競技プログラマ向け 形式言語理論 入門

稲葉 一浩

JOI 春合宿 2012

「形式言語理論」とは

文字列やツリーやグラフの集合

について考える分野

「形式言語理論」とは

文字列やツリーやグラフの
集合
を、どうやって表現するか
について考える分野

「形式言語理論」とは

今日は
文字列の集合だけ
扱います

文字列やツリーやグラフの

集合

を、どうやって表現するか

について考える分野

「形式言語理論」とは

文字列やツリーやグラフの

集合

を、表現するデータ構造

について考える分野

```
#include <set>
#include <string>
std::set<std::string>
```

?

「形式言語理論」とは

文字列やツリーやグラフの

無限かもしれない集合

の、有限のメモリでの表現

について考える分野

文字列の無限集合の例

* {“”, “a”, “aa”, “aaa”, “aaaa”, …}
* 「長さが偶数の文字列すべて」
* 「回文じゃない文字列」
* 「円周率の10進表記の部分列」
* ・・・

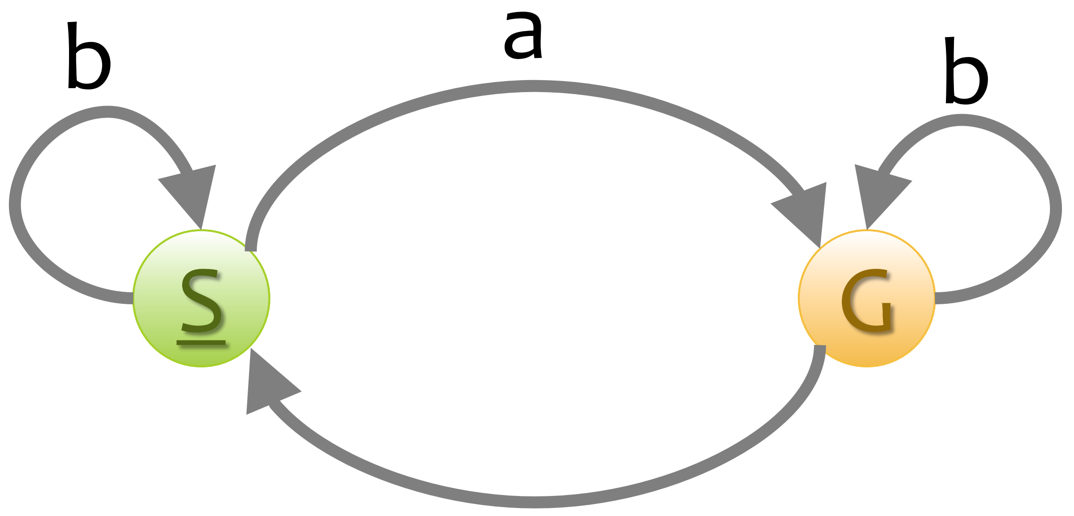
ここからの話題

* いくつかの表現方法の紹介
* どんな操作が可能か
* （コンテストっぽい）応用
* （夢の広がる）応用

文字列集合を

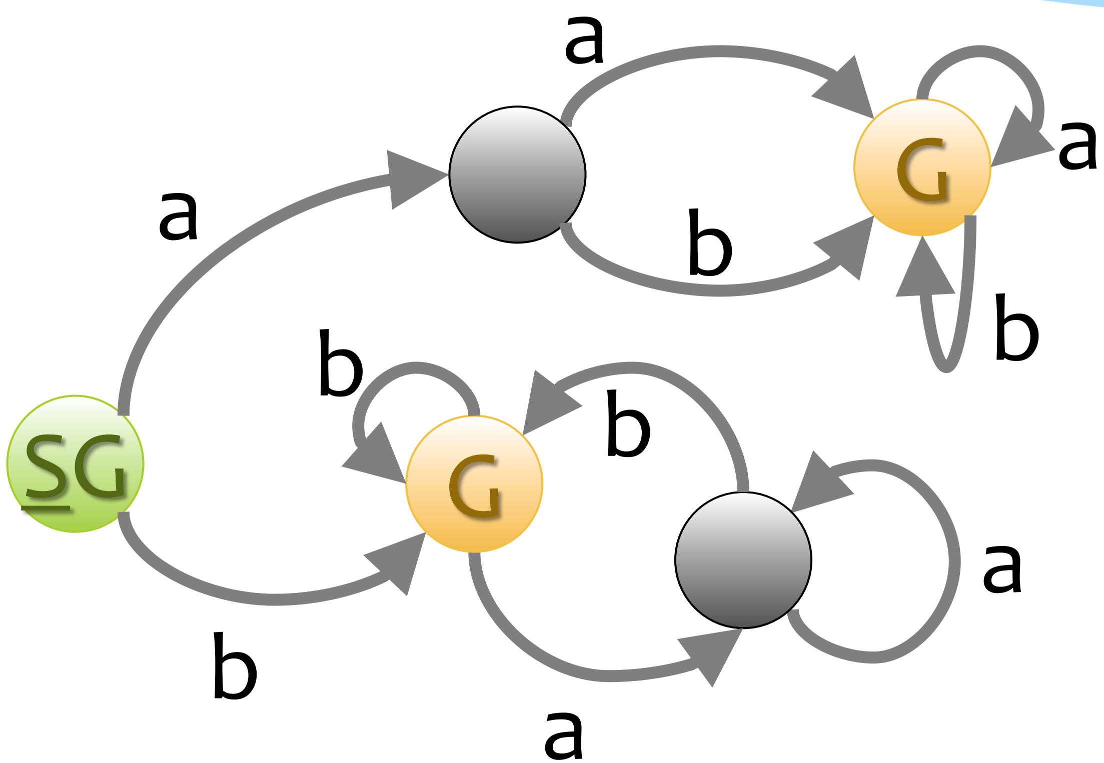
有向グラフで表現

{"a", "ba", "bba", ...}



「aが奇数個」

{“”, “aa”, “ab”, “b”, “bab”, …}

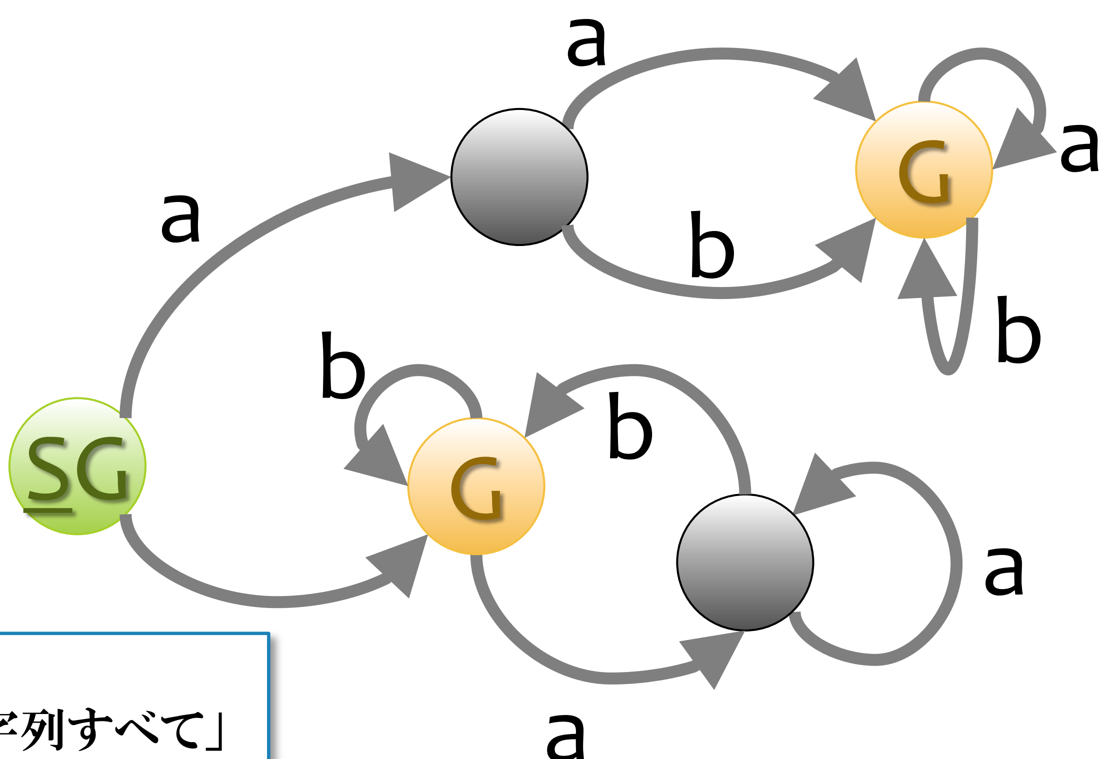


「aで始まって長さ2以上、
　またはbで始まってbで終わる」

グラフで表す文字列集合

* 辺に文字が書いてある
* 各頂点には
  * Sという印
  * Gという印
  * SとG両方

  が付いているかも



こういうグラフを、
「SからGまでの経路になってる文字列すべて」
という集合を表していると考えます。

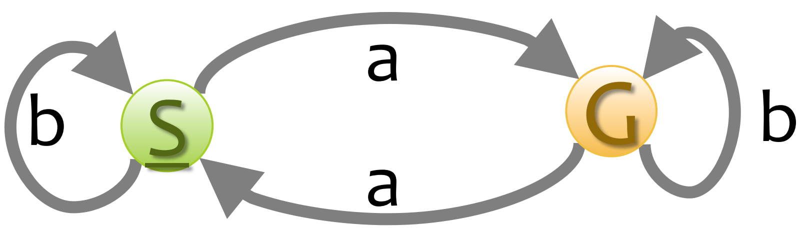
こんな集合が表せる

- 有限集合は全部書ける
  - root が S 、leaf が G の tree で表現する
- 「特定の文字列を部分に含む文字列ぜんぶ」
  - KMP法、Aho-Corasick法
- 「aとbが交互に繰り返し出てくる文字列」
  - ループっぽいものはグラフでサイクルを作ると書ける
- 書けないものもあります。
  - 「回文」「括弧の対応が取れてる文字列」

2つの流儀: DFA と NFA

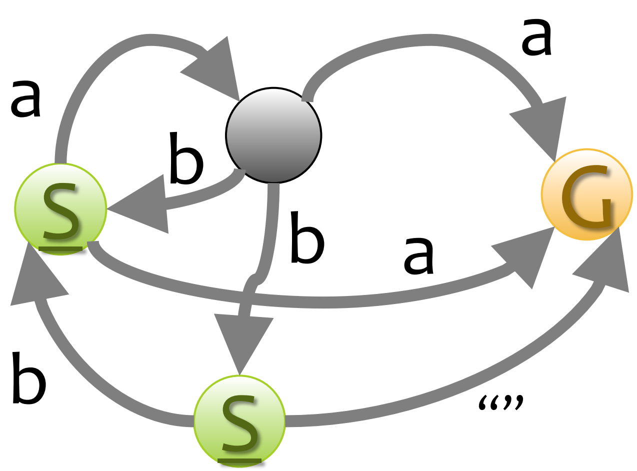
Deterministic Finite Automaton

* Sが1つ
* 一つの頂点から出る、
  同じ文字の辺は1本以下



Nondeterministic Finite Automaton

* なんでもあり
* 文字無し辺もOKとする事も

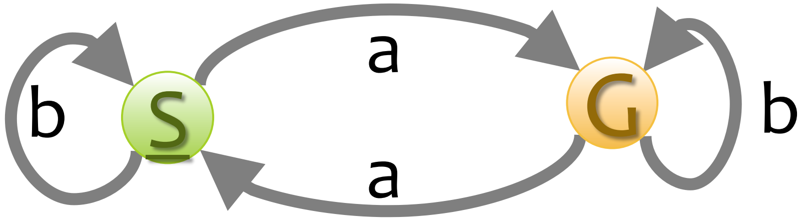


できる操作の、ごく一部
（だいたい思いつく物はなんでもできます）

* bool contains(Automaton a, String w);
  * 与えられた文字列を含むかの判定
* Automaton　complement(Automaton a);
  * 補集合の計算
* Automaton　intersect(Automaton a, Automaton b);
  * 共通部分の計算
* Automaton　equals(Automaton a);
  * 集合として等しい？

bool contains(DFA a, String w);
DFAの場合

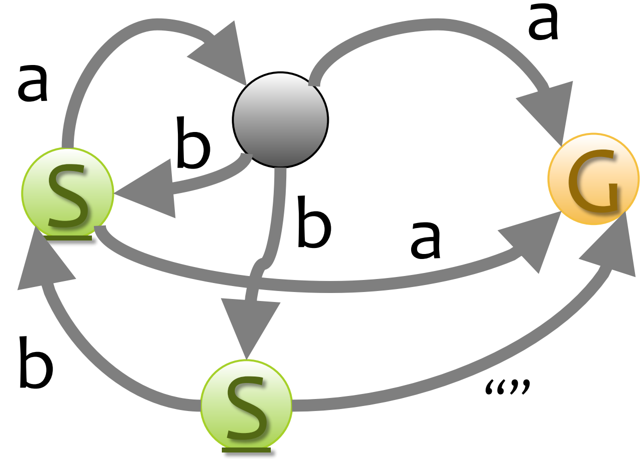
* Sが1つ
* 一つの頂点から出る、同じ文字の辺は1本以下




```
Node v = a.S; //Sは1つなので始点は1つに決まる
foreach(char c : w)
  v = a.next(v, c); //文字が決まれば辺も1つ
return v.isG();
```

bool contains(NFA a, String w); NFAの場合

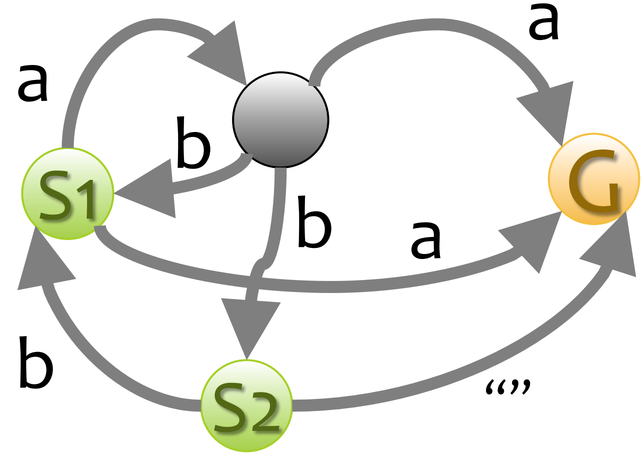
* 少しだけ難しい
* 問題:
  * S から G まで、文字列 w に合わせて動いてたどり着く道はあるか?



bool contains(NFA a, String w);
NFAの場合

* 解法1：DP
  * bool[頂点数][文字列長+1]

	“a	b	b	a”	
S1	○				
S2	○				
G					

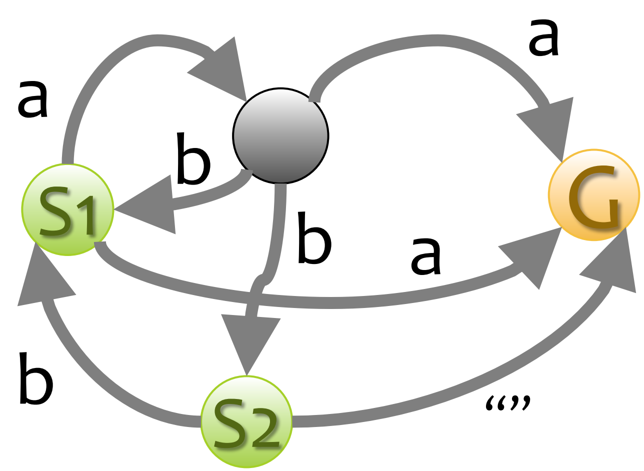


bool contains(NFA a, String w);
NFAの場合

* 解法1：DP
  * bool[頂点数][文字列長+1]

“a b b a”

