形式言語とオートマトン

Note3　正規表現
2020.4.03作成
2020.4.27update 5.18update 5.23update 5.25 update
5.30 update
2021.4.26update 5.11 5.23 5.24
中野眞一　群馬大学

正規表現

正規表現は、語の集合の代数的な表現のひとつである。
(代数とは、集合とその上の演算のことである。)

アルファベットΣ上の正規表現とは、Σ中の記号、および、6つの記号
"(" と ")" と "*" と "+"(もしくは "|") と"ε"と"ϕ"からなる有限の長さの語のうち、
次のように再帰的に定義されるものである。
正規表現αが表す語の集合を$||\alpha||$と書く。

正規表現(つづき)

(1) Σ の要素である記号は、Σ上の正規表現である。(長さ1のもの)

a∈Σ のとき ||a|| = { a }

(2) ε と φ は それぞれΣ上の正規表現である。(長さ0の文字列と空集合)

|| ε || = { ε }

|| φ || = φ

(3) (長さ2以上のもの)

α と β を それぞれΣ上の正規表現とする。

(a) (α + β)はΣ上の正規表現である。(和)

||(α + β)|| ：= || α || ∪ || β ||

(b) (α β)はΣ上の正規表現である。(連接)

||(α β)|| ：= || α || ・ || β || (言語の連接) (記号・は略してもよい)

(c) (α *)はΣ上の正規表現である。(閉包)

||(α *)|| ：= || α ||* (言語Lの閉包)

(d) (～α) は Σ上の正規表現である。

||(～α)|| := Σ* - || α ||

正規表現（例）

例)　Σ ＝{ 1, 2, 3}

2 は Σ 上の正規表現である。 || 2 || ＝{ 2 } である。

ε は Σ 上の正規表現である。 || ε || ＝{ ε } である。

ϕ は Σ 上の正規表現である。 || ϕ || ＝ ϕ である。

(1+2)は Σ 上の正規表現である。|| (1+2) || ＝ || 1 || ∪ || 2 || ＝{ 1, 2 } である。

(2+3)は Σ 上の正規表現である。|| (2+3) || ＝{ 2, 3 } である。

((1+2)3)は Σ 上の正規表現である。|| ((1+2)3) || ＝{1,2}・{3} ＝{ 13, 23 } である。

((1+2)(2+3))は Σ 上の正規表現である。|| ((1+2)(2+3)) || ＝{ 12, 13, 22, 23 }

(2*)は Σ 上の正規表現である。||(2*)|| ＝{ ε , 2, 22, 222, ... }

((1+2)*)は Σ 上の正規表現である。

|| ((1+2)*) || ＝{ ε , 1, 2, 11, 12, 21, 22, 111, 112, ... }

正規表現

ただし、冗長なカッコは削除してもよい。
(和、連接、閉包の順に優先順位が強くなるとする。)

例1) a + bc*d は
　(a + ((b(c*))d))の意味です。

例2) (hot + cold)(apple + blueberry)(pie + tart)
　が表す語の集合は何かな？要素はいくつある？

正規表現

正規表現によって表すことのできる言語を正規言語
(もしくは正規集合 正則言語)という。

正規言語は、3つの演算、(1)和、(2)連接、(3)閉包について
閉じている。すなわち、
任意の正規言語に、3つの演算をしても、正規言語のままである。

正規表現の性質

α と β を それぞれ Σ 上の正規表現とする。 次が成り立つ。
(両辺とも言語を表すことに注意しよう。)(人間がわかりやすように変形できる!)

(1) $\alpha + \beta = \beta + \alpha$
(2) $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
(3) $\alpha + \phi = \alpha$
(4) $\alpha + \alpha = \alpha$
(5) $\alpha^* \alpha^* = \alpha^*$
(6) $(\alpha^*)^* = \alpha^*$
(7) $(\varepsilon + \alpha) = (\alpha + \varepsilon) \neq \alpha$ (最後の不等号が成立するときは、どんなときかな？)
(8) $\beta \beta^* \neq \beta^*$
(9) $(\alpha^* + \beta^*)^* = (\alpha^* \beta^*)^* = (\alpha + \beta)^*$

正規表現はいろいろなシステムで利用できます

UNIXのawk, grep, lex等のコマンドで正規表現が使用できる。
Python言語には正規表現モジュールが用意されている。
Ruby言語には正規表現クラスが用意されている。
Javascriptでも正規表現を扱うしくみがある。

UNIXにおける正規表現の拡張

UNIXでの拡張表現の例。

[aghinostw]のように "["と"]"で囲むと[ ]内の文字の集合を表す。
[a-z]で範囲内のどれか1文字の集合を表す。
^で行のはじめを示し、$で行の終りを示す。
. は任意の1文字を示す。(wild card)(don't care)
aの0回以上の繰り返しをa*で示す。
1回以上の繰り返しを+で示す。 a+ = aa* である。
aの0回もしくは1回の出現を a?で示す。
¥*　　特殊文字"*"を示す。

pythonでの拡張の例
{m}　直前の正規表現をちょうど m 回繰り返したもの　例えば、 a{6} は 6 個ちょうどの 'a' 文字
{m,n}?　前にある正規表現を、m 回から n 回まで繰り返したものにマッチし、できるだけ 少なく 繰り返したもの
その他たくさんあります。。。

