目 次

オートマトンNOTE

後期中間試験までの範囲分

1　集合論の基礎知識

▷ オートマトンにおいては, **集合論**の考え方が非常に多く登場し, 理論の核となっている. 集合論の基礎知識を, まず確認しよう.

1.1　集合 (set)

▷ **集合 (set)** とは, いくつかのもの (有限または無限) からなる「集まり」のことである. 集合の要素を**元 (element)**, または単に**要素 (element)** という. 集合を記述する際には, 以下のような記法を用いる.

- 集合の名前には, 通常
  アルファベットの大文字を用いる.
  (例　A, B, X, Y, P)
- 集合の要素を記述する際は, 以下のように { } (中カッコ) を用いて, 中に要素を並べて書く.

  例　$P = \{0, 1, 2, 3, 4, 5, 6\}$

  また, 次のような記法も非常によく使われる.

  例　$A = \{x \mid 0 \leq x \leq 1000\}$

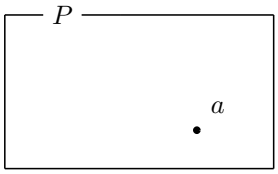
  多くの要素を持つ集合や, 無限集合を記述するときは, 後者の記法のほうが使いやすい.

また, 集合は, 単に**ものの集まり**のことなので,
要素の順番は基本的に関係ない.

1.2　集合と集合, 集合と要素の関係

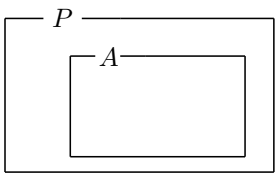
▷ 集合 P に, ある**要素** a が含まれているとき, このことを以下のように表す.

$a \in P$　(a は, 集合 P の要素である)



▷ 集合 P に, ある**集合** A が含まれているとき, このことを以下のように表す.

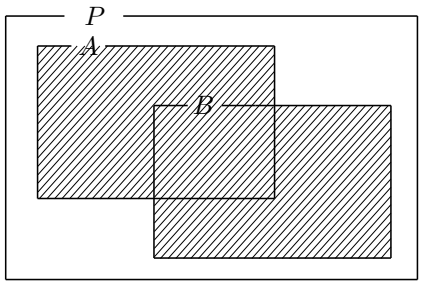
$A \subset P$　(集合 A は, 集合 P の部分集合である)



1.3　和集合 (union)

▷ 任意の集合 A, B を考えたとき,A または B に含まれる要素を全て集めて作った新たな集合を,
A と B の**和集合 (union)** といい, 以下のように表す.

$$A \cup B = \{x \mid x \in A \ \ or \ \ x \in B\}$$

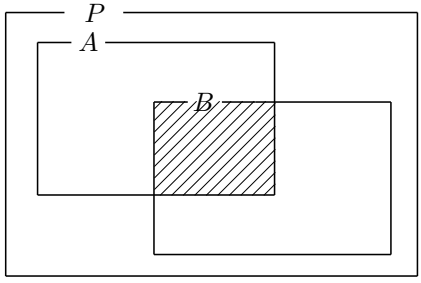


$A \cup B$: 図の斜線部分

1.4　積集合 (intersection)

▷ 任意の集合 A, B を考え,A と B に共通して含まれる要素を全て集めて作った新たな集合を,A と B の**積集合 (intersection)** といい, 以下のように表す.

$$A \cap B = \{x \mid x \in A \ \ and \ \ x \in B\}$$



$A \cap B$: 図の斜線部分

1.5 べき集合 (power set)

▷ 任意の集合 A を考え,A の部分集合全てを要素とするような新たな集合を,A の**べき集合 (power set)** といい, 以下のように表す.

$$2^A = \{P \mid P \subset A\}$$

たとえば,$A = \{a, b\}$ を考えると,A の部分集合は

$$\{a, b\}, \{a\}, \{b\}, \phi(\text{空集合})$$

の 4 つなので,2^A は, 以下のような集合となる.

$$2^A = \{\{a, b\}, \{a\}, \{b\}, \phi\}$$

べき集合 2^A の要素数は, 必ず $2^{(A \text{ の要素数})}$ 個となる. べき集合を 2^A と表現するのは, その事実に由来する. 要素に**空集合** ϕ を忘れないように気をつけよう.

1.6 直積集合 (cartesian product set)

▷ 任意の集合 A,B を考え,A, B から 1 つずつ要素を取り出した全ての**組 (pair)** を要素とするような新たな集合を,A,B の**直積集合 (cartesian product set)** という. 直積集合は以下のように表現する.

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

(a, b) の中の a は A の要素,b は B の要素である. この順序は, 例えば次のような場合に入れ替わる.

$$B \times A = \{(b, a) \mid b \in B, a \in A\}$$

一般に,$(a, b) \neq (b, a)$ であり, 組には順番が関係ある (順序対である) ということに注意が必要である.

例 $A = \{a, b\}, B = \{c, d\}$ とする.

$$A \times B = \{(a, c), (a, d), (b, c), (b, d)\}$$

$$B \times A = \{(c, a), (c, b), (d, a), (d, b)\}$$

2 記号, 記号列

▷ 記号 (文字) の有限集合を A とする.

例 ひらがな $A = \{$ あ, い, う, $\cdots$ わ, を, ん $\}$

アルファベット $A = \{A, B, C, \cdots, X, Y, Z\}$

A の要素を（重複を許して）並べたものを,A **上の記号列**という. 例えば, 以下のようなものがある.

例 $A = \{a\}$

A 上の記号列：$a, aa, aaa, aaaa, \cdots, aa \cdots aa, \cdots$

例 $B = \{a, b\}$

B 上の記号列：$a, b, ab, aba, bb, aabb \cdots, \cdots$

2.1 記号列の長さ (length)

記号列が含む記号の個数を, **記号列の長さ**という.

例 $|abaabba| = 7$

2.2 空列 ε (epsilon)

▷ 長さが 0 の記号列を**空列**といい, ε と表す.

$$|\varepsilon| = 0$$

2.3 スター閉包 (star closure)

▷ 任意の集合 A のあらゆる記号列を集めた集合を, A の**スター閉包 (star closure)** という.

$$A^*$$

例 $A = \{a\}$

$$A^* = \{\varepsilon, a, aa, aaa, aaaa, aaaaa, \cdots\}$$

例 $B = \{a, b\}$

$$B^* = \{\varepsilon, a, b, ab, ba, aaa, aab, \cdots\}$$

2 つ目の例は, **長さ**に着目すると, 分かりやすく以下のように記号列を分類して書き出せる.

0：ε
1：a, b
2：aa, ab, ba, bb
3：$aaa, aab, aba, abb, baa, bab, \cdots$

スター閉包には, **必ず空列 ε が含まれる**ことに注意!!

2.4 言語 (language)

▷ 任意の記号の集合 A を考えたとき,A^* の部分集合を**言語 (language)** という.