S1	○				
S2	○				
	×				
G	○				

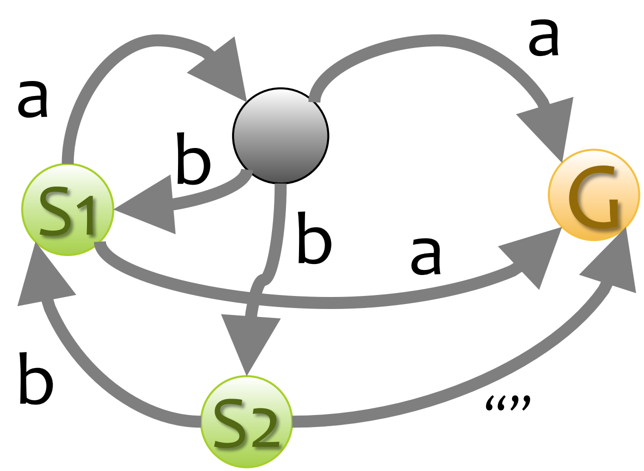


bool contains(NFA a, String w); NFAの場合

* **解法1：**DP
  * bool[頂点数][文字列長+1]

“a b b a”

S1	○	×			
S2	○	×			
	×	○			
G	○	○			

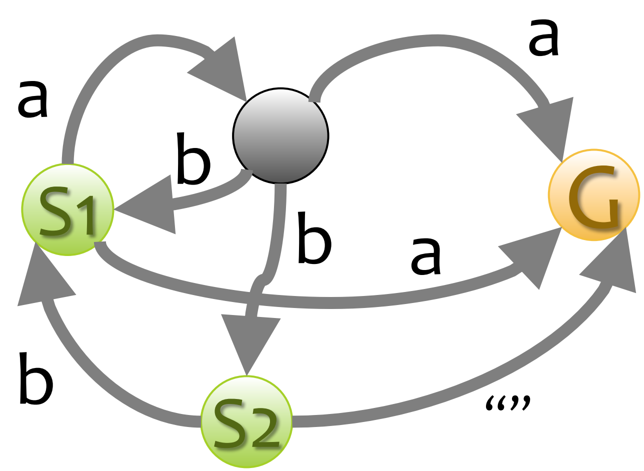


bool contains(NFA a, String w);
NFAの場合

- 解法1：DP
  - O(|edge|・|w|)

“a　　b　　b　　a”

S1	○	×	○	○	×
S2	○	×	○	×	×
	×	○	×	×	○
G	○	○	○	×	!!○!!



bool contains(NFA a, String w); NFAの場合

- 解法2：ビットDP
  - ○○×○　→ 2進法で “1101” とビットにエンコードできる
  - int [文字列長さ+1]

“a　　b　　b　　a”

S1	○	×	○	○	×
S2	○	×	○	×	×
	×	○	×	×	○
G	○	○	○	×	!!○!!
	1101	**0011**	**1101**	**1000**	**0011**

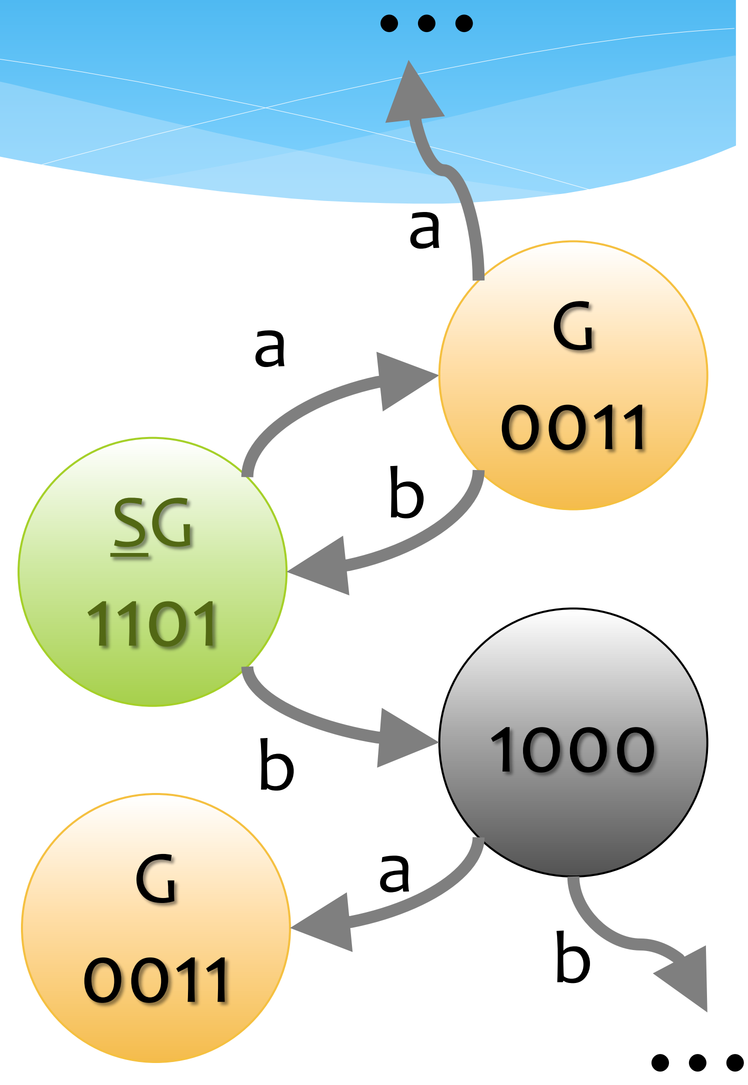
bool contains(NFA a, String w); NFAの場合

* 解法3：ビットDP & 前処理
  * $O(\ 2^{|node|} + |w|\ )$

“a　　b　　b　　a”

S1	○	×	○	○	×
S2	○	×	○	×	×
	×	○	×	×	○
G	○	○	○	×	!!○!!

1101	0011	1101	1000	0011



解法3のポイント

* NFAをDFAに変換してから処理している

* NFAの表現力　＝　DFAの表現力
  * サイズは指数でふくらみますが…

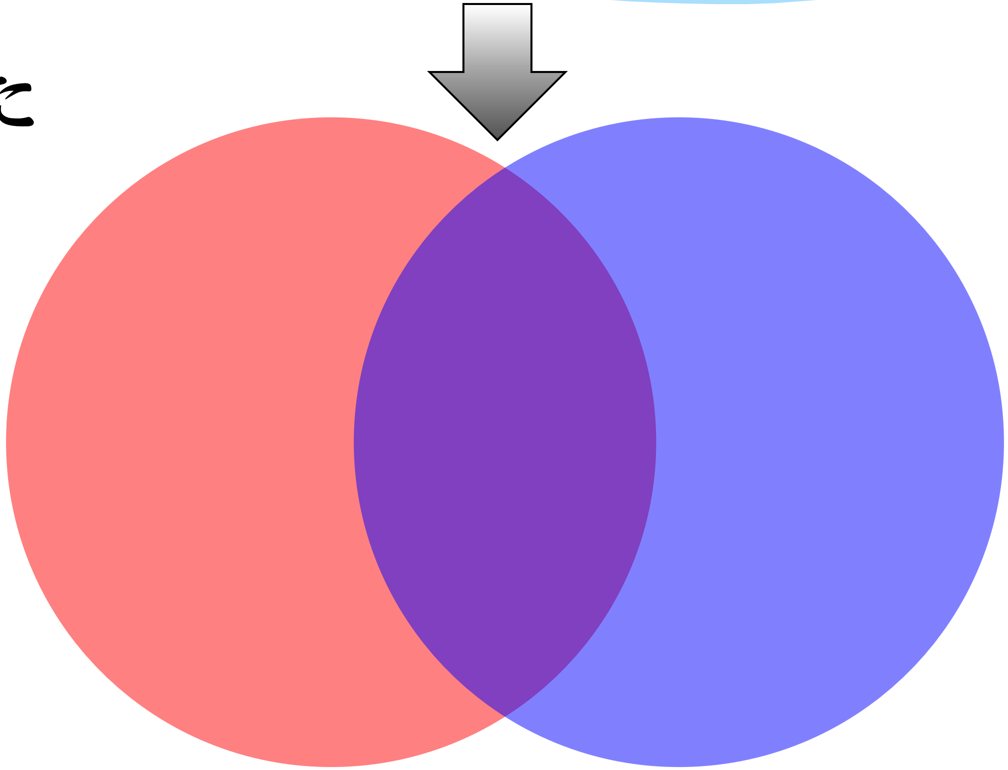
できる操作その他

* bool contains(Automaton a, String w);
  * 与えられた文字列を含むかの判定
* Automaton　complement(Automaton a);
  * 補集合の計算
* Automaton　intersect(Automaton a, Automaton b);
  * 共通部分の計算
* Automaton　equals(Automaton a);
  * 集合として等しい？