/usr/share/dict/words でいろいろな語を検索してみよう。

%egrep man$ /usr/share/dict/words　(2291語)
%egrep woman$ /usr/share/dict/words　(176語)
%egrep ".*man$|.*son$" /usr/share/dict/words　(7932語)
%egrep "^a.*a$" /usr/share/dict/words　(1433語)

正規表現とFAの等価性

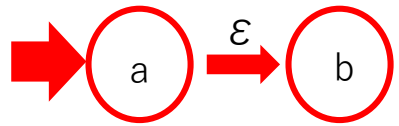
正規表現とNFAは互いに変換できることを示す。

正規表現 α が与えられたとき、$||\alpha|| = L(M)$ となる オートマトン M が構成できる。

オートマトン M が与えられたとき、$L(M) = ||\alpha||$ となる 正規表現 α が構成できる。

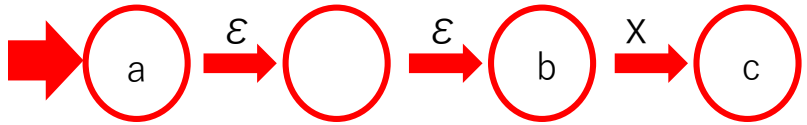
正規表現＝＞NFA

まず、NFAに、 ε 遷移を導入する。
状態aから状態bへ入力がなくても遷移できるとする。



ε 遷移つきのNFA Mは、次のようにして、(ε 遷移を用いない)通常のNFA Nに変換できる。

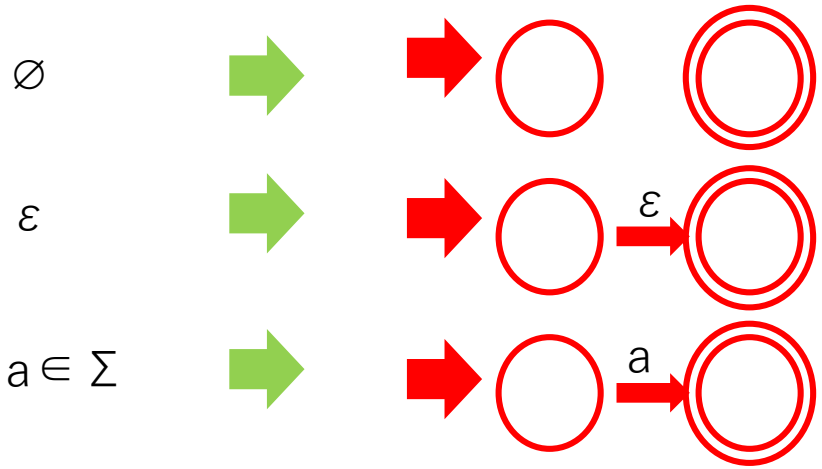
Mにおいて、状態aから ε 遷移のみを用いて状態bに遷移でき、
かつ、状態bから通常の遷移1回で(入力xとする)状態cに遷移できるとき、
Nにおいて、 δ(a,x)=cとする。



つまり、正規表現を、 ε 遷移つきのNFAに変換できれば、
(ε 遷移つきのNFAは、通常のNFAに変換できるので)
正規表現を通常のNFAに変換できることがわかる。

正規表現＝＞NFA

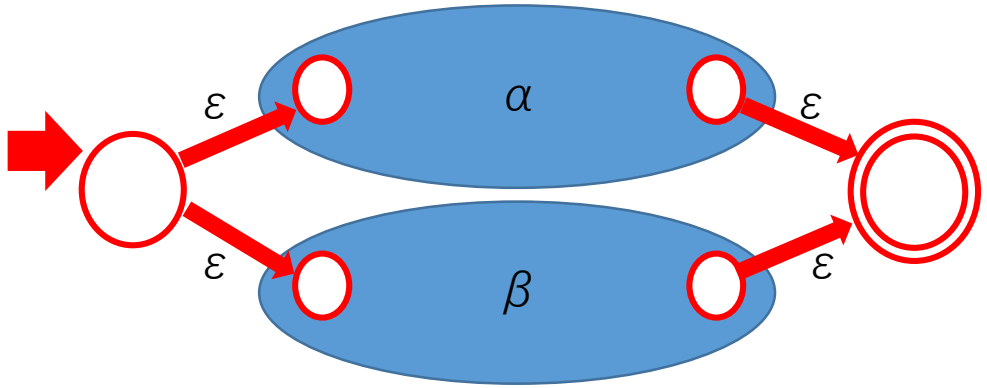
正規表現の定義に基づき、構成的にNFAを作成する方法を説明する。
定義のそれぞれについて構成方法を示す。