例 $\bigstar = \{\ a, b, c, \cdots, z, A, B, \cdots, Z\ \}$ とする.
▷ あらゆる**英単語** は,$\bigstar^*$ に含まれる.

$\therefore$ **英単語**は,$\bigstar^*$ の**部分集合**である.

2.5 連接 (concatenation)

▷ 文字列と文字列を繋げて, 新しい文字列を作る演算を**連接 (concatenation)** という.

現実においては, **ある文字列のあとに, 続けて文字列を書く**ということに相当する.

2.5.1 空列 ε との連結

▷ 任意の文字列 ω に対して,

$$\varepsilon\varepsilon\varepsilon\cdots\omega\cdots\varepsilon\varepsilon\varepsilon = \omega$$

となる. よって,ε は, **連接に対する単位元**[1]である.

3 決定性有限オートマトン (DFA)

▷ 有限オートマトンは, 次のような 5 つの要素によって決まる **5 項組 (5-tuple)** のシステムである.

- Q : **状態の有限集合**
  オートマトンが持っている状態の集合.
  例 $Q = \{q_0, q_1, q_2, q_3\}$
- $\sum$: **入力記号の有限集合**
  オートマトンに入力できる記号の集合.
  例 $\sum = \{0, 1\}$
- δ : **状態遷移関数**
  現状態と入力によって, 遷移先を返す関数.
  例 $\delta(\underbrace{q_0}_{\text{現状態}}, \underbrace{0}_{\text{入力}}) = \underbrace{q_1}_{\text{遷移先}}$
- q_0 : **初期状態**
  一番最初（無入力）の時の状態.
- F : **最終状態の集合**
  止まると受理してもらえる状態の集合.
  例 $F = \{q_1, q_3\}$

これら 5 つの項目が与えられれば, **オートマトンが定義**されたことになる. つまり, **オートマトンが一意に定まる**.5 つの項目によって定まるオートマトンを, まとめて次のように表現する.

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

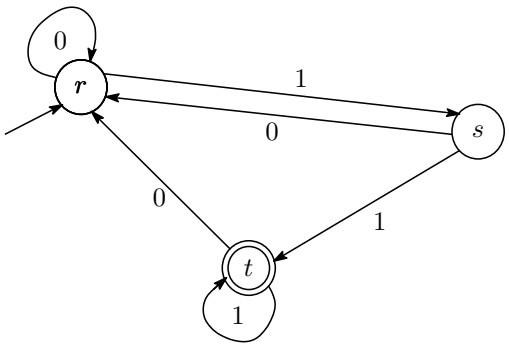
これが, オートマトンの形式的定義である.

3.1 状態遷移グラフ

▷ オートマトン $M = \{Q, \Sigma, \delta, q_0, F\}$ を考える.

- $Q = \{r, s, t\}$ (状態の有限集合)
- $\Sigma = \{0, 1\}$ (入力の有限集合)
- $\delta(r,0) = r,\ \delta(r,1) = s,\ \delta(s,0) = r$
  $\delta(s,1) = t,\ \delta(t,0) = r,\ \delta(t,1) = r$
  (状態遷移関数 δ(現状態, 入力))
- $q_0 = r$ (初期状態)
- $F = \{t\}$ (最終状態の集合)

このオートマトンを, 視覚的に見やすく, 以下のようにグラフとして書きなおすことができる.



このようなグラフを**状態遷移グラフ**という. 状態遷移グラフでは,

- 初期状態には始点のない矢印をつける.
- 最終状態は二重丸で表す.

という重要な決まりごとがある.

[1] 演算を施しても全く影響が無い要素のこと. 例えば, 加法の単位元は 0, 乗法の単位元は 1. こんな感じで, 連接の単位元は ε.

3.2 状態遷移表

▷ オートマトンを表で表現することもできる.

	0	1
$\rightarrow r$	r	s
s	r	t
ⓣ	r	t

最左列が状態, 最上行が入力を表す. これにより, **ある状態にある入力を行ったときに, どこに遷移するか？**がコンパクトに表現できている.

この表を, **状態遷移表**という. 状態遷移表では

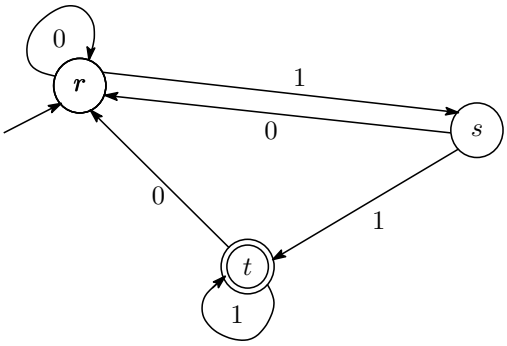
- 初期状態には矢印をつける.
- 最終状態は◯で囲む.

という重要な決まりごとがある.

3.3 受理と却下

▷ 入力記号の集合 Σ の要素から成る記号列（Σ^* の要素）をオートマトンに入力する.

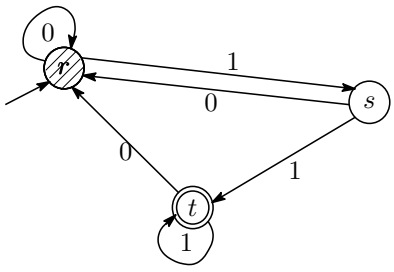
例



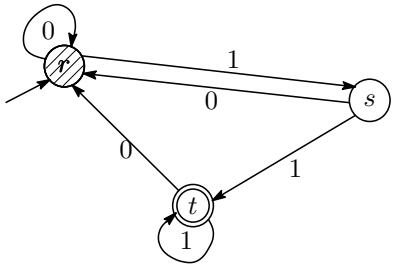
$\Sigma = \{0, 1\}$ なので, 入力記号は 0 か 1.

▷ "0011"を入力する.

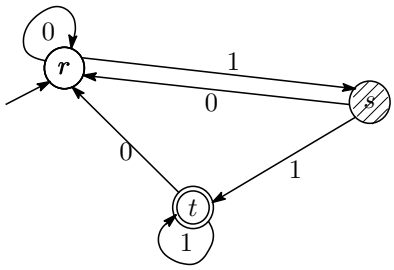
0 を入力 $\boxed{\underbrace{0}\ 011}$



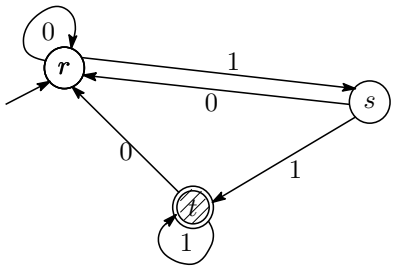
0 を入力 $\boxed{0\ \underbrace{0}\ 11}$



1 を入力 $\boxed{00\ \underbrace{1}\ 1}$



1 を入力 $\boxed{001\ \underbrace{1}}$

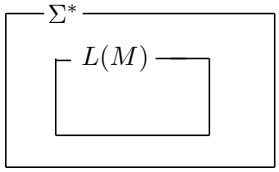


⇒ 入力を終えた時点で, **最終状態に止まった.**

このようなとき, 入力"0011"は**受理された**という.
最終状態に止まらなかった場合, **却下された**という.