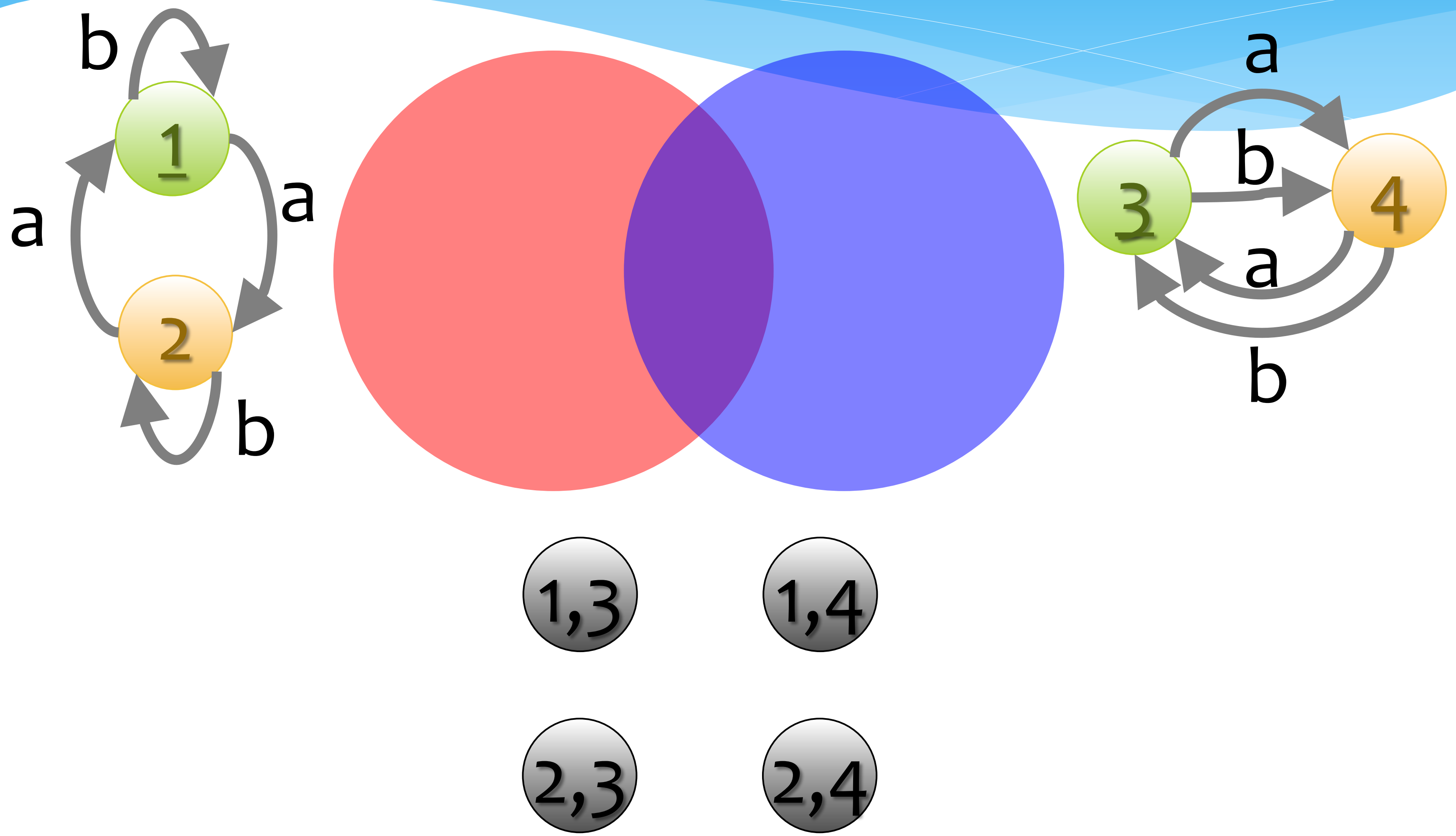
集合と集合の共通部分

Automatonで表現した
集合ａと集合ｂの
共通部分

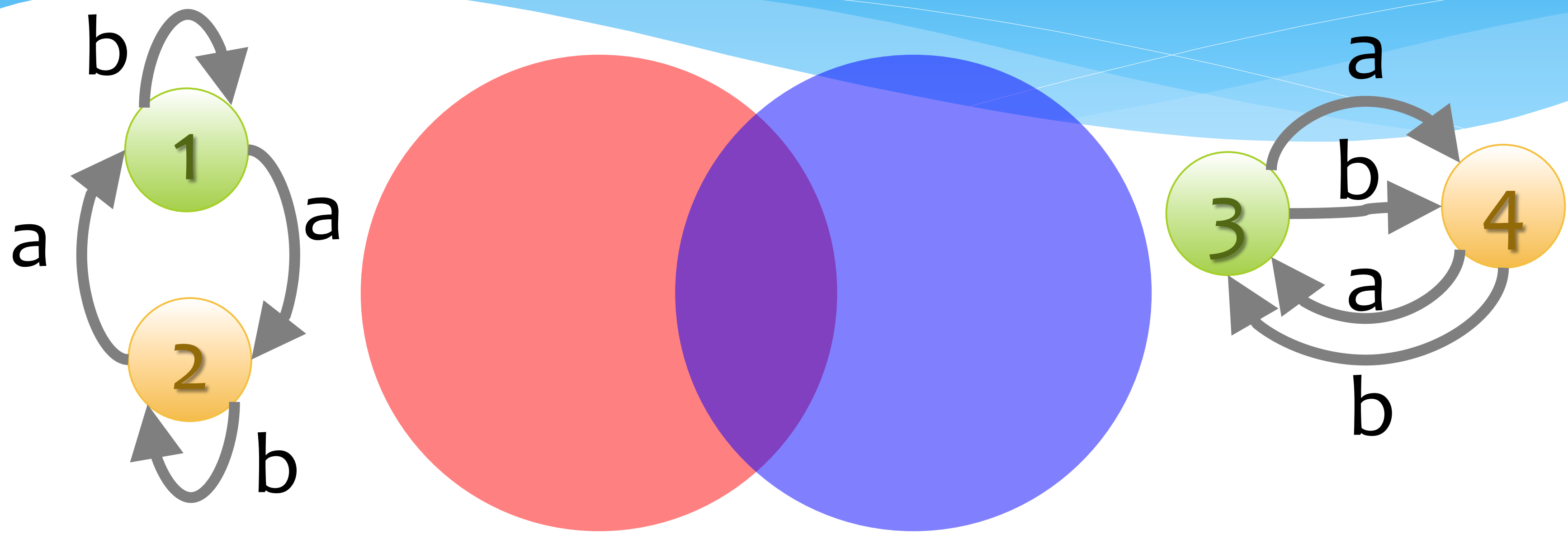
を表すAutomaton
の求め方



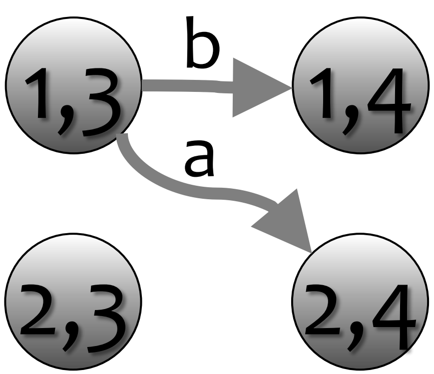
頂点と頂点のペアを作る



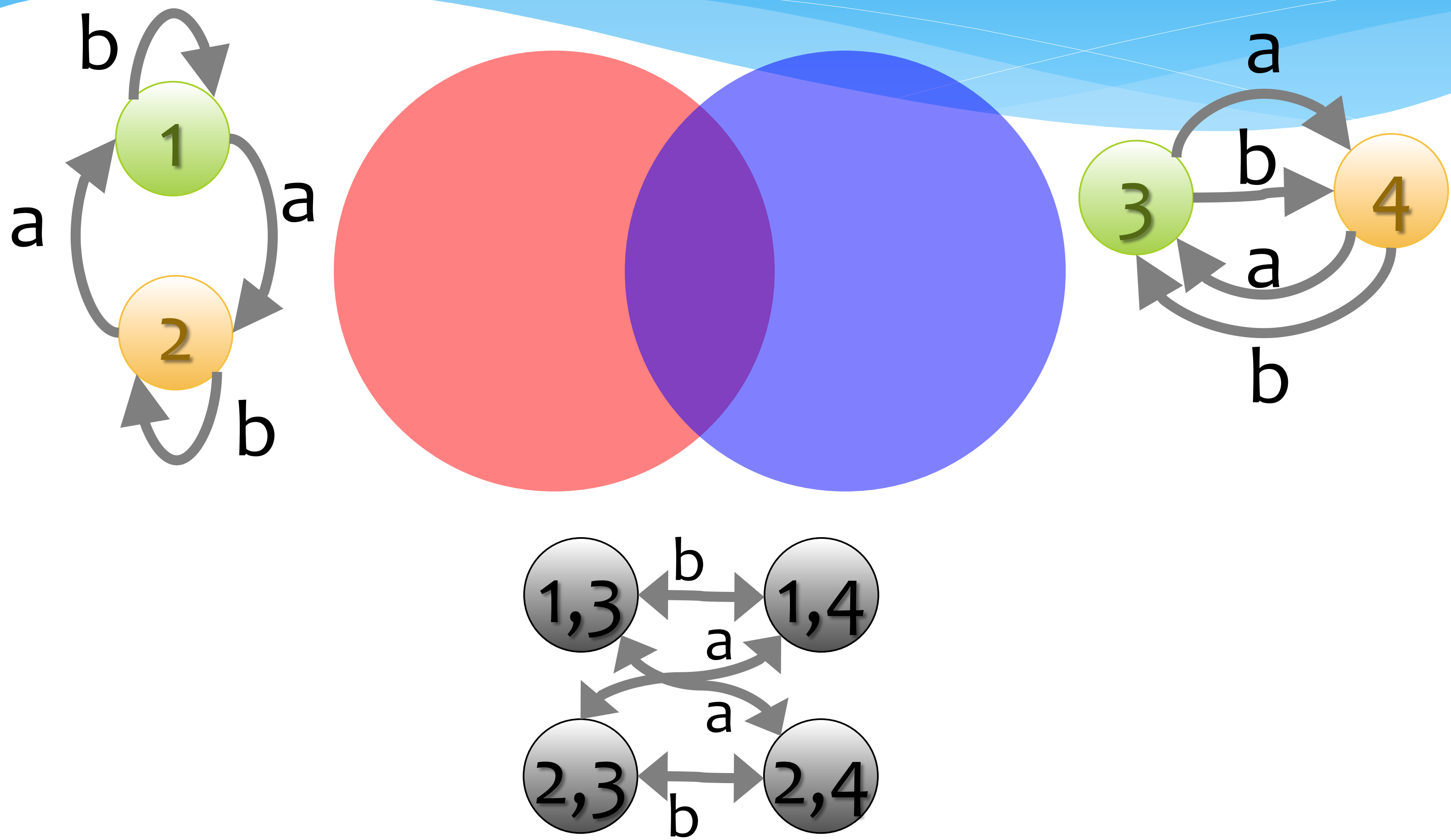
頂点と頂点のペアを作る



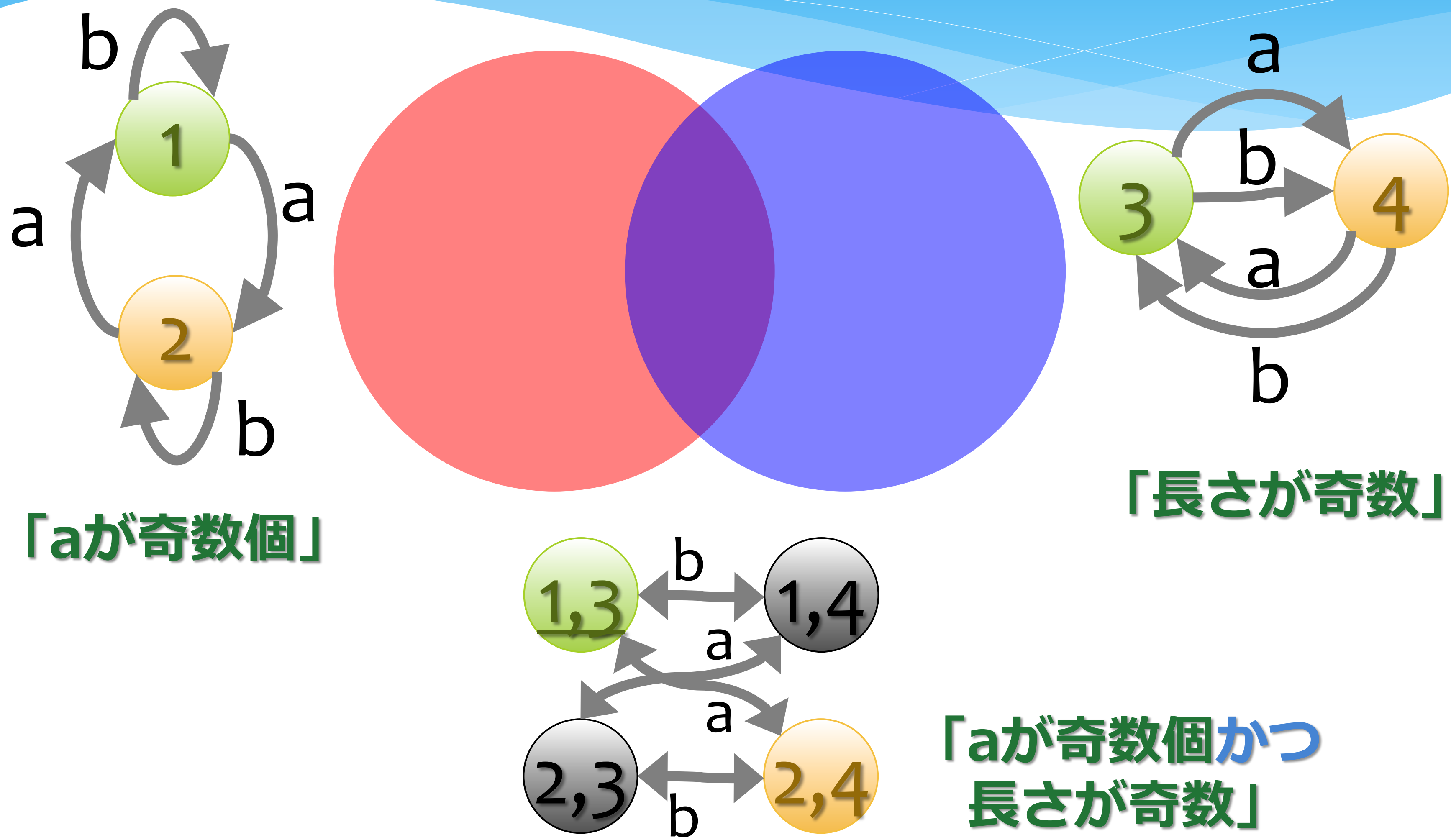
1 --- a ---> 2
3 --- a ---> 4
なので



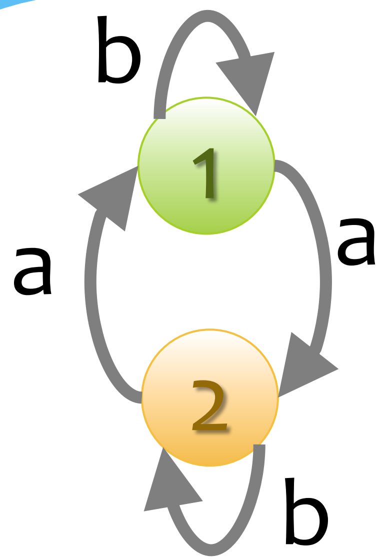
頂点と頂点のペアを作る



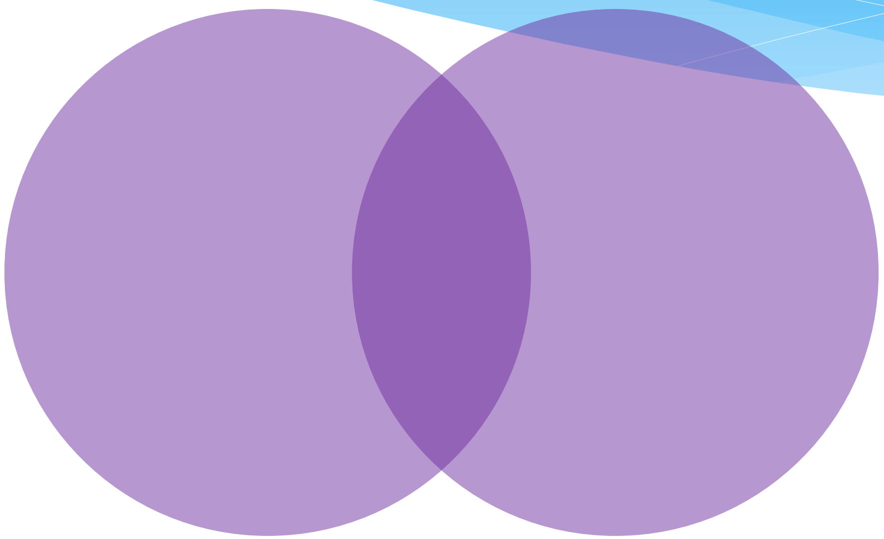
両方Sならに
両方GならGに

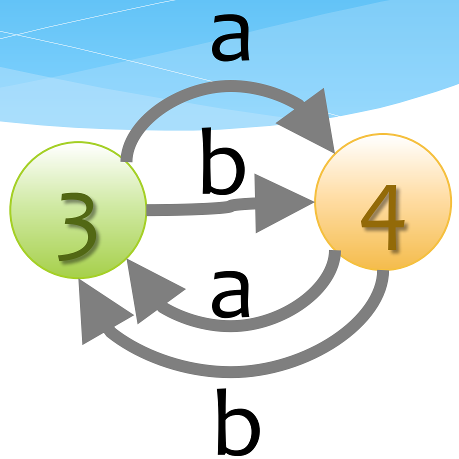


和集合
どちらかがGならGに

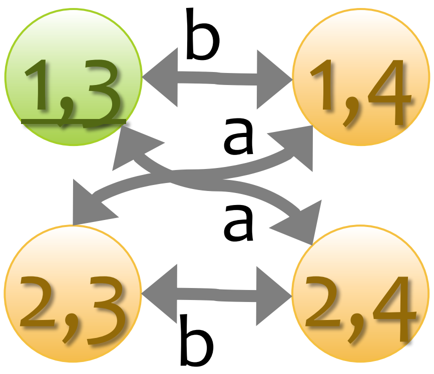


「aが奇数個」





「長さが奇数」



「aが奇数個または
長さが奇数」

できる操作その他

- 集合の補集合
  - DFA ならGとGじゃない頂点を反転するだけ
  - NFA は DFA に変換してください
- 空集合かどうか判定
  - S からGに到達可能か判定するだけ
- 集合の包含関係（A⊆B; A が完全にBに含まれているか？）
  - 「((Bの補集合)とA の共通部分)が空集合か？」と同じ
- 集合が等しいかどうか
  - A⊆B && B⊆A （もっと効率よく判定もできます）

練習問題: "Double Meaning"

1〜N の自然数を、それぞれ長さ M 以下のビット列に変換して自然数のリストを表現することにした。例えば右下の変換表を使うと

[3,1,2] が 100<u>1001</u>00 になる。

1	→	1001
2	→	00
3	→	100

ところがこの変換表は困りもので、

[3,3,3] も 100<u>100</u>100 になるし

[1,2,3] も 100<u>100</u>100 になってしまう。

変換表を受け取って、こういう困ったこと(違うリストが同じビット列になってしまう)が起こるかどうか判定せよ。(N,M≦50)

想定誤答

“100” が “1001” の prefix なのが問題。
これじゃ前から読んでってどちらか区別できない

```
1 → 1001
2 → 00
3 → 100
```

```
for(int i=0; i<code.size(); ++i)
for(int j=0; j<code.size(); ++j)
  if(i!=j && code[i].is_prefix(code[j]))
    return BAD;
return GOOD;
```

撃墜例

この変換表は一意に復元可能

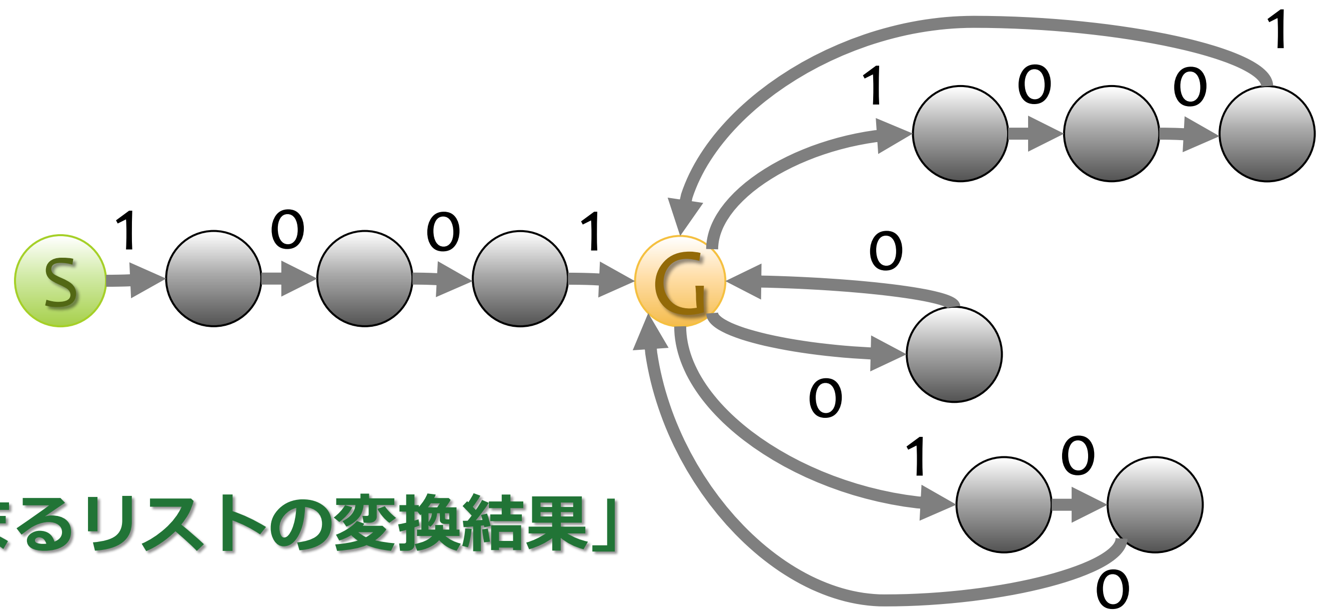
```
1 → 1
2 → 10
3 → 001
```

解答例
（もっと効率いい解法はありそう…）

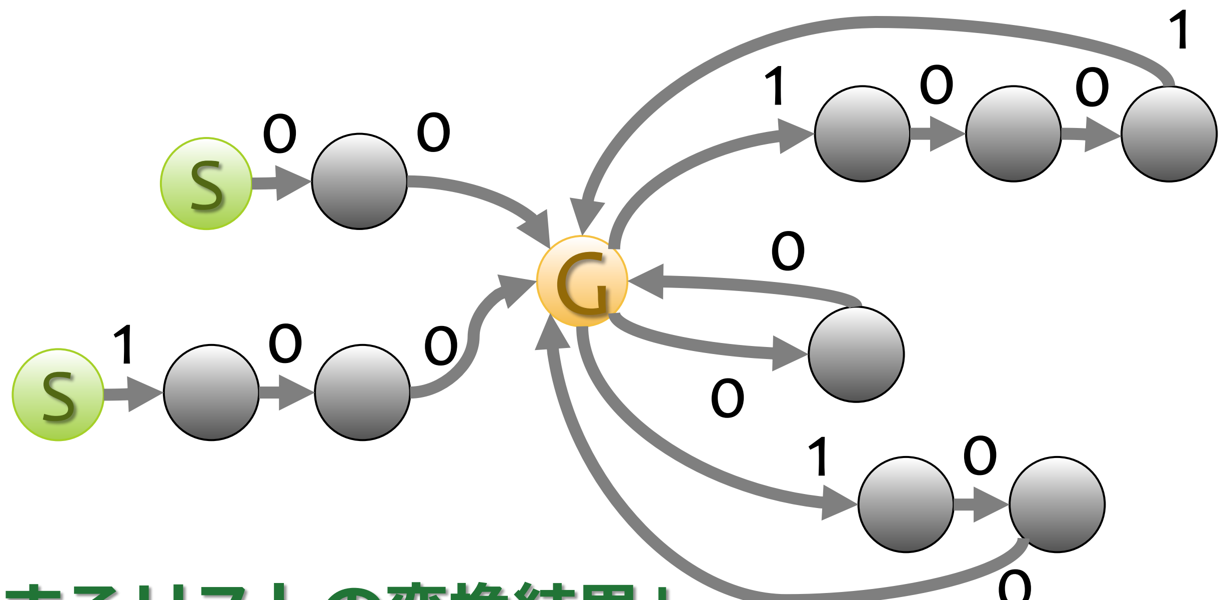
集合を使います

```
for(int i=0; i<code.size(); ++i)
 if(「iから始まるリストの変換結果の集合」
   と「i以外から始まるリストの変換結果の集合」
   の共通部分が空集合ではない )
    return BAD;
return GOOD;
```