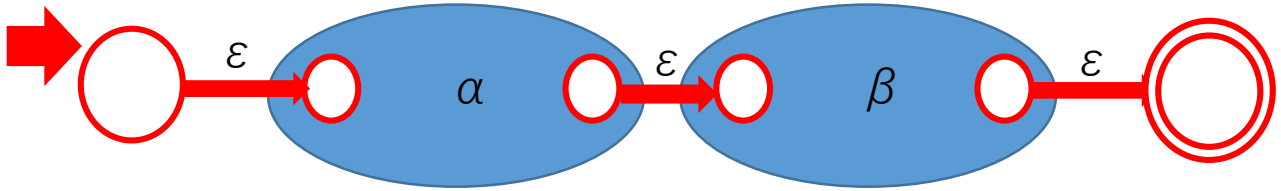
正規表現＝＞NFA（つづき）

（最後にまとめて、
ε 遷移を解消する。）

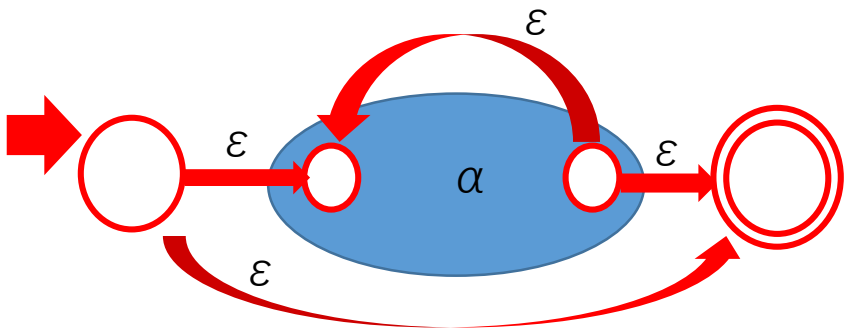
$\alpha + \beta$



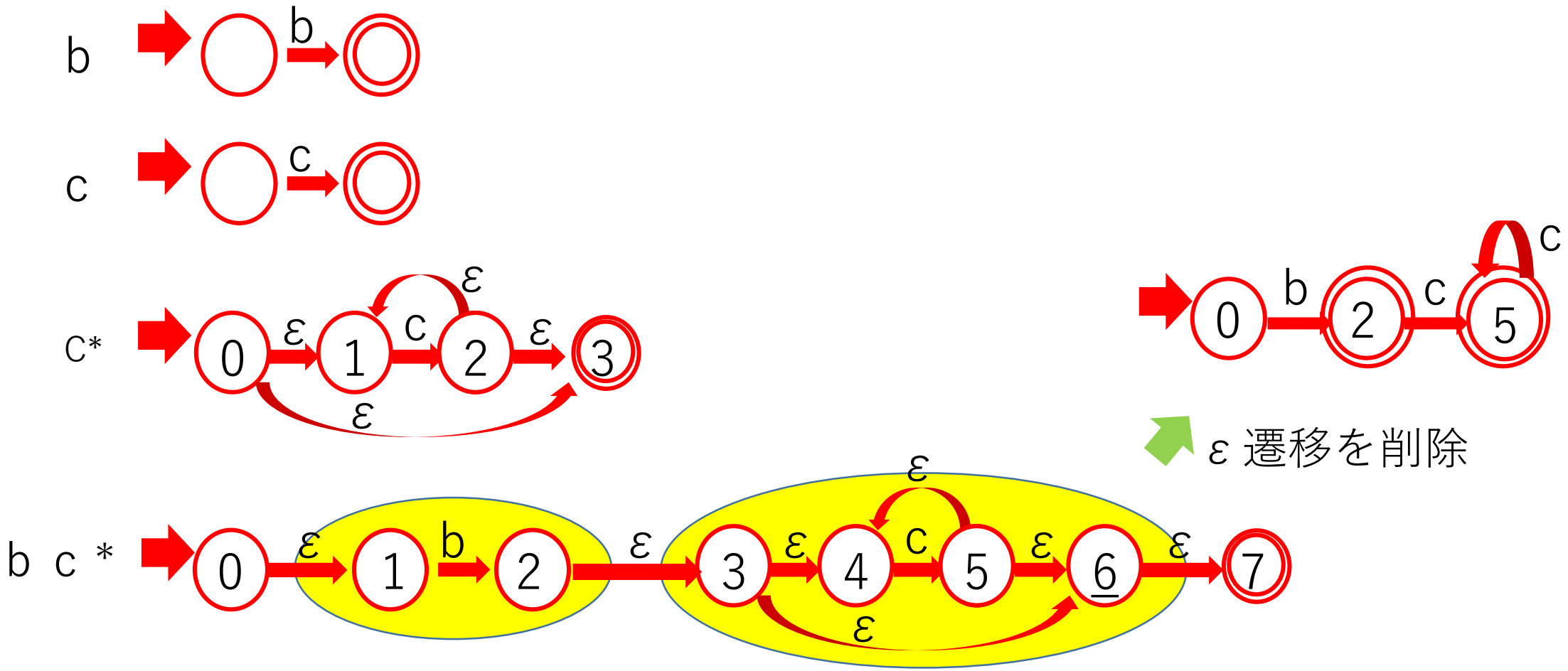
$\alpha\ \beta$



$\alpha *$



bc* に対応する オートマトン



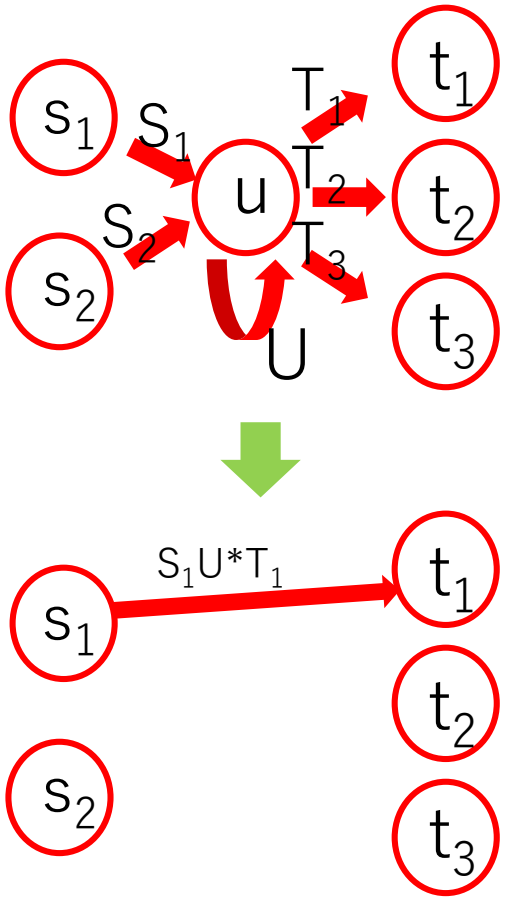
NFA => 正規表現

NFAの初期状態点以外の状態点を1つずつ消去する。
その際にNFAの辺のラベルを、次のように、順次、正規表現に置き換える。
点の個数が1個や2個になれば, 正規表現がわかる。(ゴールが多数あるときは、ゴールごとに正規表現を考える。)