3.4 DFA の能力

▷ DFA を使うと, 入力記号列の集合 Σ^* を, **全て受理と却下に分けることができる**.



Σ^*：**入力記号列**の集合
$L(M)$：オートマトン M が**受理する記号列**の集合

このことから,**DFA は, ある入力が受理か却下かを判定する機械である**ということができる.

▷ q_0(初期状態) $\in F$(最終状態の集合) なら $\cdots$

何も入力しなくても (ε を入力) 受理!!

3.5 DFA の等価性

▷ ふたつの DFA M_1, M_2 が等価（同じ）であるということは, **どんな入力を行なっても, 全く同じ出力が返される**ということである. つまり, **オートマトンの構造がどうなっていても, 入力と出力が完全に等しければその 2 つのオートマトンは等価**である.

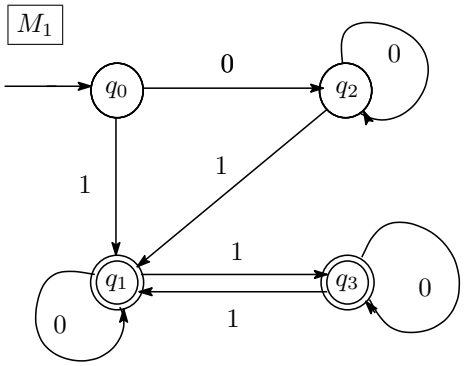
3.6 等価性判定木

▷ ふたつのオートマトン M_1, M_2 が等価であると判断するためには, 全ての入力に対して出力が等しいことが示せれば良い. が, **入力記号列は無限に存在するので, しらみつぶしに調べることは不可能**である.

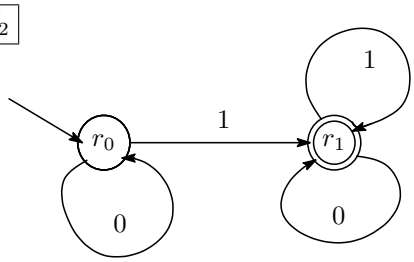
そこで用いる方法が, **等価性判定木**である.

例　次の M_1, M_2 の等価性を判定する.

M_1



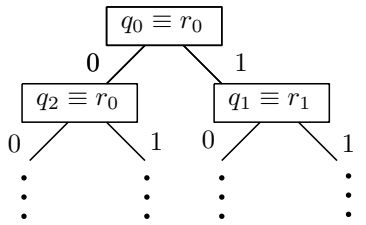
M_2



1. M_1, M_2 の初期状態 q_0, r_0 が, 最終状態であるか, そうでないかを調べる.

   - 共に最終状態, または最終状態でない.
     $\Rightarrow$ 次の手順に進む.
   - 片方が最終状態, もう片方が異なる.
     $\Rightarrow$ M_1, M_2 は, 明らかに異なる. **(終了)**

2. q_0, r_0 をスタートに, 入力に応じた遷移先を木の枝分かれのように描いて行く.



   - 一度出てきた状態対がまた現れる
     $\Rightarrow$ その状態対は木を打ち切る.
   - 片方が最終状態, もう片方は最終状態でないような, 状態対が現れる
     $\Rightarrow$ M_1, M_2 は, 異なる. **(終了)**
   - すべての状態対が打ち切れる
     $\Rightarrow$ M_1, M_2 は区別できない!!
     $\therefore$ M_1, M_2 は等価である.**(終了)**

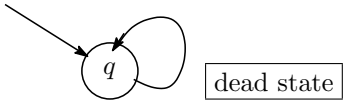
以上の手順が, 等価性判定木を用いた,DFA の等価性判定の方法である.

3.7 狭義の DFA

▷ 狭義の DFA とは, 今まで扱ってきた DFA のことである. つまり, **状態と入力に対する遷移先は必ず１つ**しかないという特徴がある.

3.8 死状態 (dead state)

▷ **その状態からはもうどこにも行けない**ような状態を, **死状態 (dead state)** という.



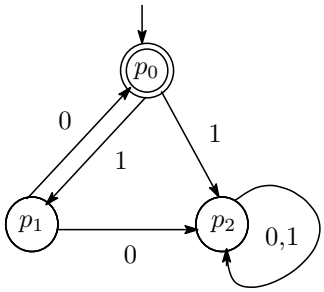
3.9 広義の DFA

▷ 広義の DFA とは, 遷移先が”無い”状態もあり得るオートマトンである. このことは, 広義の DFA は, **遷移先がたかだかひとつ**の DFA と言い換えることができる.

広義の DFA

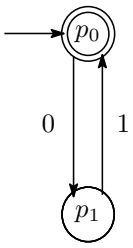
- 状態と入力に対し, 遷移先はたかだか１つ. （遷移先が無い場合もあり得る）
- 遷移先のない入力は却下される..

例えば, 次のオートマトン M_1（狭義の DFA）を考える.



たとえば, 入力”1101”を与えた場合の動作を考えると, **状態 p_2 からどこにも行けない**という状態に陥ってしまう. **つまり,p_2 は死状態**であることが分かる. 死状態 p_2 に陥った場合, 入力は必ず却下される.

死状態は, 状態としてまったく意味をもたないので, 死状態を取り除くと次のような,M_1 **と等価なオートマトン** M_2 を作ることが出来る.



先程と同じように入力”1101”を考えると, 最初の入力「1」と状態 p_0 に対する遷移先がない (広義の DFA) ので, 結局この入力は却下される.

このように, **死状態に移る**ということを**遷移先がない**という風に読み替えれば, **狭義の DFA を広義の DFA に解釈できる**し, **その逆も当然可能**である.

4 非決定性有限オートマトン (NFA)

▷ 非決定性有限オートマトン (NFA) は, 決定性有限オートマトン (DFA) を拡張したオートマトンである.NFA と DFA の違いは, 次のひとつだけである.

- DFA：遷移先はたかだかひとつ.
- NFA：**遷移先が何個でも良い.**

つまり,NFA では, 状態と入力に対する遷移先がいくつあっても構わない.0 個でも 1 個でも,2 個でも 10 個でも 100 個でも良いのである.

4.1 形式的定義の違い

▷ NFA も,DFA と同様に **5 項組**によって一意に定まる.

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

が,NFA の定義には DFA とは異なる点がある.