1 → 1001
2 → 00
3 → 100



「[1,...]で始まるリストの変換結果」

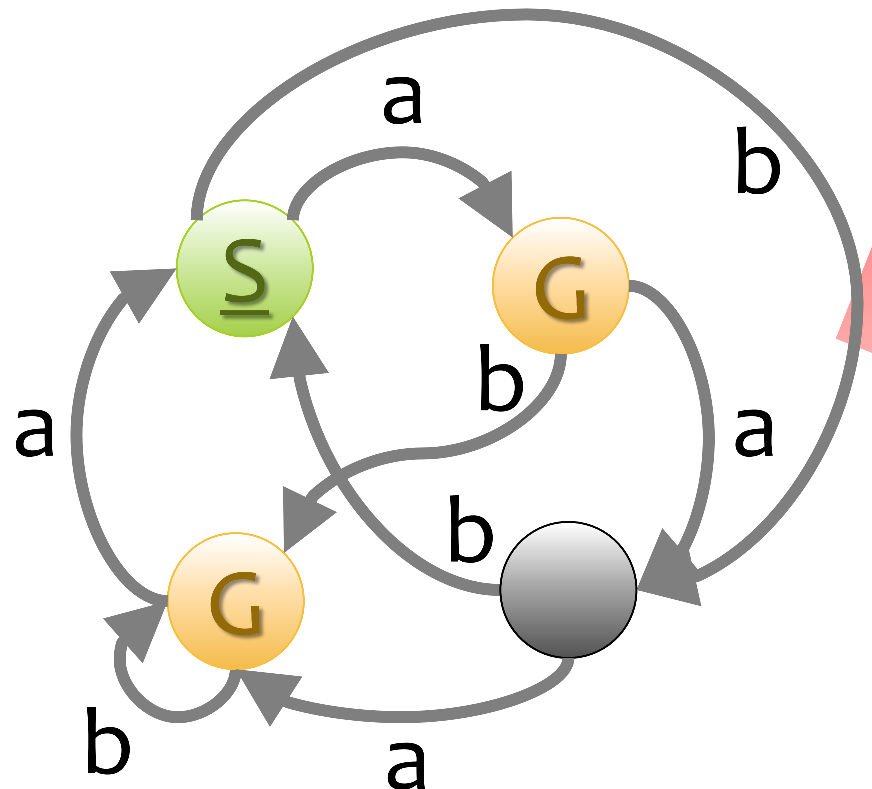


「[2,...]か[3,...]で始まるリストの変換結果」

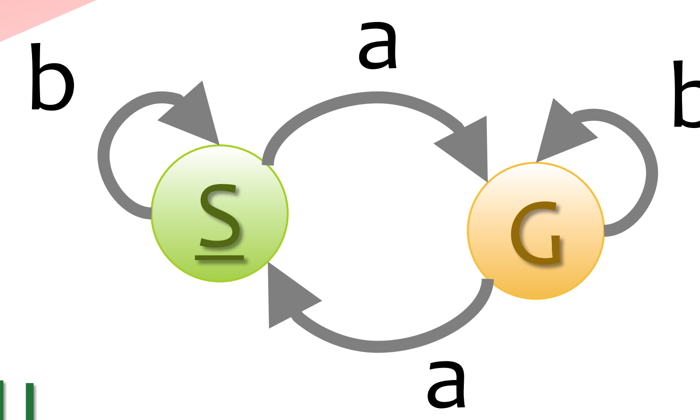
できる操作その他

- 集合の補集合
  - DFA なら G と Gじゃない頂点を反転するだけ
  - NFA は DFA に変換してください
- 空集合かどうか判定
  - S から G に到達可能か判定するだけ
- 集合の包含関係（A⊆B; A が完全にBに含まれているか？）
  - 「((Bの補集合)とＡの共通部分)が空集合か？」と同じ
- **集合が等しいかどうか**
  - A⊆B && B⊆A　**（もっと効率よく判定もできます）**

その他にできること：
DFAの最小化



実は
表してる集合は
同じ！

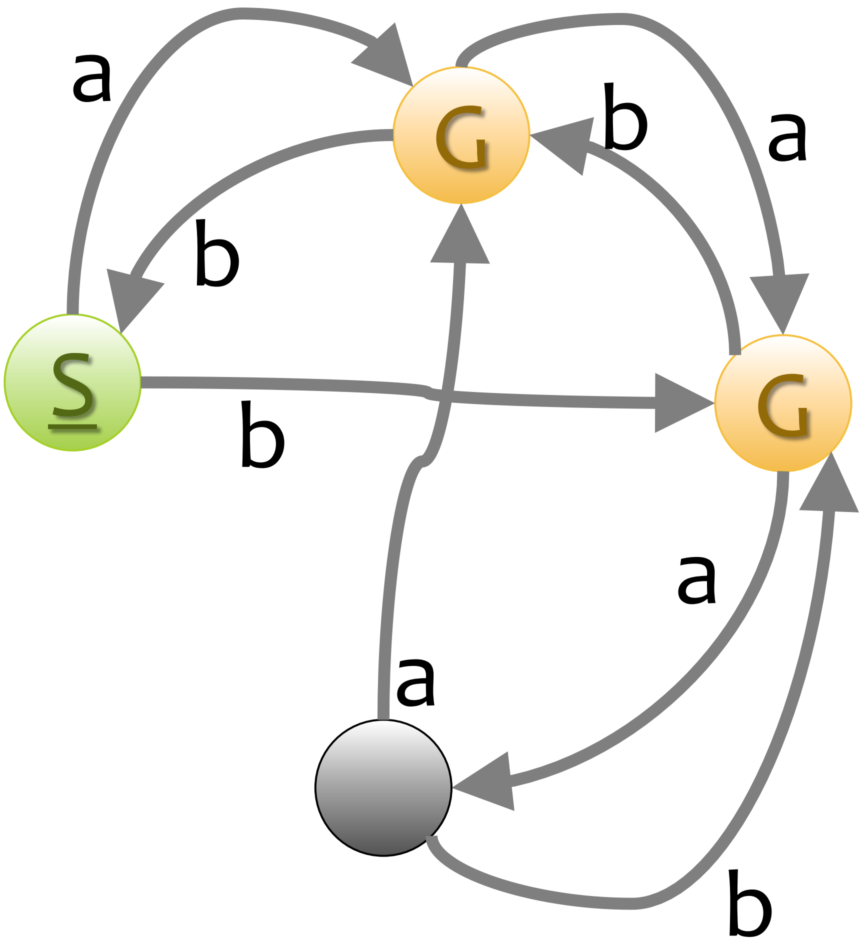


「aが奇数個」

DFA最小化の嬉しいところ

* O(|edge| log |node|) でできる。
* 最小化するとグラフの形が一つに定まる。
  * 集合の ＝ の判定が簡単
* 共通部分や和集合やNFA→DFA変換は無駄に大きいDFAを作ることが多いので、小さくするのが実用上は必須。

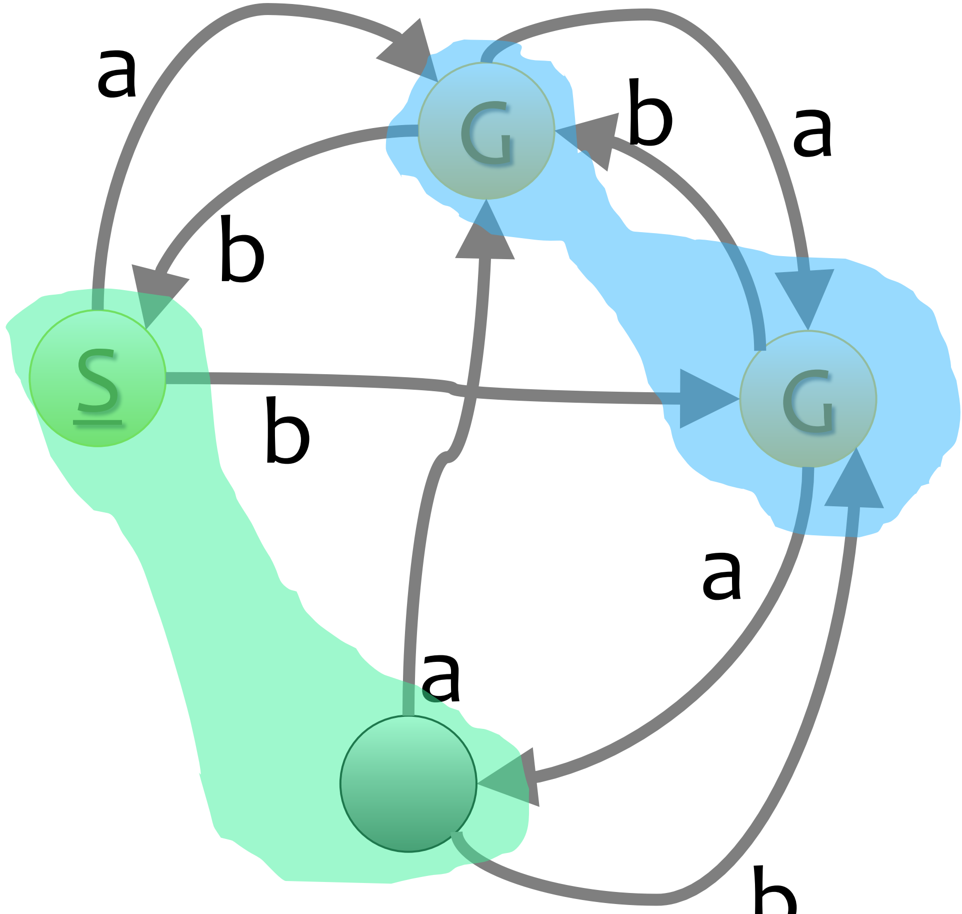
最小化のアルゴリズム



方針：

どんな文字列 w についても、
「頂点 v から w に沿って進んだ先が G」
　if and only if
「頂点 u から w に沿って進んだ先が G」
なら、v と u はマージする

