или массив) номеров состояний.

Вспомогательный алгоритм Closure вычисляет множество состояний недетерминированного автомата, достижимых из множества состояний z по λ-переходам. Это множество, кроме того, содержит z в качестве подмножества.

```
1 Closure(in Δ, in z): C
2       C ← ∅
3       for each q ∈ z:
4           Dfs(Δ, q, in/out C)
5 Dfs(in Δ, in q, in/out C)
6       if q ∉ C:
7           C ← C ∪ {q}
8           for each w ∈ Δ[q, λ]:
9               Dfs(Δ, w, in/out C)
```

(Алгоритм выполняет обход автомата в глубину по λ-переходам.)

Считается, что недетерминированный автомат принимает последовательность символов, если существует последовательность переходов по этим символам из начального состояния в одно из принимающих состояний. При этом при λ-переходе символ из входной последовательности не считывается.

```
1  RunNdRecognizerRec(in Δ, in Final, in q, in s): accept
2      accept ← Search(Δ, Final, q, s, 0)
3  Search(in Δ, in Final, in q, in s, in i): accept
4      if i = len(s):
5          accept ← Final[u]
6      else:
7          x ← s[i]
8          for each u ∈ Closure(Δ, {q}):
9              for each w ∈ Δ[u, x]:
10                 if Search(Δ, Final, w, s, i + 1):
11                     accept ← true
12                     return
13         accept ← false
```

Можно составить алгоритм «параллельной» интерпретации недетерминированного автомата. В этом алгоритме не будет возвратов, потому что мы будем считать, что автомат в процессе работы находится сразу в нескольких состояниях:

```
1  RunNdRecognizer(in Δ, in Final, in q, in s): accept
2        z ← Closure(Δ, {q})
3        i ← 0
4        while i < len(S):
5              x ← s[i]
6              z ← Closure(Δ, ⋃_{u∈z} Δ[u, x])
7              i ← i + 1
8        for each u ∈ z:
9              if Final[u]:
10                   accept ← true
11                   return
12       accept ← false
```

На основе идеи «параллельной» интерпретации недетерминированного автомата основан алгоритм детерминизации.

Действительно, при «параллельной» интерпретации мы проходим через последовательность множеств состояний недетерминированного автомата. Если принять, что состояниями эквивалентного детерминированного автомата будут являться все множества, через которые может пройти «параллельный» интерпретатор, мы можем построить для него функцию переходов.

Итак, если дан недетерминированный автомат $A = \langle Q, X, \delta, F, q_0 \rangle$, то эквивалентный ему детерминированный автомат $B = \left\langle Q', X, \delta', F', q_0' \right\rangle$ строится следующим образом:
$Q' = \left\{ \text{Closure}\,(z) \mid z \in 2^Q \right\}$,
$\delta'\,(z, x) = \text{Closure}\left(\bigcup\limits_{q \in z} \delta\,(q, x) \right)$,
$F' = \{ z \mid z \cap F \neq \emptyset \}$,
$q_0' = \text{Closure}\,(q_0)$.

Алгоритм детерминизации:

```
1  Det(in X, in Δ, in Final, in q): ⟨Q, X, δ, F, q0⟩
2        q0 ← Closure(Δ, {q})
3        Q ← {q0},   δ ← ∅,   F ← ∅
4        InitStack(out s)
5        Push(s, q0)
6        while not StackEmpty(s):
7              z ← Pop(s)
8              for each u ∈ z:
9                    if Final[u]:
10                         F ← F ∪ {z}
11                         break
12             for each a ∈ X:
13                   z' ← Closure(Δ, ⋃_{u∈z} Δ[u, a])
14                   if z' ∉ Q:
15                         Q ← Q ∪ {z'}
16                         Push(s, z')
17                   δ ← [⟨z, a⟩ → z'] δ
```

Утверждение 8.1. (Вторая часть теоремы Клини) Можно построить распознающий автомат, языком которого будет являться любой регулярный язык.

▷ По регулярному языку строим недетерминированный автомат:
...на доске...
Детерминизируем полученный недетерминированный автомат. ◁

Определение 8.2. Дополнение $\overline{L}$ языка L – это множество всех слов в алфавите языка L, не принадлежащих L.

Следствие 1 из утверждения 8.1. Дополнение регулярного языка регулярно.

▷ Строим распознающий автомат по языку в соответствии с утверждением 8.1. Делаем его принимающие состояния непринимающими, а непринимающие – принимающими. Язык получившегося автомата будет являться дополнением заданного языка, и по утверждению 7.1 он будет регулярным. ◁

Следствие 2 из утверждения 8.1. Пересечение регулярных языков регулярно.

▷ $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ ◁

Следствие 3 из утверждения 8.1. Разность регулярных языков регулярна.

▷ $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$ ◁