まずは,DFA の場合, 状態遷移関数 δ によって定まるのは, ひとつの遷移先だった

$$\delta\colon Q \times \Sigma \to Q$$

(状態遷移関数の戻り値は, 状態の有限集合 Q の要素)

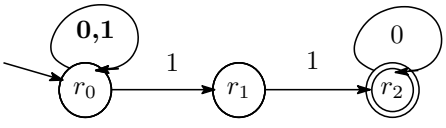
ところが,NFA の場合は, 遷移先がひとつとは限らない. このことは以下のように表現できる.

$$\delta\colon Q \times \Sigma \to 2^Q$$

(状態遷移関数の戻り値は,Q のべき集合 2^Q の要素)

※直積集合, べき集合に関しては,p3 1.5,1.6 を参照

例



このオートマトンは NFA である. なぜかというと, **状態 r_0 と入力 1 に対する遷移先が,r_0, r_1 のふたつ**あるからである. このことを数学的に表現すると, 以

下のように表すことが出来る.

$$\delta(r_0, 1) = \{r_0, r_1\} \in 2^Q$$

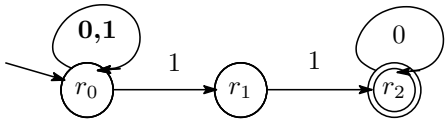
$$(2^Q = \{\phi, \{r_0\}, \{r_1\}, \{r_2\}, \underbrace{\{r_0, r_1\}}, \{r_0, r_2\}, \{r_1, r_2\}, \{r_0, r_2\}, \{r_0, r_1, r_2\}\})$$

4.2 NFA の動作

▷ NFA は, 次のように動作する.

- 入力に対して, **最終状態に止まる遷移方法がひとつでもあるならば, 受理**.
- **どんな遷移方法を考えても, 最終状態に止まることができないならば, 却下**.

たとえば, 先程の NFA に入力"110"を与えることを考えてみよう.



遷移方法

$$\rightarrow r_0 \underbrace{\rightarrow}_{1\text{ を入力}} r_0 \underbrace{\rightarrow}_{1\text{ を入力}} r_1 \underbrace{\rightarrow}_{0\text{ を入力}} \boxed{\text{却下}}$$

(最後に却下されたのは, 遷移先が無いから)

遷移方法

$$\rightarrow r_0 \underbrace{\rightarrow}_{1\text{ を入力}} r_1 \underbrace{\rightarrow}_{1\text{ を入力}} r_2 \underbrace{\rightarrow}_{0\text{ を入力}} r_2 \quad \boxed{\text{受理}}$$

このように,NFA の場合は, 同じ入力記号列でも, 遷移方法によって受理と却下が異なる場合がある. そして, このような場合は, **受理される遷移方法が１つでもあれば, 入力記号列は受理とみなす**のである.

つまり, 上の例での入力記号列 "110" は, **受理される.**

この NFA の状態遷移表は以下のようになる.

	0	1
$\rightarrow r_0$	$\{r_0\}$	$\{r_0, r_1\}$
r_1	ϕ	$\{r_2\}$
(r2)	$\{r_2\}$	ϕ

集合をあらわす {} を忘れないように気をつけよう.

4.3 NFA から DFA へ

▷ NFA は,DFA の遷移先の個数を自由にしたオートマトンなので,DFA を拡張したものであることは明らかである. つまり, 以下のようなことがいえる.

$$\textbf{NFA} \subset \textbf{DFA} \quad (\textbf{NFA は DFA を含む})$$

一方, 実は, 以下のような手順を行うことによって, **NFA を DFA としてみる**ことも可能である.

- 2^Q の要素を, ひとつの状態として見る.
  (2^Q は, 要素として集合を持つが, その集合ごとひとつの「かたまり」として見ることによって, 集合ごと１つの状態とみなす)
- 初期状態は, 開始記号のみからなる状態集合**（初期状態の集合）**からスタートする.
- 推移先は, それぞれの状態の推移先の**和集合**とする（現状態から推移する可能性のある状態すべての集合）.

このような手順により, 任意の NFA と**等価な DFA** を構成できるので, 次のようなこともいえる.

$$\textbf{DFA} \subset \textbf{NFA} \quad (\textbf{DFA は, NFA を含む})$$

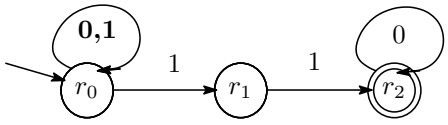
これらが同時に成り立っているということは, 以下のような結論が得られる.

$$(\textbf{NFA} \subset \textbf{DFA}) \wedge (\textbf{DFA} \subset \textbf{NFA})$$
$$\rightarrow \boxed{\textbf{NFA} = \textbf{DFA!!}}$$

つまり,**NFA が持っている能力と DFA が持っている能力は実質的には同等**ということである.

4.4 NFA と等価な DFA の作成

例



この NFA と等価な DFA を作成してみよう. そのために, 最初に状態遷移表を作る.

	0	1
$\rightarrow r_0$	$\{r_0\}$	$\{r_0, r_1\}$
r_1	ϕ	$\{r_2\}$
ⓡ$_2$	$\{r_2\}$	ϕ

まず, 初期状態は r_0 のみなので, 初期状態の集合は $\{r_0\}$ となる. このことを, 新たな表に書き加えよう.

	0	1
$[r_0]$		

このとき, 最左列では集合をあらわす {} は用いない. これは, 集合を"集合として"ではなく,**"ひとつの状態として"みなす**からである.

次に, **初期状態の集合から遷移する可能性のある状態**をまとめ, それぞれの入力に対する遷移先とする.

r_0 に 0 を入力したら必ず r_0 に遷移し,
1 を入力したら r_0 か r_1 に遷移するので,

	0	1
$[r_0]$	$[r_0]$	$[r_0, r_1]$

そして, この遷移先として現れた「かたまり」を, 今度は新たな状態とみなし, 同様の操作を繰り返す.

	0	1
$[r_0]$	$[r_0]$	$[r_0, r_1]$
$[r_0, r_1]$		

r_0, r_1 のそれぞれに 0 を入力すると,r_0 にしか推移しない. また, それぞれに 1 を入力すると r_0, r_1, r_2 に推移できるので,

	0	1
$[r_0]$	$[r_0]$	$[r_0, r_1]$
$[r_0, r_1]$	$[r_0]$	$[r_0, r_1, r_2]$

このような操作を最後まで繰り返すと, 以下のような新たな表が完成する.

	0	1
$[r_0]$	$[r_0]$	$[r_0, r_1]$
$[r_0, r_1]$	$[r_0]$	$[r_0, r_1, r_2]$
$[r_0, r_1, r_2]$	$[r_0, r_2]$	$[r_0, r_1, r_2]$
$[r_0, r_2]$	$[r_0, r_2]$	$[r_0, r_1]$

ところで, もとの NFA における最終状態は,r_2 のみであった. 上の表を見ると,r_2 を含んでいる「かたまり」が 2 つ現れている.

「かたまり」が最終状態を含んでいるということは, **そのかたまりに遷移すれば, 最終状態 r_2 に遷移する可能性がある**ということである.
NFA の場合, **最終状態に遷移する方法がひとつでもあれば受理される**という約束があったので, 上の表の最終状態は,r_2 を含む

$$[r_0, r_1, r_2], [r_0, r_2]$$

のふたつであるということが言える.