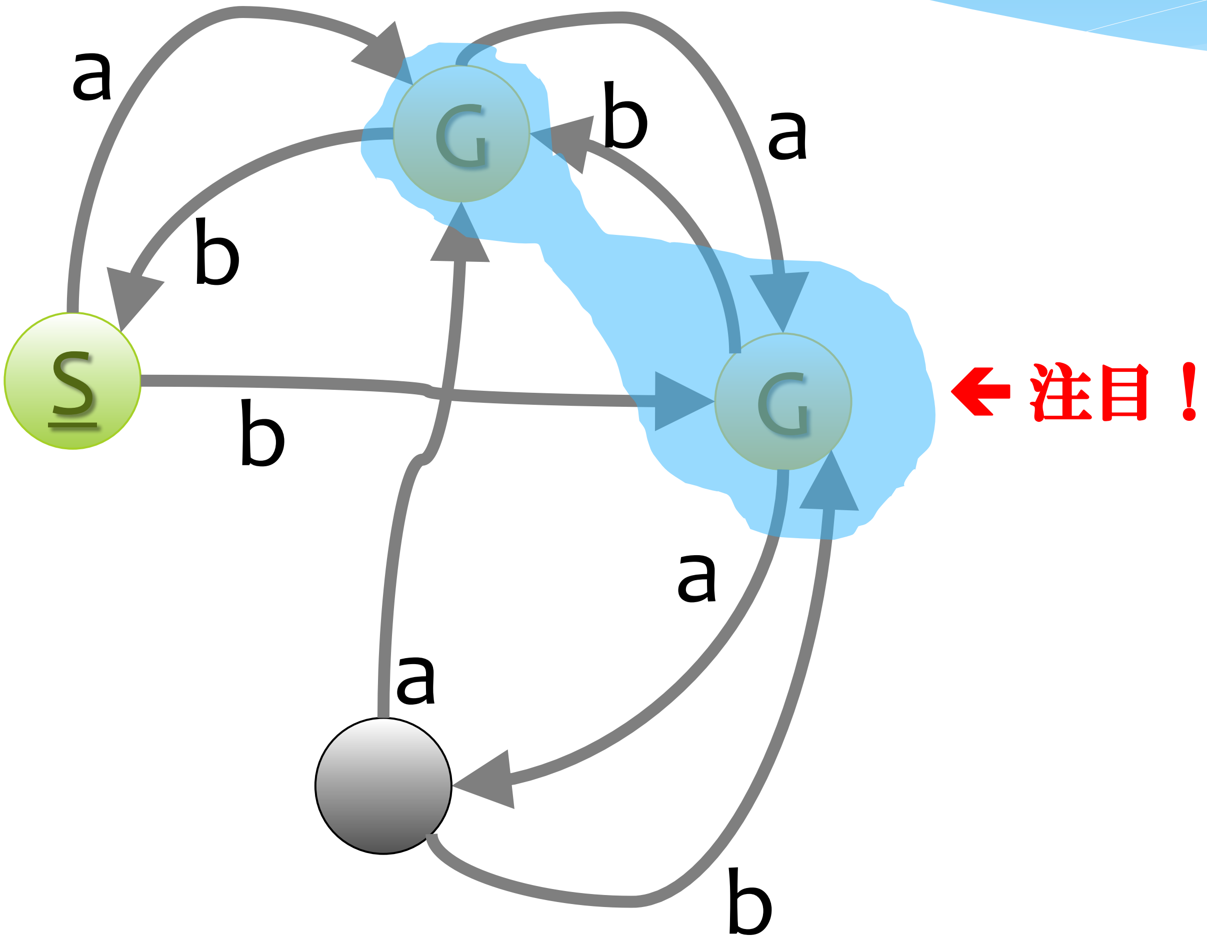
最小化のアルゴリズム



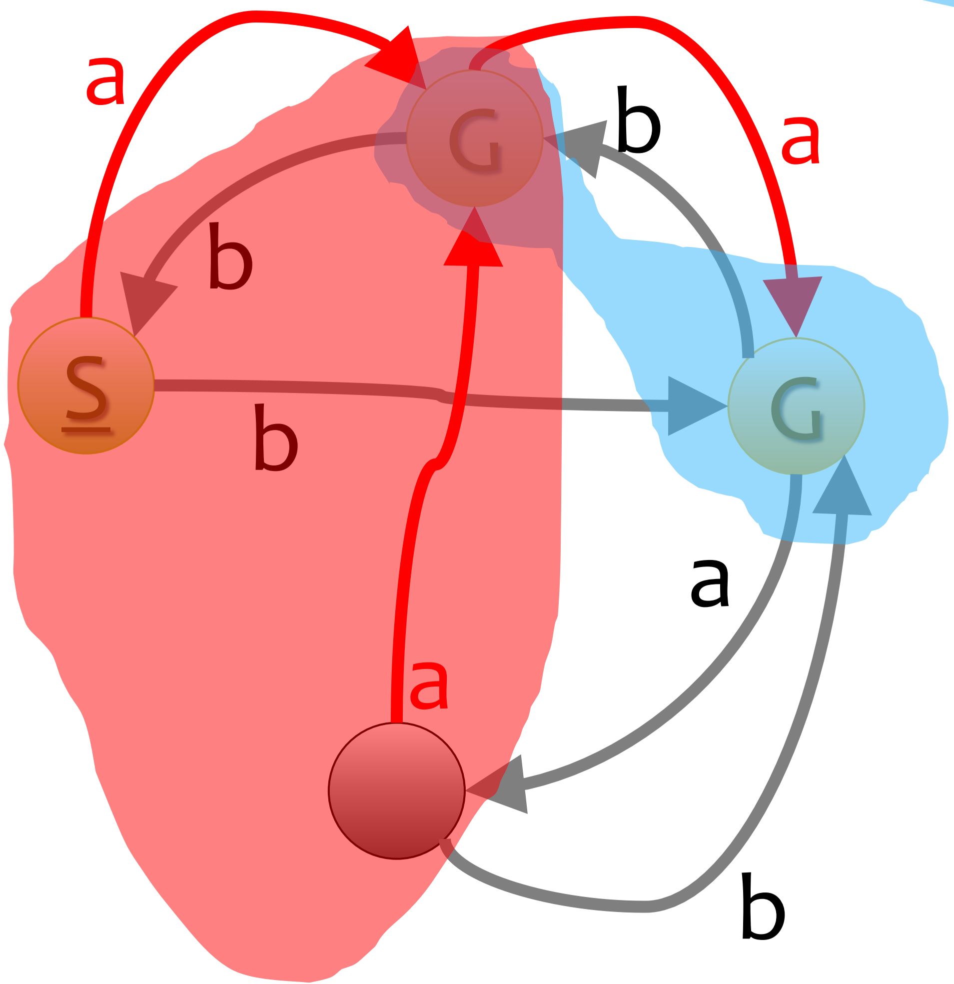
「G な頂点」
と
「G じゃない頂点」

はマージできないので
別グループ

最小化のアルゴリズム



最小化のアルゴリズム

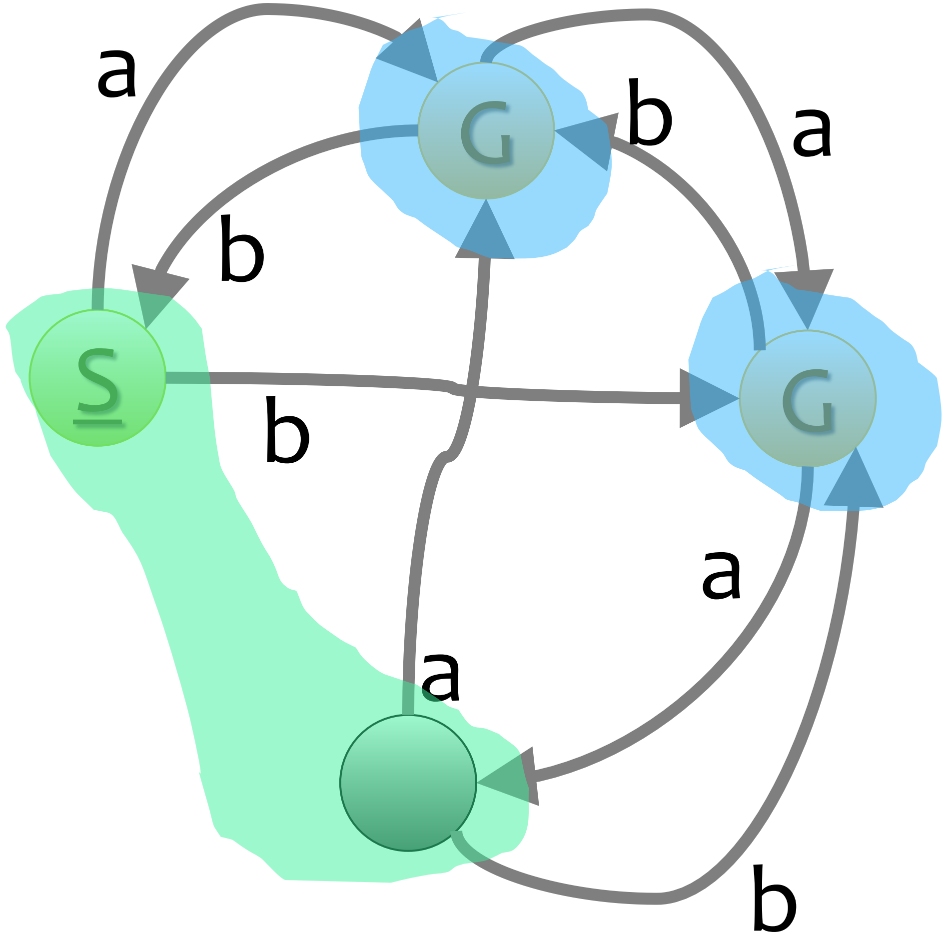


文字 'a' で進んだら

に入る頂点たちと

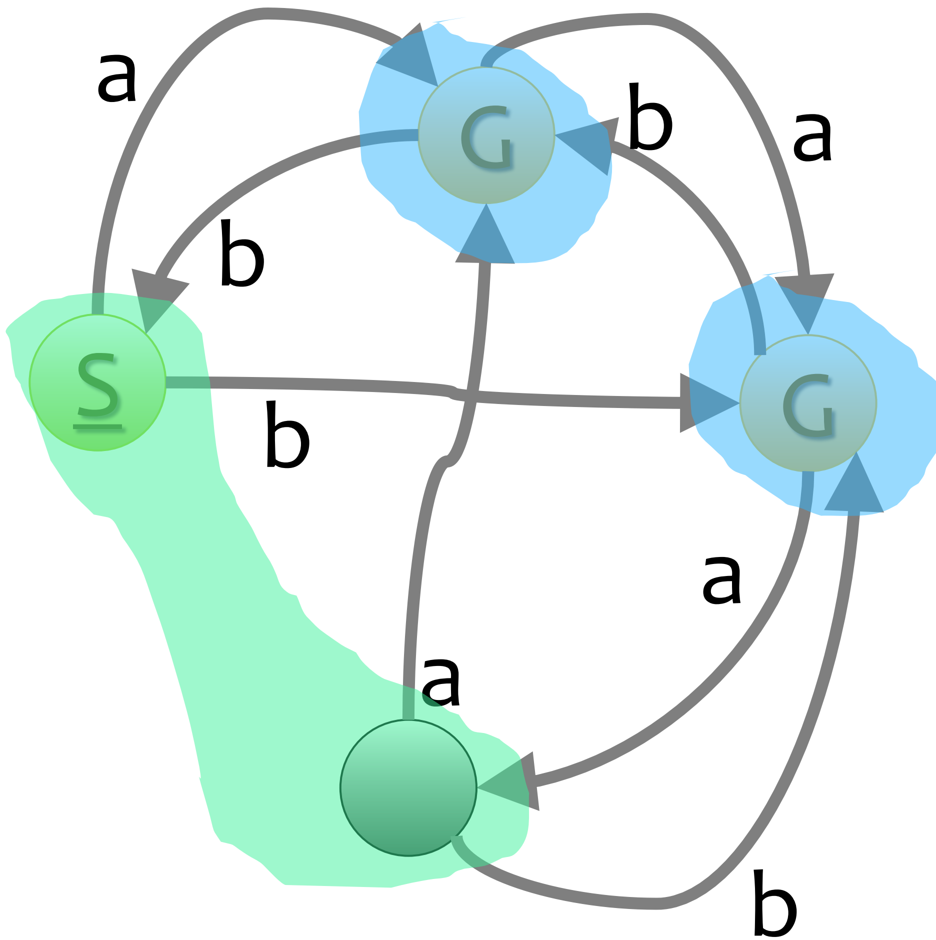
そうじゃない頂点は
マージ不可能

最小化のアルゴリズム



よって分離。

最小化のアルゴリズム

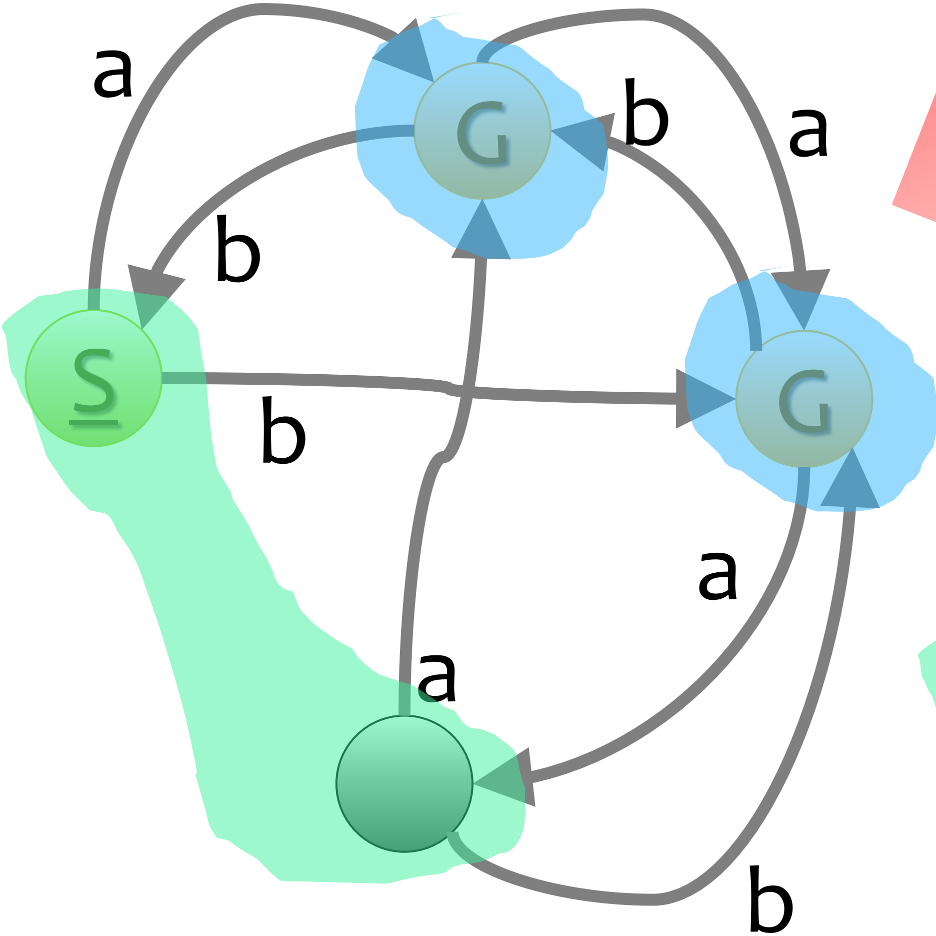


文字 ‘b’ でも
同じことを繰り返す。

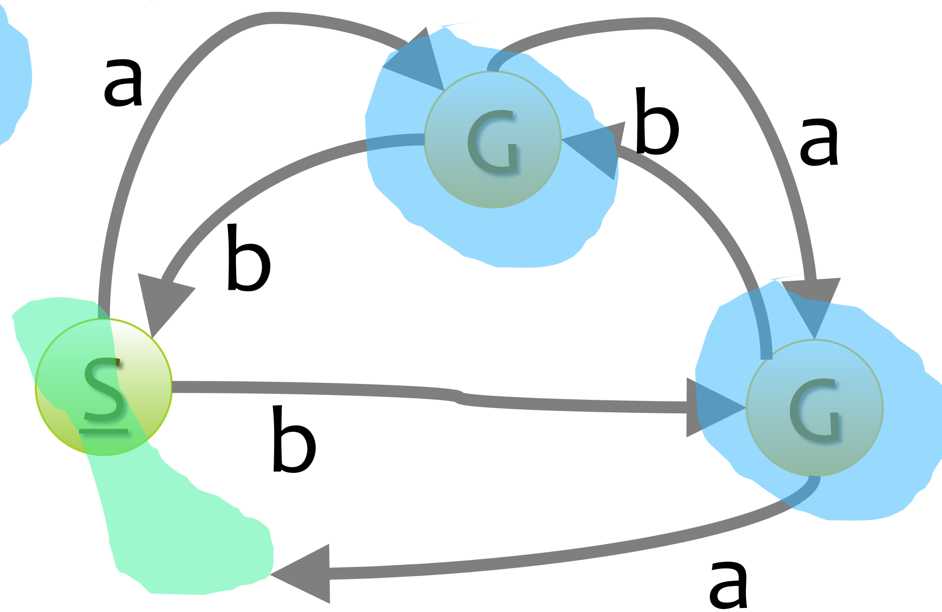
分割が起きた場合は、
分かれてできたグループ

を両方（元  を使用済
みの場合は小さい方）を
使い同様に分割を繰返す

最小化のアルゴリズム



収束したら終了
マージする。

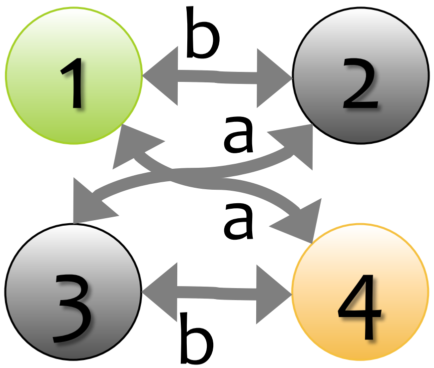


最小化のアルゴリズム

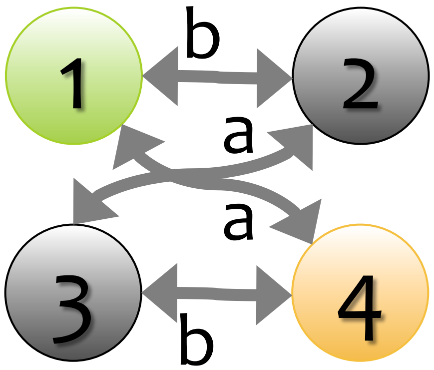
* 最初の分割
  * 0文字で G と Gじゃないところに分かれる頂点を分離
* 次の分割
  * 1文字で G と Gじゃないところに分かれる頂点を分離
* … というループなので、収束するまでやれば正しい
* を分割に使っていれば は小さい方だけ 使う、というところだけ工夫

その他にできること:
セグメント木に載せる

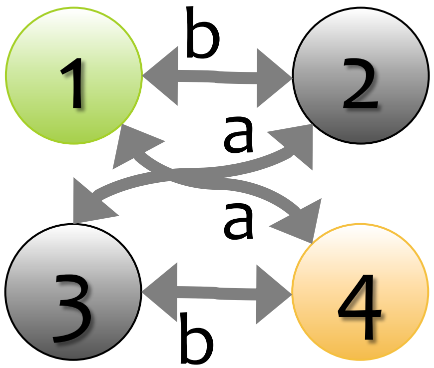
* 問題
  * DFA a（頂点数≦20）
  * 文字列 w（長さ≦10万）
  * 0≦ i ＜ k ≦ |w| な自然数の組が1万個
  * w[i, k) が DFA の表す集合に入っているか
    それぞれ判定せよ