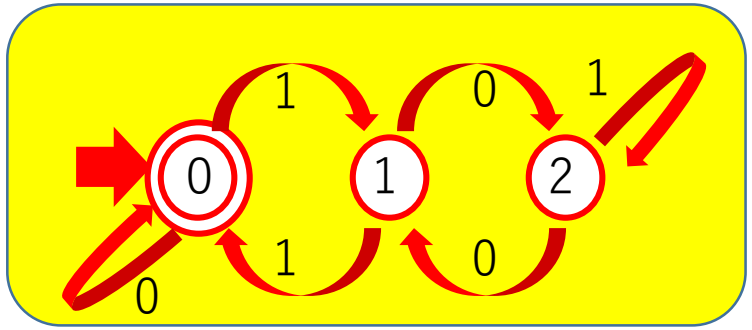
(base) 各辺のラベルを、長さ1の正規表現とみなす。
(induction) 点uを消すとする。
　uに向かう辺を、(s_1,u)、(s_2,u)、…、(s_n,u)とする。
　ラベルを、それぞれ、S_1、S_2、…、S_nとする。
　uから出ていく辺を、(u,t_1)、(u,t_2)、…、(u,t_m)とする。
　ラベルを、それぞれ、T_1、T_2、…、T_nとする。
　uからuへの辺(自己ループ辺)のラベルをUとする。
(もし、uからuへの辺がなければ、そのラベルをϕとする。
　各辺(s_i, t_j)の"現在の"ラベルをR_{ij}としよう。

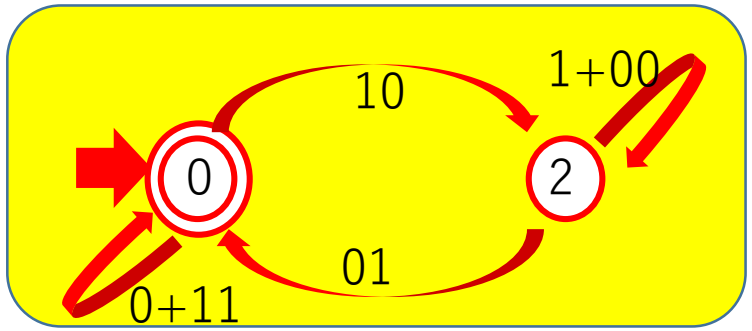
各辺(s_i, t_j)のラベルを $R_{ij} + S_i \quad U^* \quad T_j$に変更する。
(もし、$U=\phi$のときには $R_{ij} + S_iT_j$に変更するのと同じである。)

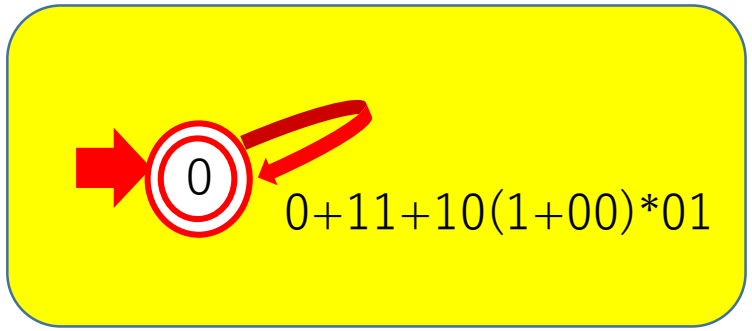
点uを削除しても、同じすごろくのまま。(ゴールするパスの集合は同じ!)