それぞれの状態を, わかりやすく再定義しよう.

$$[r_0] \rightarrow q_0$$
$$[r_0, r_1] \rightarrow q_1$$
$$[r_0, r_1, r_2] \rightarrow q_2$$
$$[r_0, r_2] \rightarrow q_3$$

表を書き換えると,

	0	1
$\rightarrow q_0$	q_0	q_1
q_1	q_0	q_2
ⓠ$_2$	q_3	q_2
ⓠ$_3$	q_3	q_1

これが, **もとの NFA と等価な DFA の状態遷移表**である. このように,NFA を DFA に変換できるということが示せた.

5　オートマトンの簡単化

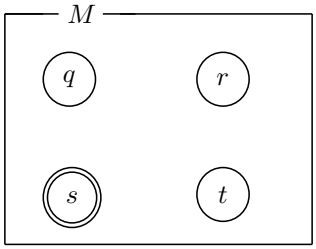
▷ オートマトンは, 状態数を減らして, 簡単な, 等価なオートマトンに変形できる場合がある. その方法について考えてみよう.

5.1　状態の等価性

▷ オートマトンに等価性があったように, **オートマトンの各状態にも等価性がある**. 状態の等価性とは, 以下のように定義される.

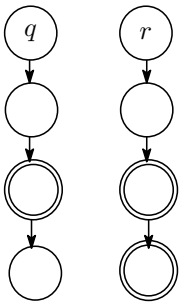
- ふたつの状態を入力によって区別できれば, ふたつの状態は等価でない.
- ふたつの状態を入力によって区別できなければ, ふたつの状態は等価である.

例　状態 q, r が等価であるとは？



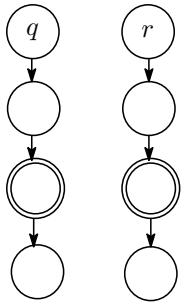
上のように,4 つの状態を持つオートマトン M を考える. そして, この内の 2 状態 q, r が等価であるとはどういうことかを考えよう.

case1
状態 q, r に, 同じ入力を与え, 図のように遷移した.



この場合, 最終的に, **最終状態とそうでない状態に** q, r **がそれぞれ辿りついている**ので,q, r は異なる. つまり, 等価でないといえる.

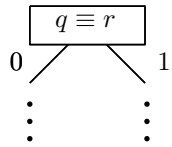
case2
状態 q, r に, 同じ入力を与え, 図のように遷移した.



この場合, **どちらの遷移先にも違いがない**. よって, 状態 q, r を区別することはできない. **すべての入力に対して遷移先に違いがなければ,**q, r **は等価**である.

5.2　等価性判定木

▷ 状態の等価性の判定にも, 等価性判定木を用いる. 方法も, オートマトンの等価性判定とほとんど同じである.



ただし · · ·

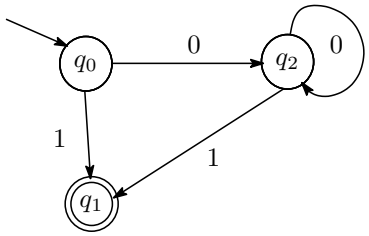
- 同じ状態 $(q \equiv q)$ にたどり着く ⇒ **打切**
- 逆の状態 $(r \equiv q)$ にたどり着く ⇒ **打切**

この違いにだけは, くれぐれも注意しよう.

5.3　オートマトンの簡単化

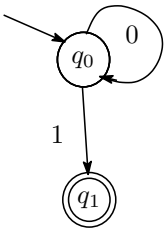
▷ 等価な状態をひとつとみなせば, オートマトンの状態数は減ることになる. この操作ことが, オートマトンの簡単化そのものである.

例



この NFA において, **状態 q_0, q_2 は等価**である.

2 つの等価な状態をうまく重ね合わせると, 等価な**(状態数も減った)**オートマトン（下図）を得られる.

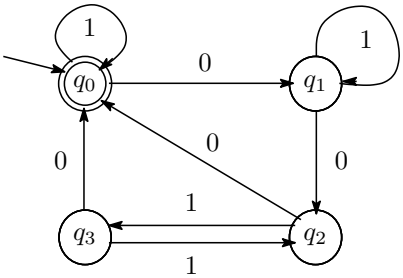


5.4 最小化アルゴリズム

▷ オートマトンは, 簡単化を続けると「これ以上簡単化できない」状態（**もっとも簡単な状態**）に最終的に辿り着く. これから, その「**もっとも簡単な状態**」を手に入れるためのアルゴリズムを考えよう.

オートマトンをもっとも簡単な状態に直すことを, オートマトンの**最小化**という. 例として, 以下のようなオートマトンを最小化してみよう.

例



オートマトンを最小化するということは, 等価な状態を全て見つけ出すということなので, 状態同士の関係が分かりやすいよう, 以下のような表を作る.

q_0				
q_1				
q_2				
q_3				
	q_0	q_1	q_2	q_3

この表を用いて, 以下のようなアルゴリズムでオートマトンを最小化する.

- 等価でないと分かった状態に×印をつける
- **最後まで残った（×が付かなかった）マスに対応する状態対が, 等価な状態対!!**

さて, 最小化を始める前に, 次の点に注意しよう.

- たとえば,"q_0と q_0" , "q_2と q_2"のように, **自分と自分が等価なのは当然**である.
- たとえば,"q_0と q_1 が等価"だということと,"q_1と q_0 が等価"だということは同じことである. **(等価だということに状態の順番は関係ない)**

よって, 結局調べなければいけないのは, 以下の残った部分だけであることがわかる.

q_0	-	-	-	-
q_1		-	-	-
q_2			-	-
q_3				-
	q_0	q_1	q_2	q_3

では, ステップを追って進んで行こう.

1. **"最終状態 (q_0)"**と
   "最終状態でない状態 (q_1, q_2, q_3)"は等価でない.

q_0	-	-	-	-
q_1	×	-	-	-
q_2	×		-	-
q_3	×			-
	q_0	q_1	q_2	q_3

2. (q_1, q_2) に 0 を入力すると,(q_2, q_0) に遷移する.
q_0 は最終状態.q_2 はそうでないので, **遷移先が区別できる**. よって,(q_1, q_2) は等価でない.

q_0	-	-	-	-
q_1	×	-	-	-
q_2	×	×	-	-
q_3	×			-
	q_0	q_1	q_2	q_3

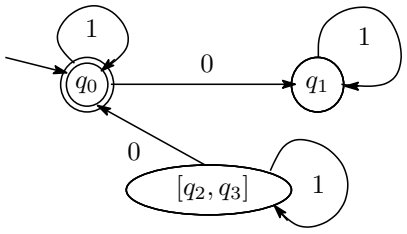
3. (q_1, q_3) に 0 を入力すると,(q_2, q_0) に遷移する.
先程と同様の理由により,(q_1, q_3) は等価でない.

q_0	-	-	-	-
q_1	×	-	-	-
q_2	×	×	-	-
q_3	×	×		-
	q_0	q_1	q_2	q_3

4. (q_2, q_3) に 0 を入力 $\rightarrow (q_0, q_0)$.
(q_2, q_3) に 1 を入力 $\rightarrow (q_3, q_2)$.
遷移先が区別できないので, 等価.

q_0	-	-	-	-
q_1	×	-	-	-
q_2	×	×	-	-
q_3	×	×	○	-
	q_0	q_1	q_2	q_3

よって, 等価なのは q_2と q_3 であることがわかった.
これをもとに等価な（最小化した）オートマトンをつくると, 以下のようになる.