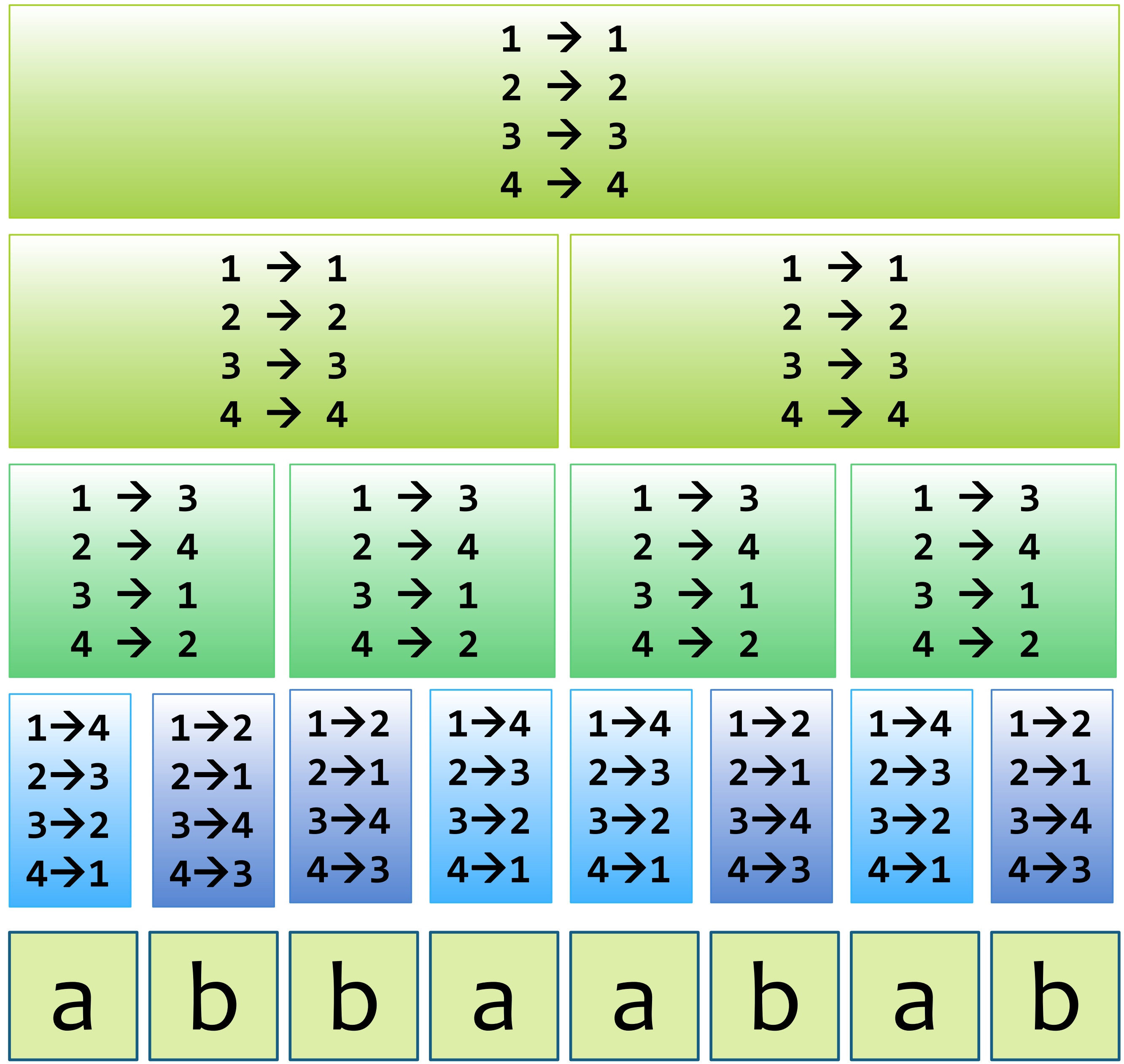


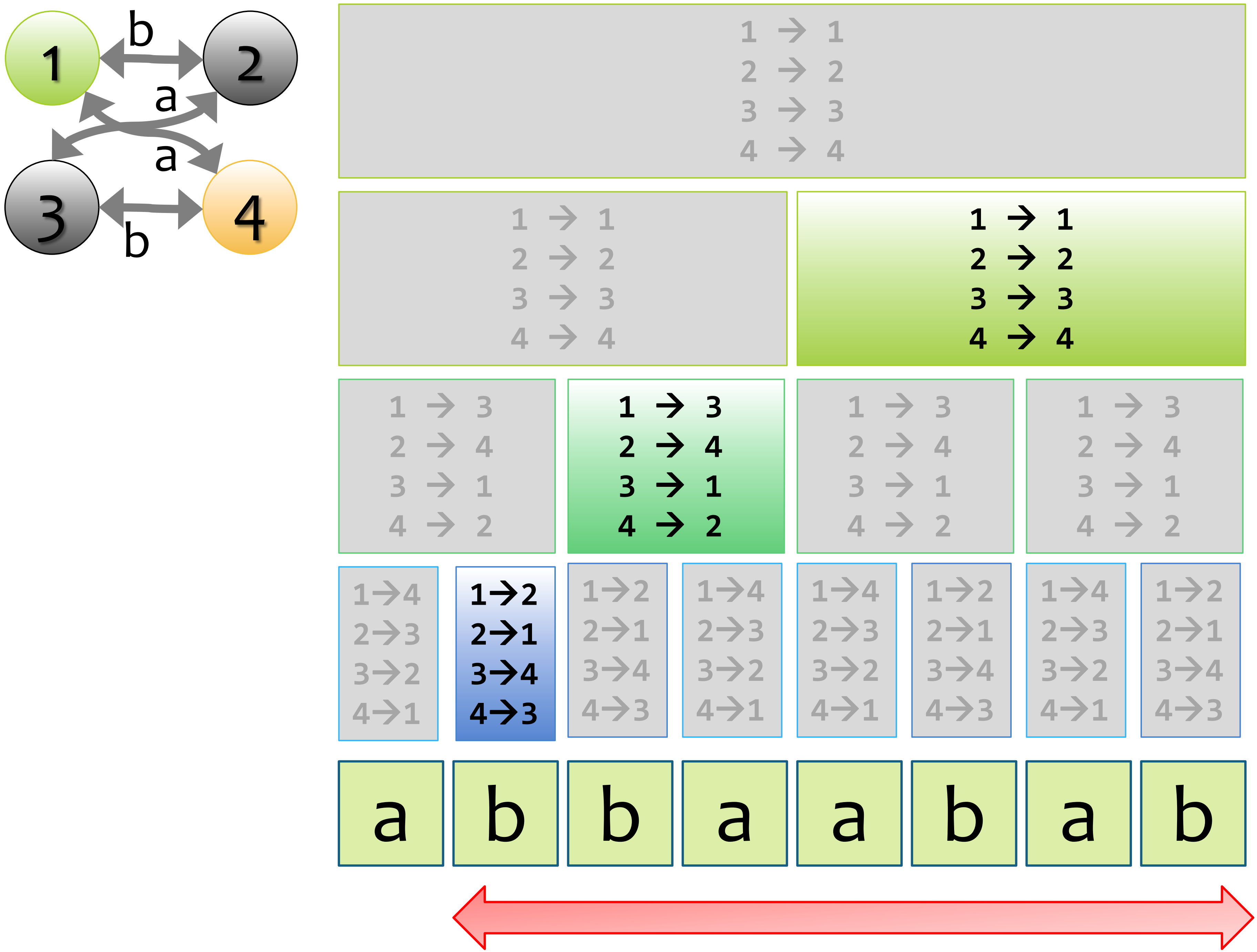
1→4 2→3 3→2 4→1	1→2 2→1 3→4 4→3	1→2 2→1 3→4 4→3	1→4 2→3 3→2 4→1	1→4 2→3 3→2 4→1	1→2 2→1 3→4 4→3	1→4 2→3 3→2 4→1	1→2 2→1 3→4 4→3
a	b	b	a	a	b	a	b



<table>
<tr><td colspan="2">1 → 3
2 → 4
3 → 1
4 → 2</td><td colspan="2">1 → 3
2 → 4
3 → 1
4 → 2</td><td colspan="2">1 → 3
2 → 4
3 → 1
4 → 2</td><td colspan="2">1 → 3
2 → 4
3 → 1
4 → 2</td></tr>
<tr><td>1→4
2→3
3→2
4→1</td><td>1→2
2→1
3→4
4→3</td><td>1→2
2→1
3→4
4→3</td><td>1→4
2→3
3→2
4→1</td><td>1→4
2→3
3→2
4→1</td><td>1→2
2→1
3→4
4→3</td><td>1→4
2→3
3→2
4→1</td><td>1→2
2→1
3→4
4→3</td></tr>
<tr><td>a</td><td>b</td><td>b</td><td>a</td><td>a</td><td>b</td><td>a</td><td>b</td></tr>
</table>


b
1
2
a
a
3
4
b


1 → 1 2 → 2 3 → 3 4 → 4
1 → 1 2 → 2 3 → 3 4 → 4
1 → 1 2 → 2 3 → 3 4 → 4
1 → 3 2 → 4 3 → 1 4 → 2
1 → 3 2 → 4 3 → 1 4 → 2
1 → 3 2 → 4 3 → 1 4 → 2
1 → 3 2 → 4 3 → 1 4 → 2
1→4 2→3 3→2 4→1
1→2 2→1 3→4 4→3
1→2 2→1 3→4 4→3
1→4 2→3 3→2 4→1
1→4 2→3 3→2 4→1
1→2 2→1 3→4 4→3
1→4 2→3 3→2 4→1
1→2 2→1 3→4 4→3
a
b
b
a
a
b
a
b



1
b
2
a
a
3
b
4
1 → 1
2 → 2
3 → 3
4 → 4
1 → 1
2 → 2
3 → 3
4 → 4
1 → 1
2 → 2
3 → 3
4 → 4
1 → 3
2 → 4
3 → 1
4 → 2
1 → 3
2 → 4
3 → 1
4 → 2
1 → 3
2 → 4
3 → 1
4 → 2
1 → 3
2 → 4
3 → 1
4 → 2
1→4
2→3
3→2
4→1
1→2
2→1
3→4
4→3
1→2
2→1
3→4
4→3
1→4
2→3
3→2
4→1
1→4
2→3
3→2
4→1
1→2
2→1
3→4
4→3
1→4
2→3
3→2
4→1
1→2
2→1
3→4
4→3
a
b
b
a
a
b
a
b

※詳しくはWebで！

DFA・NFA の場合、Segment Tree の替わりに

Factorization Forest

というテクニックで
同じ機能のものが定数高さの木でできます。

練習問題

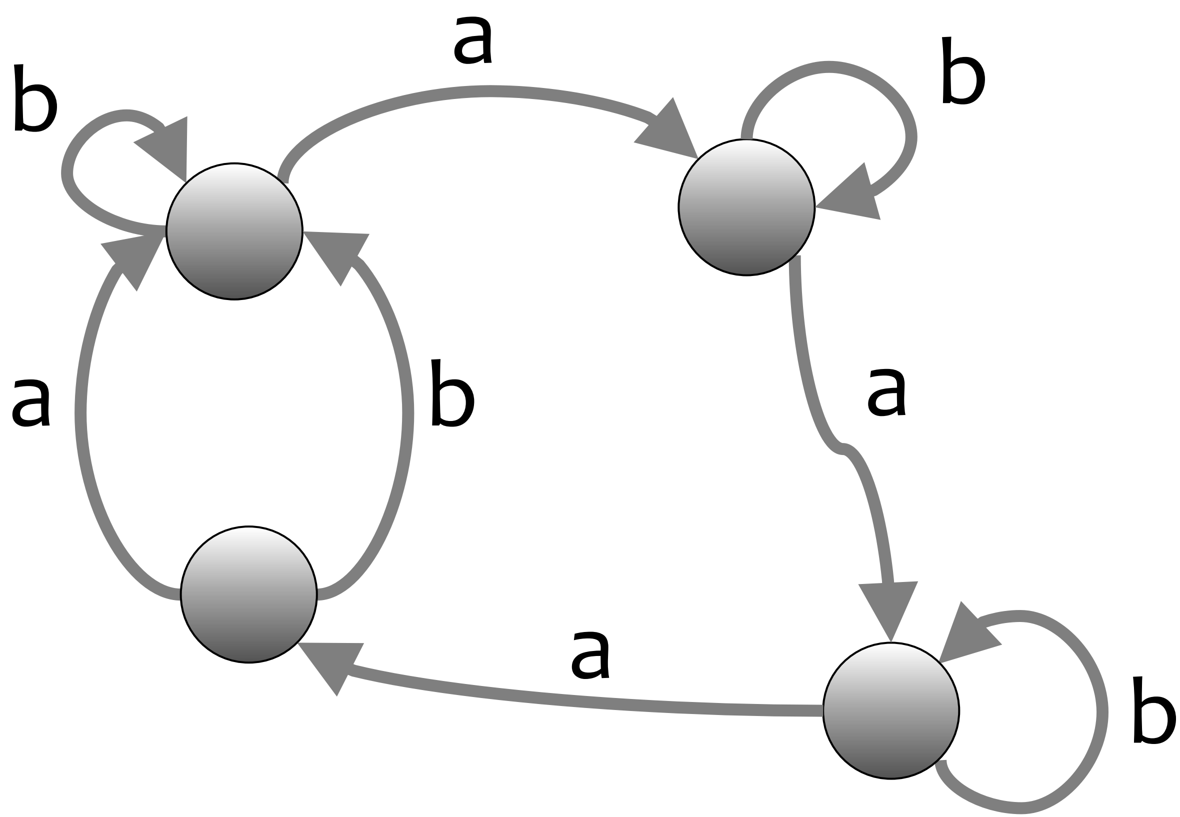
AOJ 2017

AOJ 1169

Topcoder SRM378 Div1 Hard

Topcoder TCO'09 Semifinal Medium

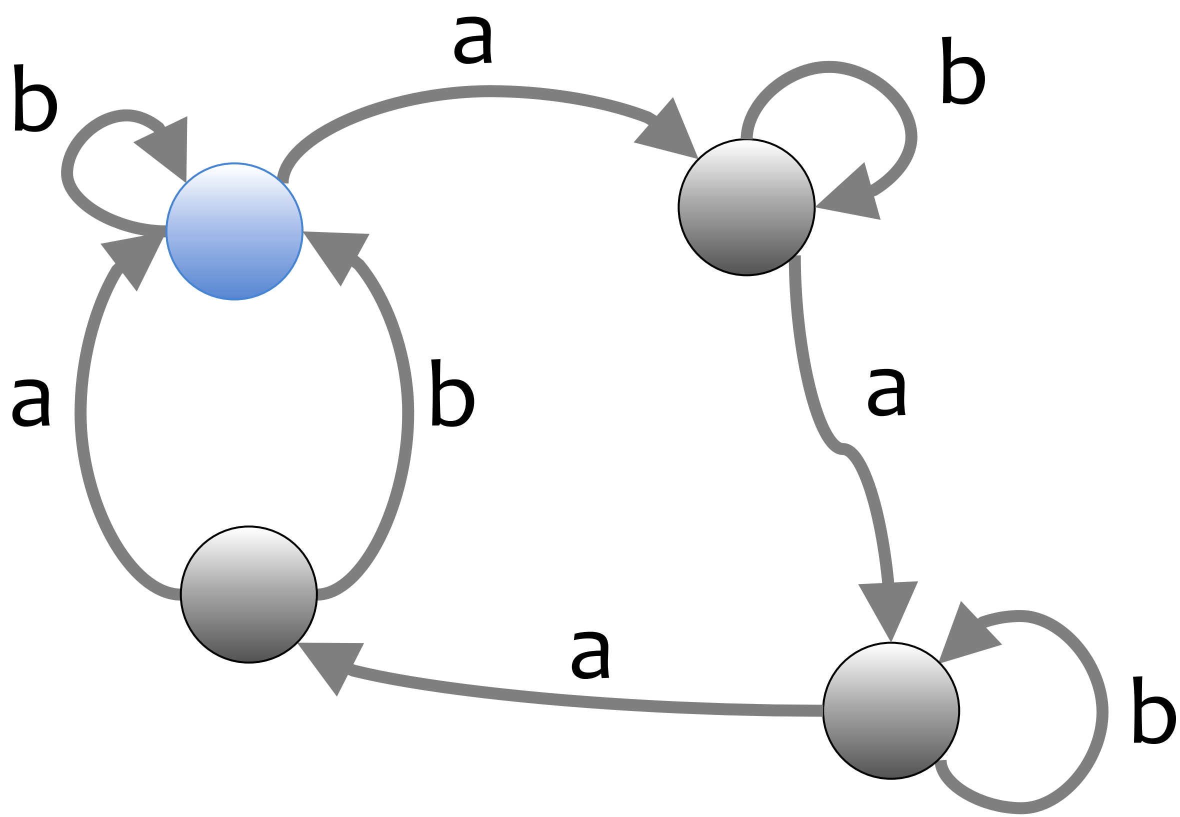
おまけ: 形式言語理論の未解決問題
Cerný 予想



DFAを
考える

どの頂点に
いても
baaabaaab
と歩くと…

Cerný 予想



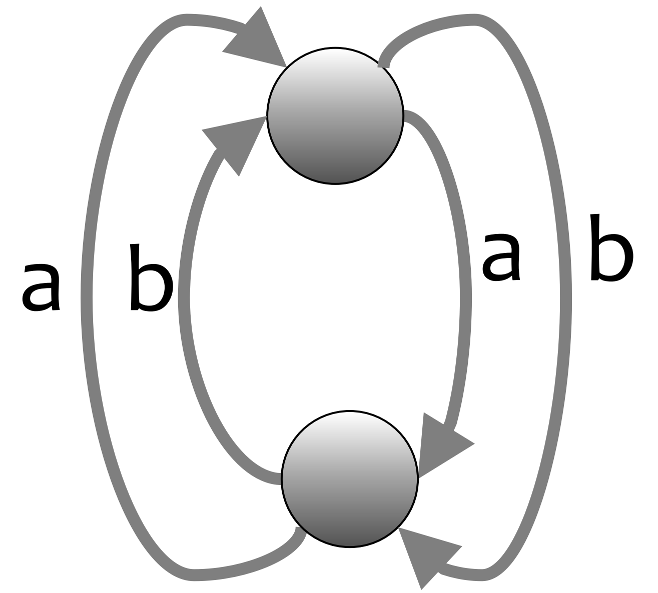
baaabaaab
と歩くと…
必ず同じ点に
着きます

このDFAの
同期語
といいます。

Cerný 予想

問題:
N 頂点の DFA が与えられる。
最短の同期語の長さを求めよ。

存在しない場合は -1 を返せ



NP 困難

≪Cerný 予想≫

予想：**同期語**が存在するなら、
長さは必ず $(N\text{-}1)^2$ 以下である。

Jan Cerný, 1964

文字列集合を

論理式で表現

```
b_after_a(string s) :=
  forall i in indices(s)
    if s[i]==‘a’ then
      exists j in indices(s)
         i≦j and s[j]==‘b’
```