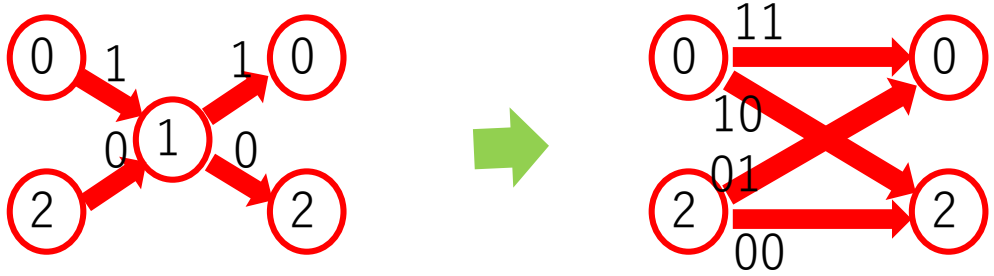
NFA => 正規表現　の例1



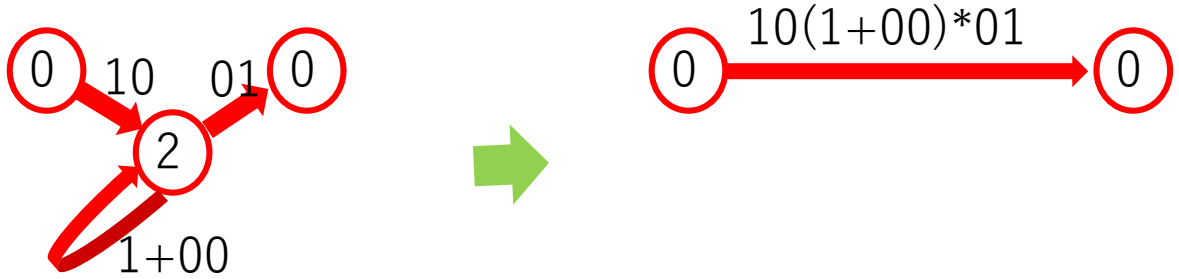




状態1を消す



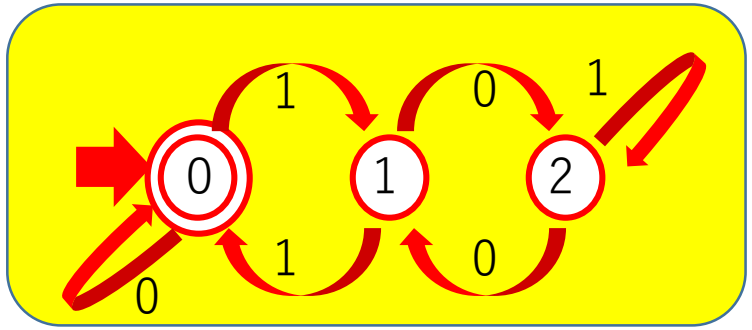
状態2を消す

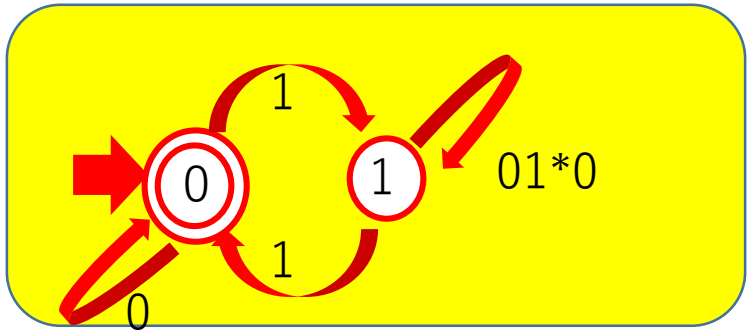


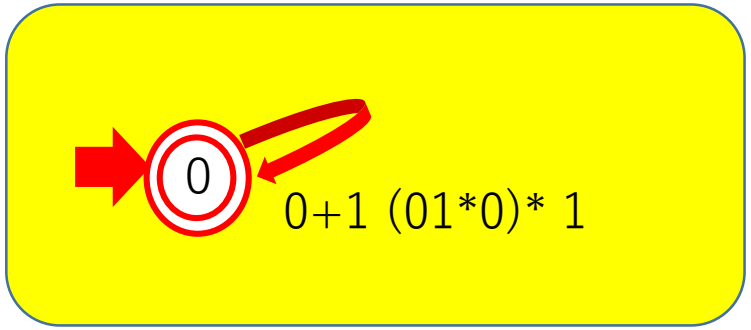
0, 11, 110, 1001, 1111,
10010, 10101, 11000,…

正規表現は(0 + 11 + 10(1+00)*01　)*

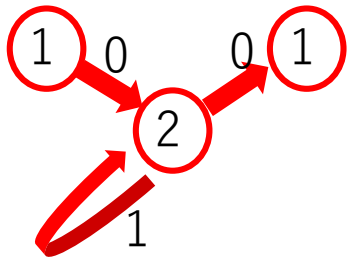
NFA => 正規表現　の例2

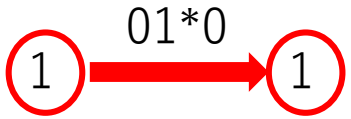




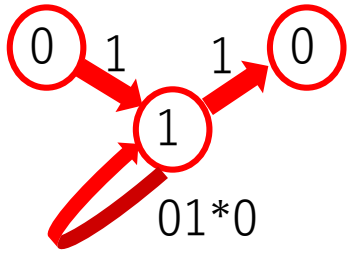


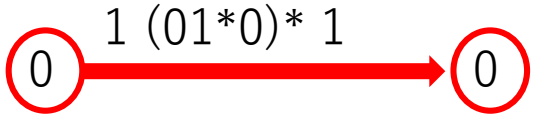
状態2を消す





状態1を消す

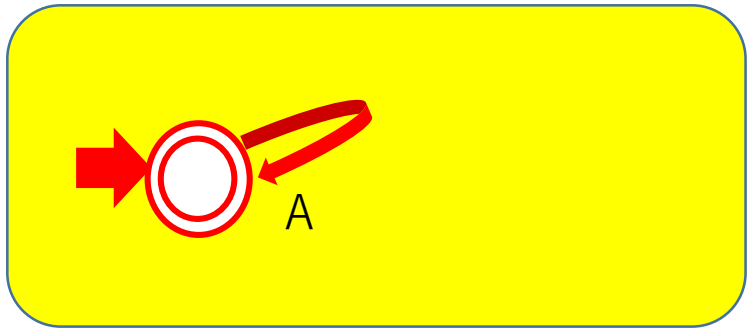




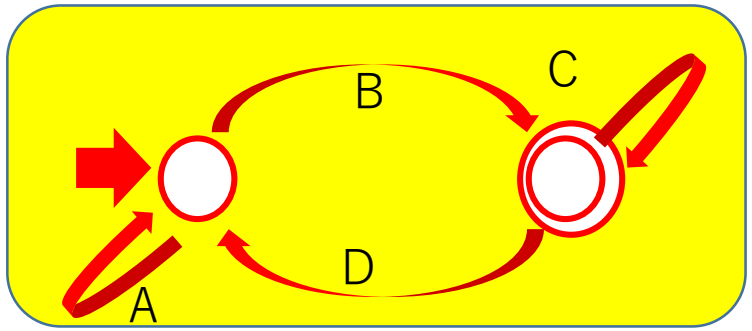
0, 11, 110, 1001, 1111,
10010, 10101, 11000,…