状態が等価であると判断するときは, かならず**遷移先が区別できるかどうか**をよく考えるべし!!　以上が, オートマトンの最小化アルゴリズムである.

5.5　ε 動作を許す NFA (ε-NFA)

▷ ε 動作を許す NFA とは, **入力として空列**[2]**を許可**した NFA である. 数学的には, 次のように表現できる.

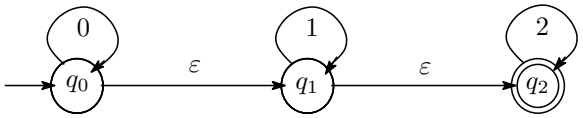
DFA : $\delta : Q \times \Sigma \rightarrow Q$
（状態に入力を与えたとき, 遷移先は**ひとつ**の状態）

NFA : $\delta : Q \times \Sigma \rightarrow 2^Q$
（状態に入力を与えたとき, 遷移先は **0 個以上**の状態）

ε-NFA : $\delta : Q \times (\Sigma \cup \varepsilon) \rightarrow 2^Q$
（状態に入力を与えたとき, 遷移先は **0 個以上**の状態）
⇒ **ただし, 入力として空列 (ε) も許す!!**

以下に ε-NFA の例を示す.

$$M = (\underbrace{\{q_0, q_1, q_2\}}_{Q}, \underbrace{\{0, 1, 2\}}_{\Sigma}, \delta, q_0, \underbrace{\{q_2\}}_{F})$$



このように, 入力記号列の集合 Σ の要素のほかに,ε が, 入力記号として許されていることがわかる. この M がどのような入力を受理するのかを考えよう.

1. **入力記号列 "00"**

   素直に"00"を入力すると**却下**されるように見えるが,"$00 = 00\varepsilon\varepsilon$" なので, この入力は**受理**.
   （NFA なので, **受理される遷移方法がひとつでもあるなら, その入力は受理**される!!）

2. **入力記号列 "01"**

   "$0\varepsilon 1\varepsilon = 01$" より, **受理**.

3. **入力記号列 "012"**

   "$0\varepsilon 1\varepsilon 2 = 012$" より, **受理**.

4. **入力記号列 "10"**

   1 を入力したあと,0 に戻れない ので, **却下**.

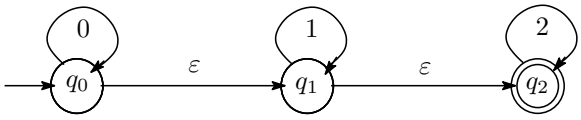
[2] p3 2.2 : **空列 ε(epsilon)** を参照

5.6 ε の除去

▷ 入力記号として ε が許されていると, **一見却下されそうなものが受理されたり, 紛らわしくて分かりにくい**. よって, これから, ε-NFA を, **等価な NFA に変換**する方法を考える (ε の除去).

例として, 次の ε-NFA の,ε を除去してみよう.

例



まずは, **状態遷移表**を作る.

	0	1	2	ε
$\rightarrow q_0$	$\{q_0\}$	$\{\}$	$\{\}$	$\{q_1\}$
q_1	$\{\}$	$\{q_1\}$	$\{\}$	$\{q_2\}$
ⓠ$_2$	$\{\}$	$\{\}$	$\{q_2\}$	$\{\}$

これは, ε **もひとつの入力としてみなした状態遷移表**である. これを作る手順は, 今までと全く同様で良い.

この次に,ε を除去した状態の状態遷移表を作る. その手順を今から説明しよう. その表の基本形は, 次のように**入力から ε を取り除いただけ**のものである.

	0	1	2
$\rightarrow$ ⓠ$_0$			
q_1			
ⓠ$_2$			

ここで注意しなければならないのは,q_0 **が最終状態になっていることである**. 理由は, **何も入力しなくても（ε のみの入力でも）初期状態の q_0 から, 最終状態の q_2 に遷移できるので,q_0 も最終状態とみなせるから**である. これには十分注意しよう.

状態 q_0 に 0 を入力した場合を考えよう. ここで考えなければならないのは,**0 を入力するのと,**$\varepsilon\varepsilon\cdots 0\cdots\varepsilon\varepsilon$ **を入力することは同じ**ということである.

つまり,**0 の前後に ε を補って到達できる状態も,0 を入力したときに到達できる状態のひとつとしてみなす必要がある**.

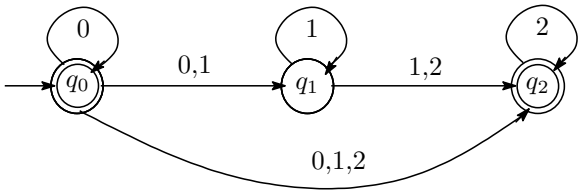
0 の前後に ε を補うと,q_0 から q_0, q_1, q_2 に遷移できるので, 表は以下のようになる.

	0	1	2
$\rightarrow$ ⓠ$_0$	$\{q_0, q_1, q_2\}$		
q_1			
ⓠ$_2$			

このような手順（**入力記号の前後に ε を補った際の遷移先を全て考える**）を繰り返すと, 表は以下のとおりとなる.

	0	1	2
$\rightarrow$ ⓠ$_0$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	$\{\}$	$\{q_1, q_2\}$	$\{q_2\}$
ⓠ$_2$	$\{\}$	$\{\}$	$\{q_2\}$

これをそのままグラフに描くと, 以下のような, **等価な NFA の状態遷移グラフ**が完成する.



このような手順で, 任意の ε-NFA は, 等価な NFA に変換できる. よって, 次のような結論が得られる.

$$\varepsilon - \mathbf{NFA} \subset \mathbf{NFA}$$

ところで,ε-NFA は,NFA を拡張したものだったので,

$$\mathbf{NFA} \subset \varepsilon - \mathbf{NFA}$$

この 2 つの命題が同時になりたつということは, 結局のところ

$$\mathbf{NFA} = \varepsilon - \mathbf{NFA}$$

という結論が得られる.

6 正則表現 (RE)

▷ 例えば,Linux のコマンドラインでファイルをまとめて指定するときに”ls *.gif”などと入力するように, パターンを指定する表現を**正則表現 (regular expression:RE)** という.

6.1 正則集合 (regular set)

▷ 正則集合とは, 以下のように定義される集合である.