「aが出たら後でbも出る」

```
even_A(string s) :=
 exists A ⊆ indices(s)
  forall i in indices(s)
     if i in A then s[i]==‘a’
                    else s[i]!=‘a’
   & |A| mod 2 = 0
```

「aが偶数個」

（詳細略）（紹介だけ）

* and, or, not , if ～ then
* (添え字に関する) forall
* (添え字に関する) exists
* (添え字の集合に関する) forall
* (添え字の集合に関する) exists
* mod 定数 で数をカウント

すべて Automaton に変換できることが
知られています

文字列集合を

パターンで表現

正規表現

Perl, Ruby, Java 等での
普通のプログラミングでよく使います。

b*ab*(ab*ab*)*

「aが奇数個」

a(a|b)(a|b)*　|　b((a|b)*b)?

「aで始まって長さ2以上、
　またはbで始まってbで終わる」

正規表現の使用例

```
C:\Users\kinaba\Desktop> irb
irb(main):001:0> /^b*ab*(ab*ab*)*$/ === "ababa"
=> true
irb(main):002:0> /^b*ab*(ab*ab*)*$/ === "abababa"
=> false
irb(main):003:0> /^a(a|b)(a|b)*|b((a|b)*b)?$/ === "a"
=> false
irb(main):004:0> /^a(a|b)(a|b)*|b((a|b)*b)?$/ === "bb"
=> true
irb(main):005:0>
```

正規表現

hoge | fuga

- hoge の表す集合と fuga の表す集合の和集合

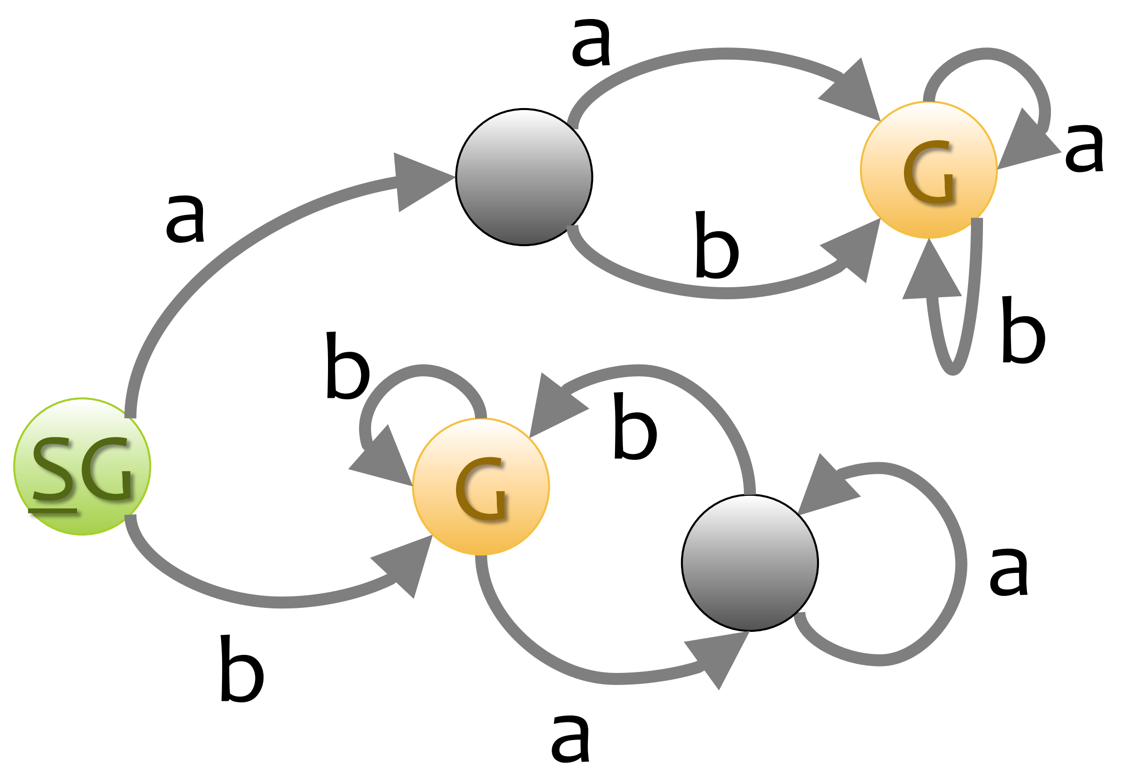
hoge*

- hoge の表す集合の要素を
  0個以上何個でも並べた文字列の集合

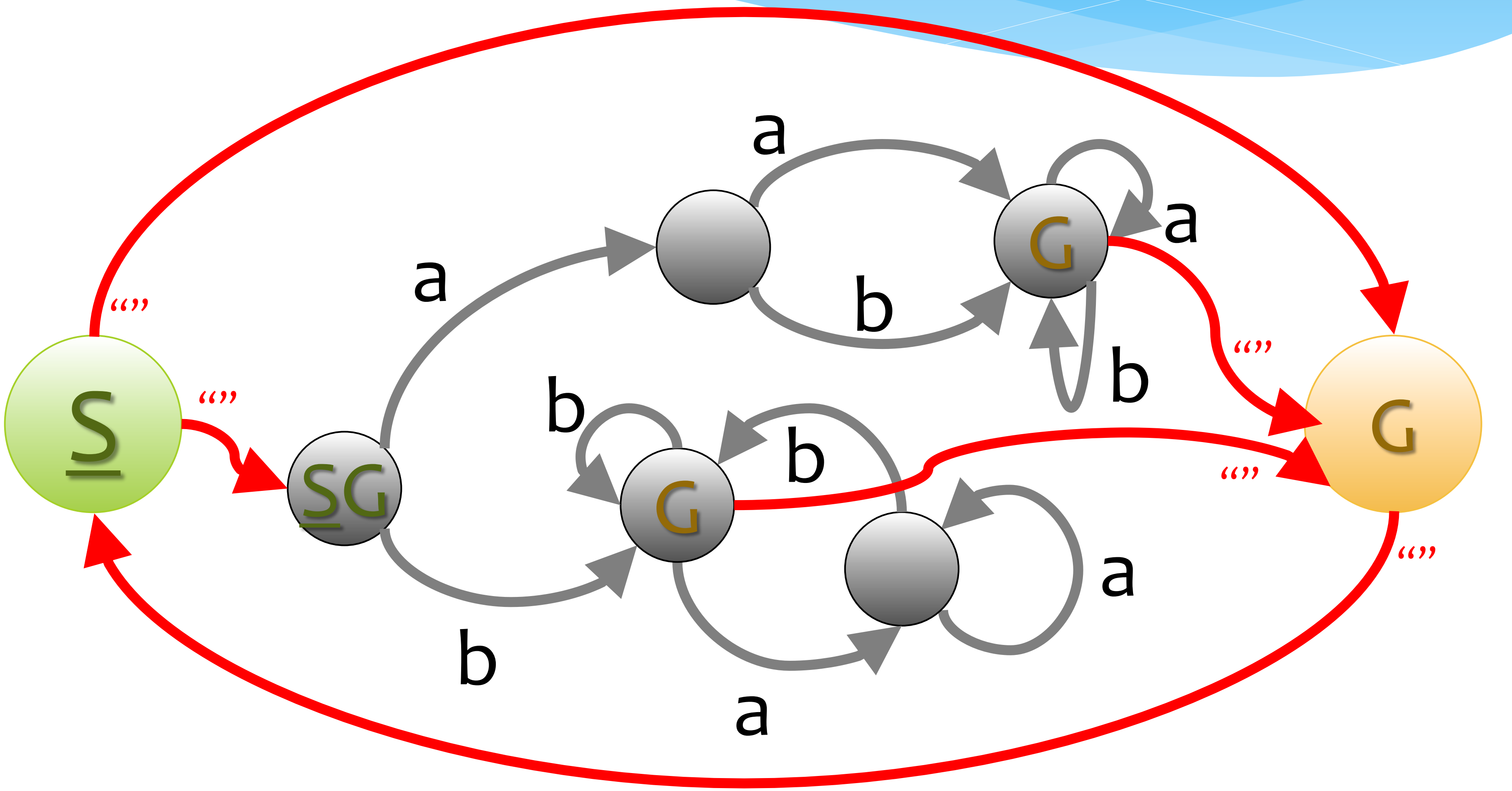
hoge?

- hoge または空文字列

NFA で表せます
hoge* の例



NFA で表せます
hoge* の例

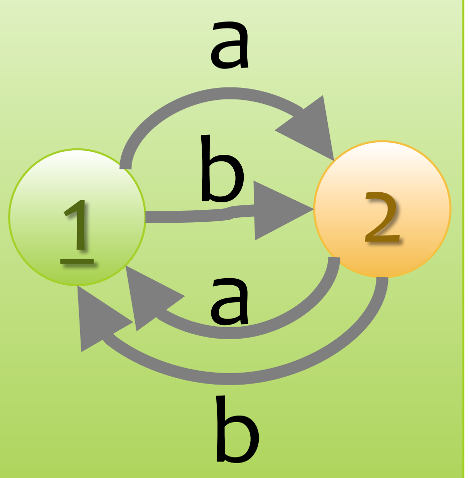


逆に、Automaton は すべて正規表現で書けます

```
foreach(i : nodes)
  d[i][i] = 0;
foreach(i--c-->j : edges)
  d[i][j] = c;
foreach(k : nodes)
  foreach(i : nodes)
    foreach(j : nodes)
      d[i][j] = min(
        d[i][j], d[i][k]+d[k][j]);
```

方針:

Warshall-Floyd

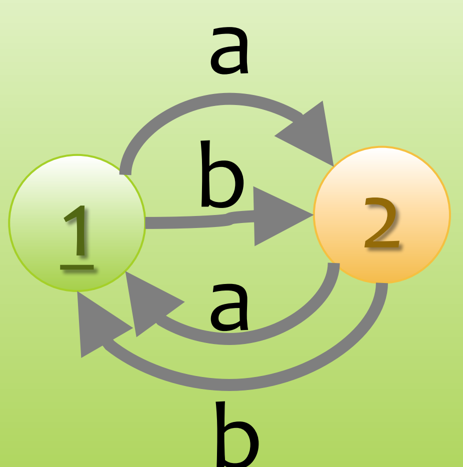


逆に、Automaton は すべて正規表現で書けます

```
foreach(i : nodes)
  d[i][i] = “”;
foreach(i--c-->j : edges)
  d[i][j] = d[i][j] | c;
foreach(k : nodes)
  foreach(i : nodes)
    foreach(j : nodes)
      d[i][j] =
        d[i][j] | d[i][k] d[k][k]* d[k][j];
```

方針:

Warshall-Floyd



おまけ: 形式言語理論の未解決問題
Star-Height 問題

さっきの「Automaton→正規表現」変換は * を使いすぎる。もっと減らせないか？

- | (和集合), * (繰り返し) の他に & (積集合), ¬(補集合), Φ (空集合) も使う正規表現を一般正規表現という

b*ab*(ab*ab*)*

「aが奇数個」

「aが偶数個、じゃない」

¬((¬(¬Φa¬Φ)a¬(¬Φa¬Φ)a¬(¬Φa¬Φ))*)

≪Star-Height 問題≫

*** のネストなし**で、Automatonを
全て一般正規表現で表せるか？

Janusz Brzozowski, 1980

文字列集合を

文法で表現

{“1+2*3/(4-5)”, “0*0*0+0”, …}

```
EXPR ::= TERM
       | TERM “+” TERM
       | TERM “-” TERM
TERM ::= FACTOR “*” FACTOR
       | FACTOR “/” FACTOR
FACTOR ::= “(“ EXPR “)”
         | “0” | “1” | ... | ”9”
```

「一桁の数の四則演算式」

{“”, “a”, “b”, “aa”, “aba”, …}

```
PALINDROME ::= “a”
             | “b”
             | “”
             | “a” PALINDROME “a”
             | “b” PALINDROME “b”
```

「回文」

文脈自由文法
Context Free Grammar

例）

PALIN
→ “a” PALIN “a”
→ “a” “b” PALIN “b” “a”
→ “a” “b” “a” PALIN “a” “b” “a”
→ “a” “b” “a” “” “a” “b” “a”
→ “abaaba”

```
PALINDROME ::= “a”
             | “b”
             | “”
             | “a” PALINDROME “a”
             | “b” PALINDROME “b”
```

左辺 ::= 右辺 | 右辺 | … | 右辺
という規則の集まりを、
「左辺の記号を右辺のどれかに書き換え」を
繰り返したら作れる文字列の集合、と見なします。

bool contains(CFG g, string w);

右辺の長さが
2以下になるように
変形してから・・・

```
P  ::= “”
     | “a”
     | “b”
     | “a” P “a”
     | “b” P “b”
```

```
P  ::= “”
     | “a”
     | “b”
     | “a” Q
     | “b” R
Q  ::= P “a”
R  ::= P “b”
```

bool contains(CFG g, string w);

右辺の長さが
2以下になるように
変形してから・・・

動的計画法。O (|g| |w|³)。

```
bool
dp[左辺記号][i][k] =
  "左辺記号" から w[i .. k)
  が作れるか？
```

```
P ::= “”
    | “a”
    | “b”
    | “a” Q
    | “b” R
Q ::= P “a”
R ::= P “b”
```

bool contains(CFG g, string w);

（オーダーの意味で）世界最速は2012年3月現在

$$O(|g| \cdot |w|^{2.3723})$$

だそうです。

行列の掛け算と同じオーダです。

ほかの集合演算は？

- 空集合かどうかの判定 ➔ できる（簡単）
- 和集合の計算 ➔ できる（簡単）
- 共通部分
  - CFG と CFG の共通部分は CFG では書けない！
- 補集合
  - CFG の補集合は CFG では書けない！
- 等しさ ＝ の判定、包含関係 ⊆ の判定
  - 決定不能！