正規表現は(0 + 1(0 1* 0)*1)*

NFA => 正規表現　最後の変換



正規表現は(A)*



正規表現はA * B (C + DA*B)*

初ゴール

再ゴール

ポンプ定理

正規言語は必ず次の条件を満たす。(ワンパターンである！)

(ポンプ定理)

Lが正規言語(正規集合)ならば、次の条件を満たす正整数 n が存在する。

任意の $|x| \geqq n$ なる $x \in L$ は $x = uvw$ と書け、任意の整数 $i \geqq 0$ について

$uv^iw \in L$ である。

ただし、$|uv| \leqq n$ かつ $|v| \geqq 1$ である。

ポンプ定理の意味

Pumping lemma for regular languages, the fact that all sufficiently long strings in such a language have a substring that can be repeated arbitrarily many times, usually used to prove that certain languages are not reg

(https://en.wikipedia.org/wiki/Pumping_lemma)

ポンプの定理（証明）

任意の正規言語Lに対して、L(M) = L となるオートマトンM が存在する。このMの状態数を n とする。

長さがnより大きな任意の語 x ∈ L を選ぼう。 $x = a_1a_2...a_m$とする。

n < m なので、この入力に対するパスには、2回以上現れる状態がある。

このように2回以上現れる状態のうち、2回目が最も早く現れる状態をqとしよう。s文字目およびt文字目を読んだときに状態qとする

すなわち、$a_1a_2...a_s$まで入力を読んだとき状態qとなり、この後$a_{s+1}a_{s+2}...a_t$まで入力を読んだとき、再び状態qとなるとしよう。

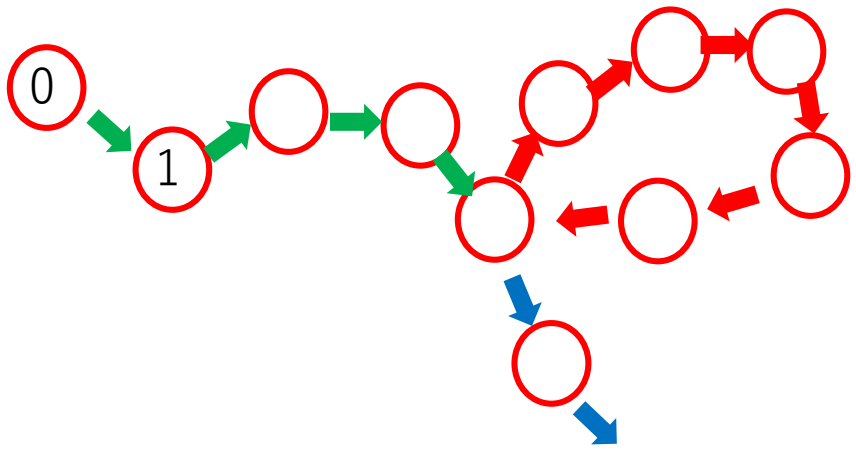
qの選び方より、$a_1, a_2, ..., a_{t-1}$はすべて異なる。

$u = a_1a_2...a_s$

$v = a_{s+1}a_{s+2}...a_t$

$w = a_{t+1}a_{t+2}...a_m$　とする。

(オートマトンの略図を書いてみよう。)



このとき、任意の整数 i ≧ 0 についてuv^iw ∈ L である。

また、|uv| ≦ n かつ |v| ≧ 1 である。(証明終)

正規表現の限界

言語 $L = \{ a^k b^k \mid k \geqq 0 \} = \{ \varepsilon, ab, aabb, aaabbb, ... \}$
は正規言語ではない.
(証明)
もし L が正規言語だとすると、ポンプの定理が成立するような 整数 n がある。
語 $x = a^{n+1}b^{n+1}$ とすると $|x| = 2n+2 > n$ なので、ポンプの定理より
$x = uvw$ と書け、(ただし、$|uv| \leqq n$ かつ $|v| \geqq 1$ である)
任意の整数 $i \geqq 0$ について、$uv^iw \in L$ である。
$|uv| \leqq n$ より uv はaのみからなる語である。
また、wは、1個以上のaと、 n+1 個の b を含む。
このとき、 $y = uv^2w \in L$ とはならない。
なぜならば、yに含まれるbの個数はxと同じであるが、aの個数は増えるからである。
よって矛盾する。(証明終)

正規表現の限界

対応のとれたカッコからなる言語も、
正規表現では記述できない。
(これまで、いくつ余分に左カッコが現れたかを、
有限の状態では記憶できないので。)
(もし、スタックが使えれば、記憶できるのですが。。。)