1. ϕ(空集合) は**正則集合**である.
2. $\{\varepsilon\}$ は**正則集合**である.
3. $\{a\}$ は**正則集合**である. (ただし,$a \in \Sigma$)
4. R, S がそれぞれ正則集合であるとすると,
   (a) 和集合 $R \cup S$ は**正則集合**である.
   (b) 連接 RS は**正則集合**である.
   (c) スター閉包 R^*, S^* は**正則集合**である.

正則集合とは, このように作られる集合の総称である. 4. より, **正則集合同士の和集合, 連接などを考えることにより, 再帰的に正則集合を大きくして行くことができる.**

6.2 正則表現 (regular expression)

▷ たとえば, 正則集合をいちいち { } を使って書くのは面倒である. その手間を省くために, 正則集合に記号を対応づけたものが**正則表現**である. 正則集合に対応した正則表現を, 以下に表として示す.
(ただし,R, S は正則集合とする.)

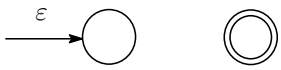
正則集合	正則表現
ϕ	ϕ
$\{\varepsilon\}$	ε
$\{a\}$	$\mathbf{a}$
$R \cup S$	$\mathbf{R + S}$
RS	$\mathbf{RS}$
R^*	$(\mathbf{R})^*$

6.3 正則集合に対応する有限オートマトン

▷ 正則集合には, 対応した有限オートマトンを必ず作ることができる. **どんなに複雑な正則集合でも, 以下に示す基本的な形を組み合わせることで等価な有限オートマトンを作れる**.

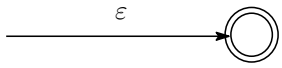
ということで, 以下に, 正則集合に対応する, 等価な有限オートマトンを示す.

6.3.1 $\phi : \phi$



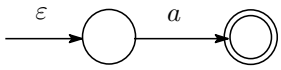
(決して受理しない)

6.3.2 $\{\varepsilon\} : \varepsilon$



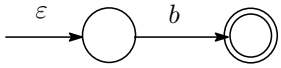
(いきなり受理)

6.3.3 $\{a\} : \mathbf{a}$



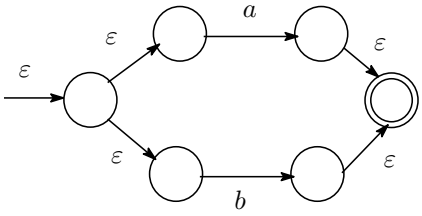
(1 文字で受理)

6.3.4 $\{b\} : \mathbf{b}$



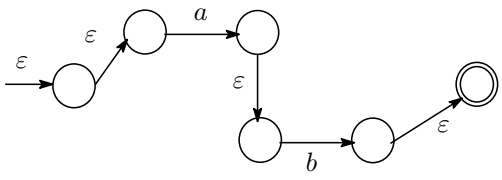
(1 文字で受理)

6.3.5 $\{a\} \cup \{b\}$ (和集合) : **a+b**



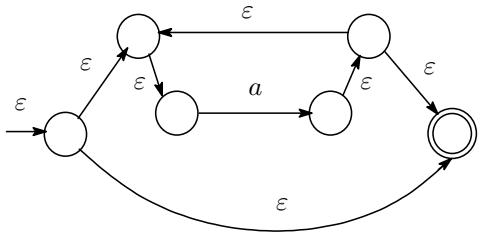
(a または b で受理)

6.3.6 $\{a\}\{b\}$ (連接) : **ab**



(a のあとに b で受理)

6.3.7 $\{a\}^*$ (スター閉包) : $(\mathbf{a})^*$



(a の 0 回以上の繰り返しで受理)

★ これらの基本形の組み合わせで, 任意の正則集合に対応した有限オートマトンを作ることができる!!

6.4 DFA から正則表現へ

▷ 今までは, 正則表現をオートマトンに変換するということを考えてきたが, 今度は, **オートマトン (DFA) から正則表現を求める**ことについて考えてみよう.

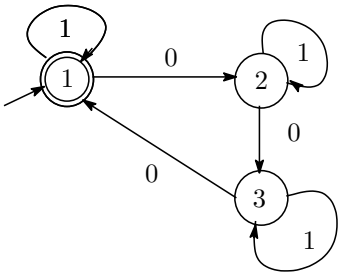
6.5 R_{ij}^k

▷ R_{ij}^k とは, 状態 i から j に遷移する方法 (path) の表現法である.

- オートマトンの各状態に番号をつける. (1,2,3,$\cdots$)
  ※番号の付け方は任意だが, 必ず 1 から!!
- R_{ij}^k は, 状態 $i \to j$ に, 状態 1〜k まで経由して良いような遷移のしかたを表す.

つまり,R_{ij}^k は 状態 i から j に,状態 1〜k のみを通って行く経路を表す.(経由しなくても OK)

例



このオートマトンを R_{ij}^k で表現すると,

$$R_{11}^3$$

となる.

(1(初期状態) から 1(最終状態) へ行く.
1〜3 の状態 (全部) を経由して良い)

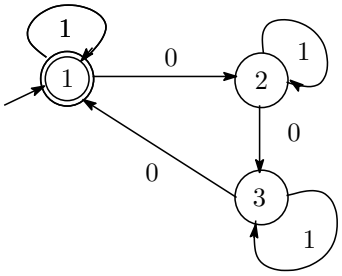
6.5.1 $k = 0$

▷ 特に $k = 0$ の場合,

$$R_{ij}^0$$

は, どの状態も経由せずに (つまり, 直接) $i \to j$ に辿り着く経路を表す. これは, グラフを一目みただけで判断できる.

例



この場合,R_{ij}^0 は, 以下のようになる.

$$R_{11}^0 = \{1, \varepsilon\}, R_{12}^0 = \{0\}, R_{13}^0 = \{\}$$

$$R_{21}^0 = \{\}, R_{22}^0 = \{1, \varepsilon\}, R_{23}^0 = \{0\}$$

$$R_{31}^0 = \{0\}, R_{32}^0 = \{\}, R_{33}^0 = \{1, \varepsilon\}$$

また,R_{ij}^0 を数学的に表現すると, 以下のように表現することができる.

if $i \neq j$

$$R_{ij}^0 = \{x \mid a \in \Sigma, \delta(q_i, a) = q_j\}$$

if $i \neq j$

$$R_{ij}^0 = \{x \mid a \in \Sigma, \delta(q_i, a) = q_j\} \cup \{\varepsilon\}$$

6.5.2 $k \neq 0$

▷ $k \neq 0$ の場合は少々ややこしい. 数学的表現を先に述べると, 以下のようになる.

$$R_{ij}^k = R_{ij}^{(k-1)} \cup R_{ik}^{(k-1)} (R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$$

とてもわかりにくい式だが, この式の各部分の意味は次の通りである.

$$R_{ij}^{k} = \underbrace{R_{ij}^{(k-1)}} \cup \underbrace{R_{ik}^{(k-1)}}\underbrace{(R_{kk}^{(k-1)})^{*}}\underbrace{R_{kj}^{(k-1)}}$$

k を通らない場合

k を通る場合

$i \to k$ に,k を通らずに遷移する経路
$(k \to k$ に, k を通らずに推移する経路$)^{*}$
$k \to j$ に,k を通らずに推移する経路

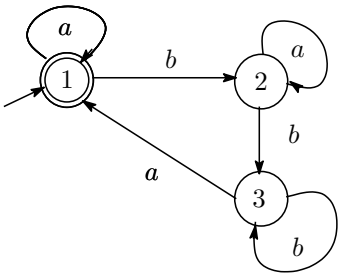
この式は, 暗記をすることよりも, **意味を理解することが大切**である. この漸化式を用いれば, **再帰的に経路を求めることができる**.