決定不能

Post の対応問題

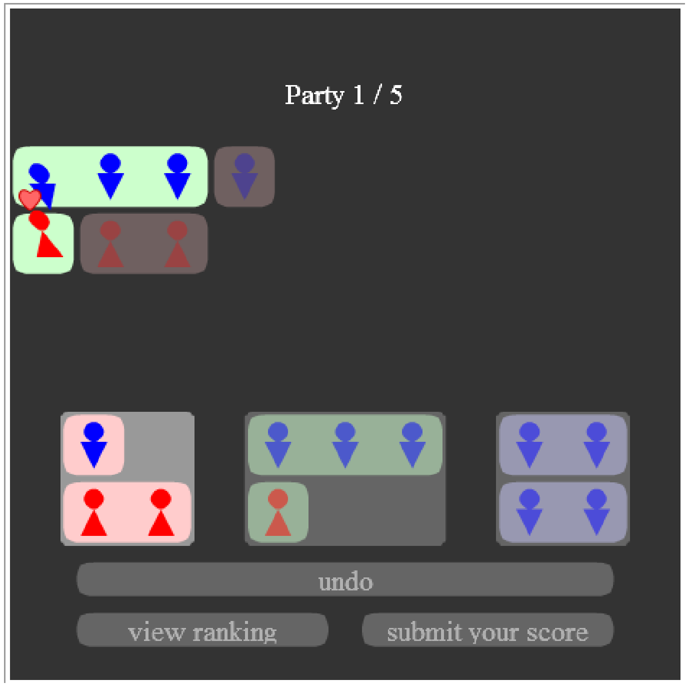
[入力] 0/1 の列たち $a_1, \ldots, a_r, b_1, \ldots, b_r$

[出力] $a_{i1}a_{i2}\ldots a_{ik} = b_{i1}b_{i2}\ldots b_{ik}$ となるような $i_1, i_2, \ldots, i_k$ $(k > 0)$ は存在するか？ (i_j は同じ値を複数回含んでもよい)

http://hos.ac/slides/ から引用

Postの対応問題ゲーム

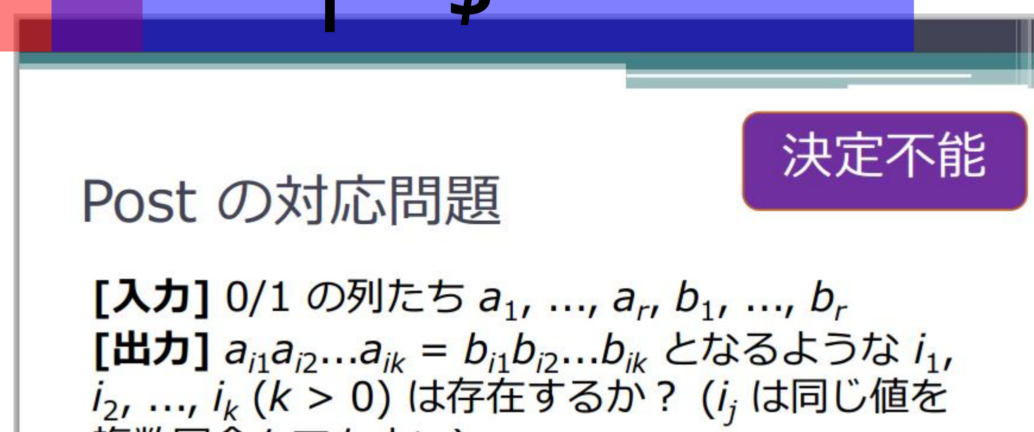
http://d.hatena.ne.jp/ku-ma-me/20100724/p1



「2つのCFGの共通部分が空か？」は決定不能

$$
\begin{aligned}
A \rightarrow\ & a_1\ A\ \text{“1”} \\
\mid\ & a_2\ A\ \text{“2”} \\
& \cdots \\
\mid\ & a_r\ A\ \text{“r”} \\
\mid\ & \text{“\$”}
\end{aligned}
$$

$$
\begin{aligned}
B \rightarrow\ & b_1\ B\ \text{“1”} \\
\mid\ & b_2\ B\ \text{“2”} \\
& \cdots \\
\mid\ & b_r\ B\ \text{“r”} \\
\mid\ & \text{“\$”}
\end{aligned}
$$



演算はあまりサポートしない

* CFG と CFG の共通部分の空判定は決定不能
  → CFG と CFG の共通部分は CFG では書けない
  * 空判定はできるので、Post の対応問題が解けちゃう
* CFG が文字列全部を表してるかの判定は決定不能（証明略）
  → CFG の補集合は CFG で書けない
  → CFG =? CFG や CFG ⊆? CFG は決定不能
* CFG と DFA の共通部分は計算可能
  → CFG ⊆? DFA は判定可能

文字列検索／解析以外への

【応用事例】

文字列の「型」に使う

* **和集合**や**文字列結合**を使って、演算結果の型を計算
* 集合の**包含判定**を使って、型キャストのチェック

```
string<(a|b)(a|b)*> s;
s = “”;    //コンパイルエラー！
s = “c” ; //コンパイルエラー！
string<(a|b)(a|b)(a|b)*> t = s; //エラー！
string<(a|b)(a|b)(a|b)*> u = s+s; //OK！
```

プログラムが正しい順で関数を呼ぶことの検証

```
File file = new File(“input.txt”);
solve(file);

void solve(File file) {
  int T = file.ReadInt();
  solveCases(T, file);
}
void solveCases(int N, File file) {
  if(N == 0) file.Close();
  else {String s = file.ReadLine();
        // solve here...
        solveCases(N-1, file); }
}
```

プログラムが正しい順で関数を呼ぶことの検証

```
File file = new File("input.txt");
solve(file);

void solve(File file) {
  int T = file.ReadInt();
  solveCases(T, file);
}
void solveCases(int N, File file) {
  if(N == 0) file.Close();
  else {String s = file.ReadLine();
        // solve here...
        solveCases(N-1, file); }
}
```

プログラムの挙動

```
MAIN  ::= "new" SOLVE
SOLVE ::= "readint" CASES
CASES ::= "close" | "readline" CASES
```

プログラムが正しい順で関数を呼ぶことの検証

```
File file = new File("input.txt");
solve(file);

void solve(File file) {
  int T = file.ReadInt();
  solveCases(T, file);
}
void solveCa
  if(N == 0) file.Close();
  else {String s = file.ReadLine();
        // solve here...
        solveCases(N-1, file); }
}
```

ファイルは
この順で
操作しないとダメ！

```
"new" ("readint" | "readline")* "close"
```

プログラムの挙動

```
MAIN  ::= "new" SOLVE
SOLVE ::= "readint" CASES
CASES ::= "close" | "readline" CASES
```

プログラムが正しい順で関数を呼ぶことの検証

```
File file = new File("input.txt");
solve(file);

void solve(File file) {
  int T = file.ReadInt();
  solveCases(T, file);
}
void solveCa
  if(N == 0) file.Close();
  else {String s = file.ReadLine();
        // solve here...
        solveCases(N-1, file); }
}
```

CFG ⊆? DFA
は　決定可能

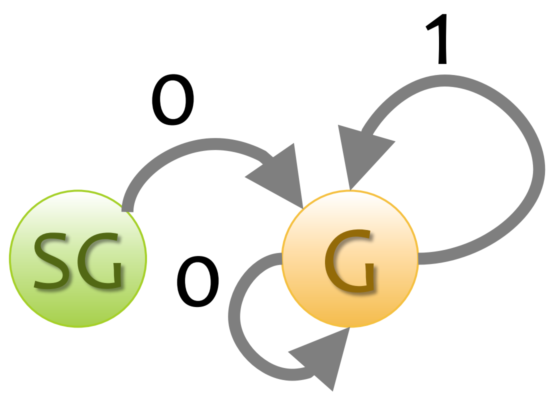
```
"new" ("readint" | "readline")* "close"
```

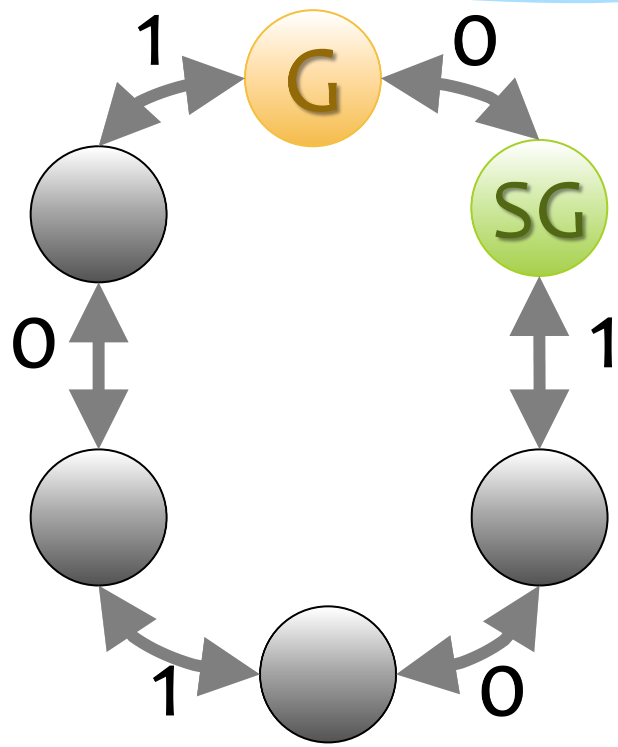
⊆ ?

```
MAIN  ::= "new" SOLVE
SOLVE ::= "readint" CASES
CASES ::= "close" | "readline" CASES
```

自然数の集合を表す
(2進数表記を下位ビットから並べた文字列として)

「２の倍数」

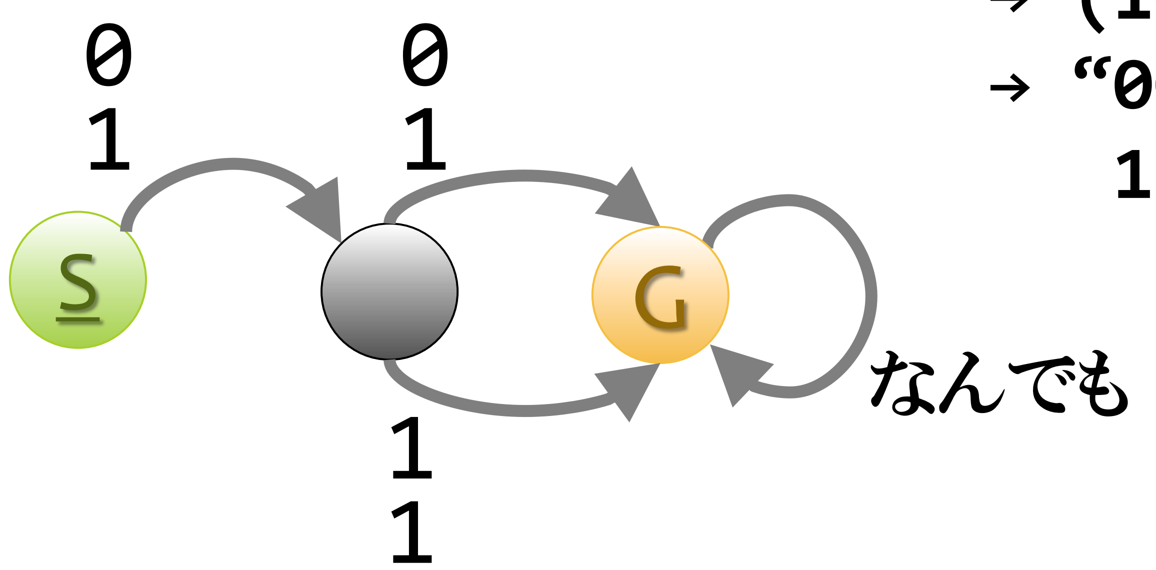




「３の倍数」

自然数のペアやn個組の集合
(nビットを一文字として)

「偶数と、3 mod 4 の数のペア」



(16, 7)
→ (10000, 00111)
→ "00001
11100"

いろいろな演算

Automaton で自然数の

* 大小関係
* 足し算
* 固定幅ビットシフト
* 定数での mod

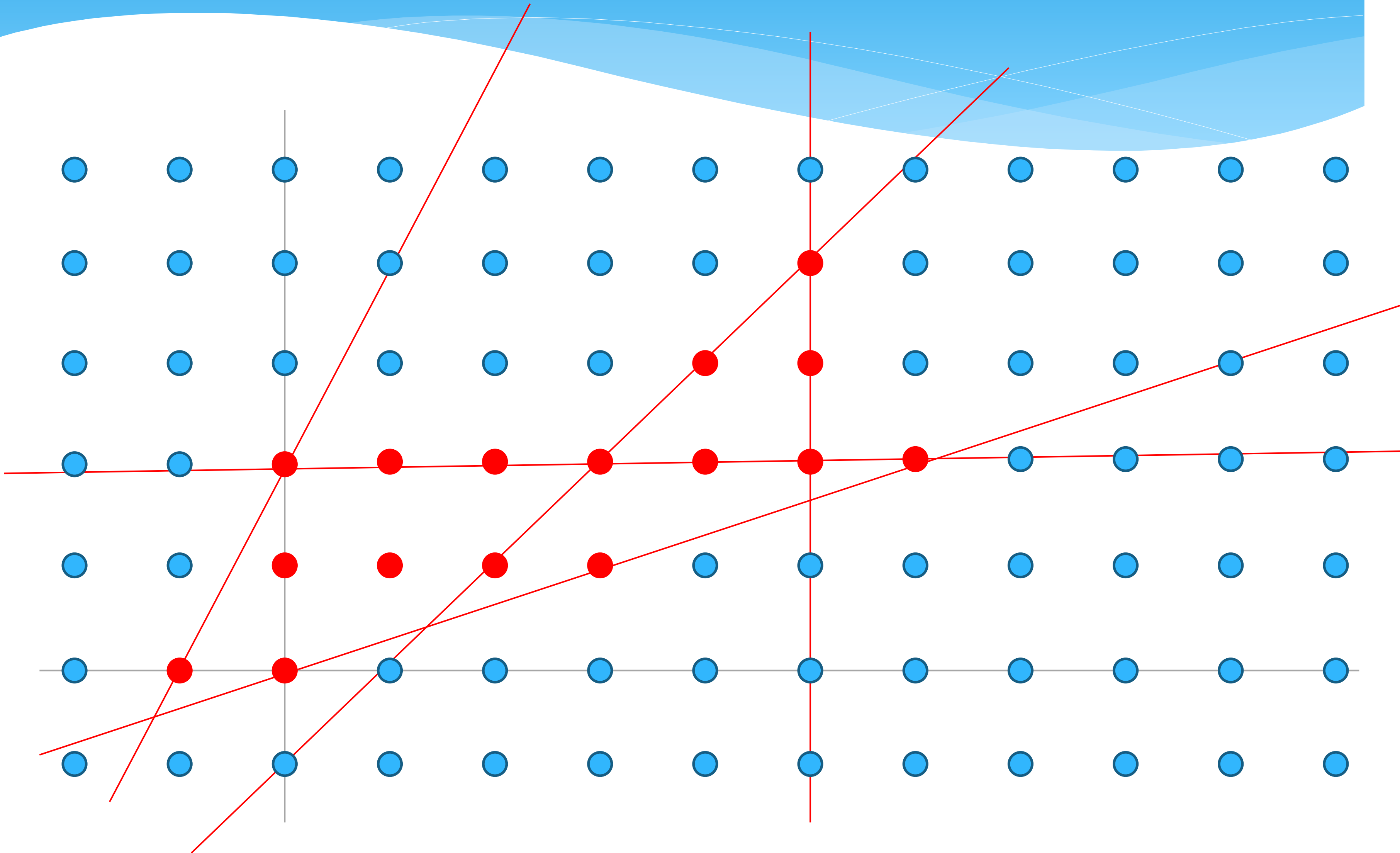
などが定義できる

もちろん

* 和集合
* 共通部分

なども・・・

こんな集合が表現できます。幾何！



まとめ

まとめ

まとめ

* 文字列の(無限)集合の幾つかの表現方法を紹介しました
  * 有向グラフで表現する
  * 論理式で表現する
  * パターンで表現する
  * 文法で表現する
* 色々な集合演算の
  * 実装方法
  * 応用

を紹介しました。