理解確認クイズ

(1) Σ={a,b,c}とする。Σ上の次の言語を正規表現で表せ。
 (a) 少なくとも3個のaを含む語からなる言語
 (b) 高々3個のaを含む語からなる言語
 (c) 最後から3文字目はaである語からなる言語
 (d) 最後から3文字目はaでない語からなる言語
 (e) 長さが5の語からなる言語
 (f) aの次の文字は必ずbである語からなる言語

(2) Σ={0,1}とする。Σ上の次の言語を正規表現で表せ。
 (a) 偶数個の1を含む語からなる言語
 (b) 4個以上1が連続する部分をもつ語からなる言語

(3) 正規表現 ((a+b)(a+b+d))* をオートマトンに変換せよ。

理解確認クイズ(1)

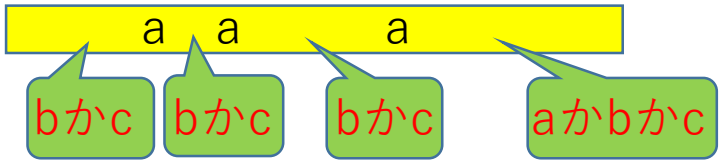
(1) Σ={a,b,c}とする。Σ上の次の言語を正規表現で表せ。

(a) 少なくとも3個のaを含む語からなる言語

(b+c)* a (b+c)* a (b+c)* a (a+b+c)*

(b) 高々3個のaを含む語からなる言語

(b+c)* + (b+c)*a(b+c)* + (b+c)*a(b+c)*a(b+c)* + (b+c)*a(b+c)*a(b+c)*a(b+c)*



つまりaは0個か1個か2個か3個

(c) 最後から3文字目はaである語からなる言語

(a+b+c)*a(a+b+c)(a+b+c)

(d) 最後から3文字目はaでない語からなる言語

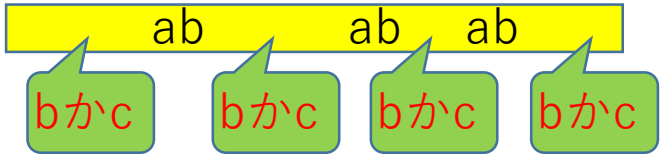
(a+b+c)*(b+c)(a+b+c)(a+b+c)

(e) 長さが5の語からなる言語

(a+b+c)(a+b+c)(a+b+c)(a+b+c)(a+b+c)

(f) aの次の文字は必ずbである語からなる言語

((b+c)* ab)* (b+c)*

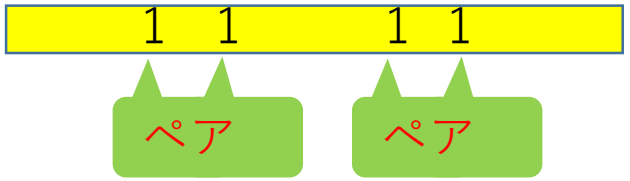


理解確認クイズ(2)

(2) Σ ={0,1}とする。Σ上の次の言語を正規表現で表せ。

(a) 偶数個の1を含む語からなる言語

(0*10*1)* 0*



(b) 4個以上1が連続する部分をもつ語からなる言語

(0+1)*1111(0+1)*

理解確認クイズ(3)

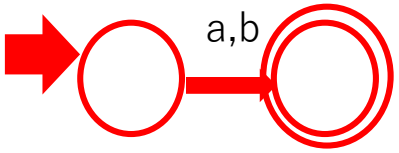
(3) 正規表現 ((a+b)(a+b+d))* をオートマトンに変換せよ。

実際に挑戦してみましょう！

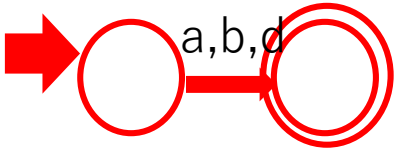
理解確認クイズ(3)つづき

((a+b)(a+b+d))*

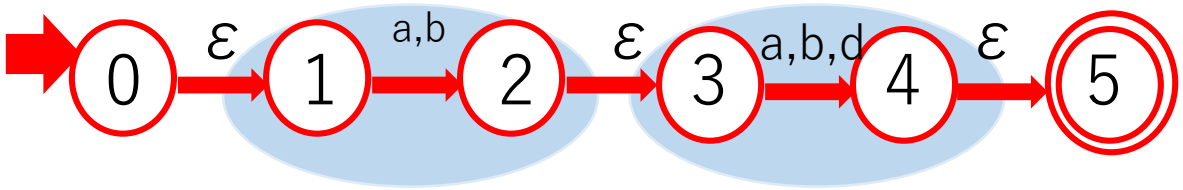
(a+b)

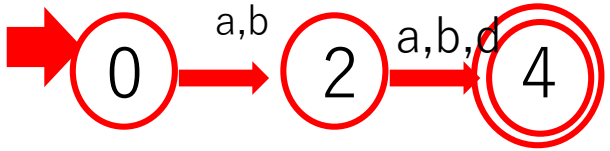


(a+b+d)



(a+b)(a+b+d)