ちなみに, ただ $k \neq 0$ の場合の経路のパターンを調べたいだけなのであれば, 必ずしもこの漸化式を用いる必要はない. **どのような経路があり得るかを頭で考えれば, 答えは導き出すことができる**.
(具体的な方法については後述)

例　先程の例では

$$R_{11}^{3} = R_{11}^{2} \cup R_{13}^{2}(R_{33}^{2})^{*}R_{31}^{2}$$

例　R_{ij}^{2} $(i, j \in Q)$ を求めよう.



では早速, 実際に R_{ij}^{2} を求めてみよう.R_{ij}^{2} の意味を考えると, 求めたいのは, **状態 1,2 を通って, ある状態からある状態に遷移する経路**ということがわかる.

まず, 次のような表をつくる.

	$k=2$
R_{11}^{k}	
R_{12}^{k}	
R_{13}^{k}	
R_{21}^{k}	
R_{22}^{k}	
R_{23}^{k}	
R_{31}^{k}	
R_{32}^{k}	
R_{33}^{k}	

1. R_{11}^{2}

   初期状態で, **何も入力しない**, または a **を何度も入力する (ぐるぐる回る)** ということが考えられるので,$R_{11}^{2} = a^{*}$（**正則表現!!**) となる.

2. R_{12}^{2}

   初期状態で,b を一度だけ入力する.
   初期状態で,a を何度も入力（ぐるぐる）し, b を入力し, 状態 2 で a を入力（ぐるぐる）.
   初期状態で,a を何度も入力（ぐるぐる）し, b を入力.
   初期状態で,b を入力し,
   状態 2 で a を入力（ぐるぐる）

   これらを合わせると,$R_{12}^{2} = a^{*}ba^{*}$ となる.

このような要領ですべてのケースを求めて行くと, 最終的に以下のような結果となる.

	$k=2$
R_{11}^{k}	a^{*}
R_{12}^{k}	$a^{*}ba^{*}$
R_{13}^{k}	$a^{*}ba^{*}b$
R_{21}^{k}	ϕ
R_{22}^{k}	a^{*}
R_{23}^{k}	$a^{*}b$
R_{31}^{k}	aa^{*}
R_{32}^{k}	$aa^{*}ba^{*}$
R_{33}^{k}	$aa^{*}ba^{*} + b + \varepsilon$

(最後の $+$ は, $\cup$ に対応する正則表現.)

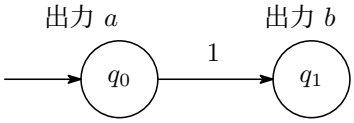
7　出力付きオートマトン

▷ 今まで扱ってきたオートマトンは,**受理**か**却下**の,2 通りの出力しか返ってこなかった.

ここで, 出力が 3 通り以上あるようなオートマトン**（出力付きオートマトン）**について考えよう.

7.1　ムーア機械

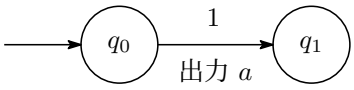
▷ ムーア機械とは, **各状態に遷移し終わった瞬間**に, 出力が返ってくるオートマトンである.



▷ 初期状態 q_0 $\underbrace{\rightarrow}_{a\text{ 出力}}$ $\underbrace{q_1}_{b\text{ 出力}}$

7.2　ミーリー機械

▷ ミーリー機械とは, **次状態に遷移し始める瞬間**に, 出力が返ってくるオートマトンである.



▷ 初期状態 q_0 $\underbrace{\rightarrow}_{a\text{ 出力}}$ q_1

7.3　ムーア機械とミーリー機械の関係

▷ 2 つの機械の出力を返すタイミングの違いから, 次のことが言える.

- **ムーア機械**
  入力記号列の長さ = 出力記号列の長さ.
- **ミーリー機械**
  入力記号列の長さ**+1** = 出力記号列の長さ.

ふたつの機械にはこのような違いがあるが, この些細な違いを無視すれば, 次のことが言える.

ムーア機械 = ミーリー機械

つまり, **ムーア機械はミーリー機械に変換できるし, その逆も当然可能**である.

7.4　P mod 3 計算機

▷ 出力付きオートマトンを使うと, **任意の 2 進数 P を 3 で割った余り（P mod 3）を機械的に求められるオートマトンが作れる**.

任意の 2 進数 P を考える. また,P を 10 進数に直した数を N とする.

$$[P]_2 = [N]_{10}$$

2 進数の「しっぽ」（後ろ）に 0 をつけると, 値が 2 倍になる.**(2 進数の左シフト**[3]**)**
また,2 進数の「しっぽ」に 1 をつけると, 値が 2 倍+1 になる.**(2 進数の左シフト+1)** よって,

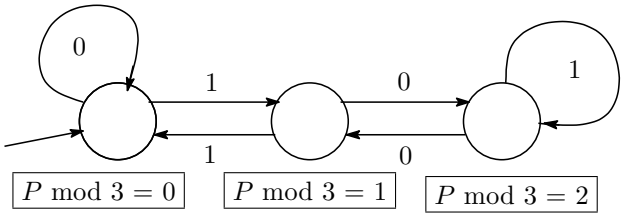
$$P = N, P0 = 2N, P1 = 2N + 1$$

($P0 : P$ のしっぽに 0 をつけた)
($P1 : P$ のしっぽに 1 をつけた.)

これらのことから, 次の表を作ることができる.

表現	値	mod 3		
P	N	0	1	2
$P0$	$2N$	0	2	1
$P1$	$2N+1$	1	0	2

この表をもとに, **任意の 2 進数 P の左側からの入力**[4]を前提とした, 出力付きオートマトンを作ると,



このオートマトンに入力するのは,mod 3 を求めたい 2 進数 P であり, 最終的に辿り着いた状態における出力は,P mod 3 である.

以上, 出力付オートマトン（ムーア機械）を用いた**「mod 3 計算機」**を作ることができた.

[3]例えば,10 のしっぽに 0 をつけて 10**0** にすると 10 倍になるように,2 進数はしっぽに 0 をつけると 2 倍になる.

[4]たとえば,1010110 だったら, 一番左に 0 があるとみなして (01010110 だとみなす) 最初に一番左の 0 を入力する. そして, そこから右に一桁ずつ順番に入力する.