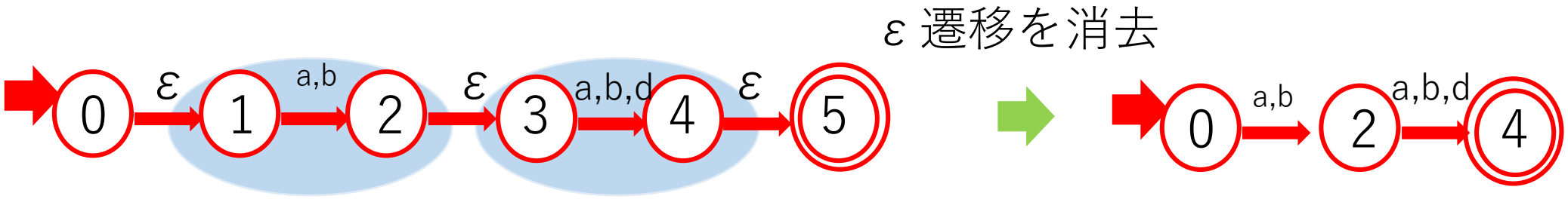




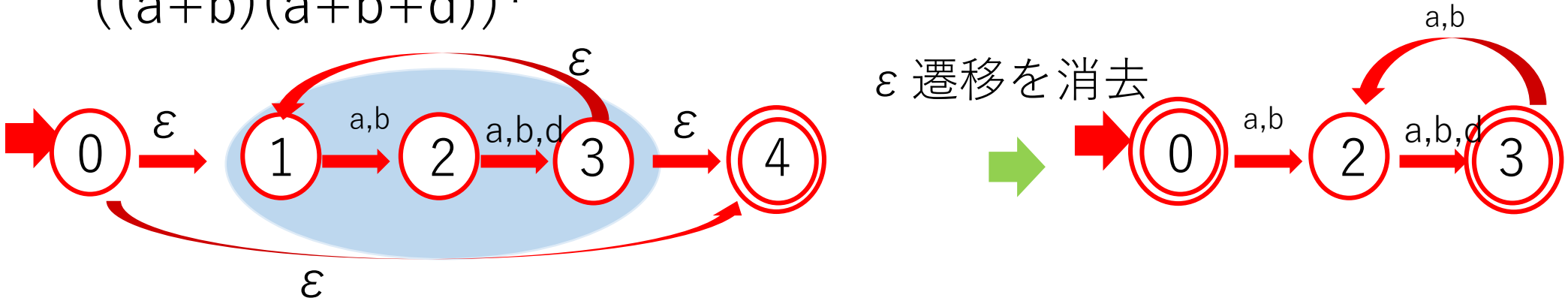
理解確認クイズ(3)つづき

((a+b)(a+b+d))*

(a+b)(a+b+d)



((a+b)(a+b+d))*



理解確認クイズ

(4) Σ ={0,1}とする。Σ 上の語で、二進数として解釈すると
値が4n, n=0,1,2,...になるものからなる言語を正規表現で表せ。

(5) Σ ={0,1}とする。Σ 上の語で、二進数として解釈すると
3の倍数になるものからなる言語を正規表現で表せ。
（ヒント
　オートマトンを設計してから正規表現に変換します。）

理解確認クイズ

(4) Σ ={0,1}とする。Σ 上の語で、二進数として解釈すると
値が4n, n=0,1,2,...になるものからなる言語を正規表現で表せ。

0	0
4	100
8	1000
12	1100
16	10000
20	10100
24	11000

(0+1)*00 + 0

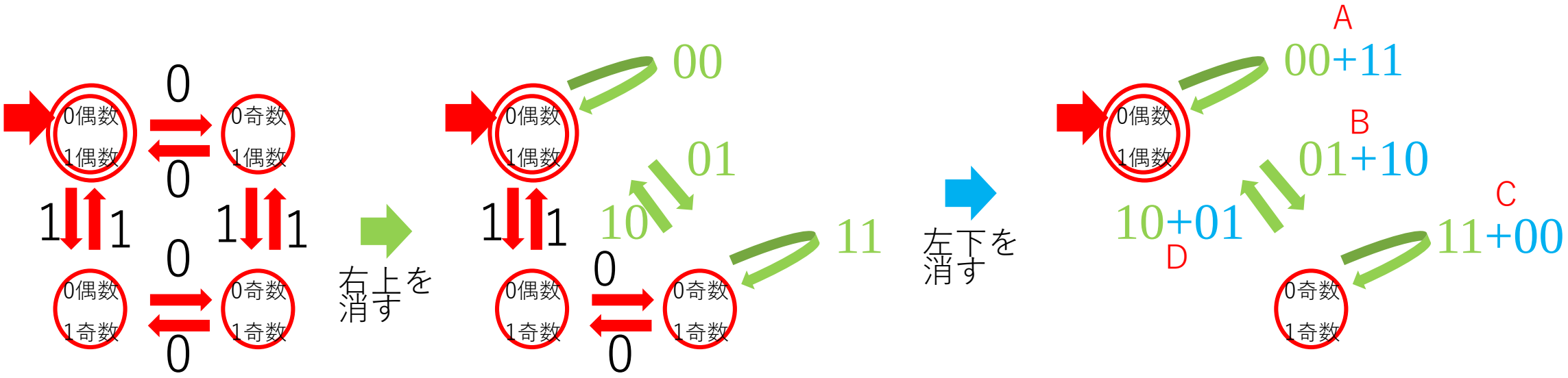
理解確認クイズ

(6) Σ ={0,1}とする。Σ 上の語で、0の個数も1の個数も偶数個であるものからなる言語を正規表現で表せ。

ヒント
まずオートマトン、それを、正規表現に変換しましょう！

理解確認クイズ

(6) Σ={0,1}とする。Σ上の語で、0の個数も1の個数も偶数個であるものからなる言語を正規表現で表せ。



(A+BC*D)*

(00+11 + (01+10) (11+00)* (10+01))*