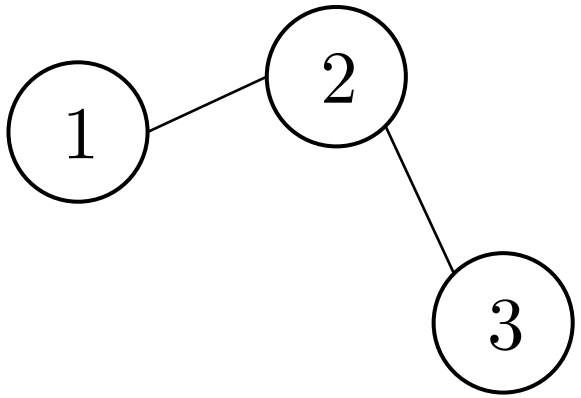
コードによる作図法 morph

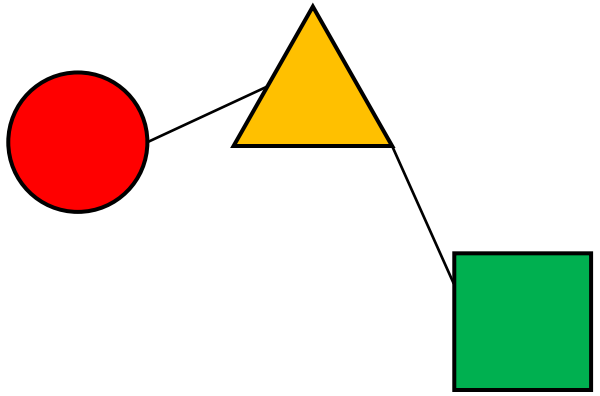
概要

- コードによる**定量的な作図法**を設計する
  - cf. LaTeX
- Pythonを実装に用いた
  - `usage: python3 morph.py > foo.svg`
  - `code:` https://github.com/sora410/morph
  - `.py`内で, `@morphscript`デコレータを付与した関数に, 図形を生成するコードを書き込み実行すると, SVGの中身が標準出力される

```
@morphscript
def sketch():
    ids = [1,2,3]
    d = Declare(ids, lambda i, N: (Vec.d(240) + Vec.d(120 * i)) * 4)

    def arrow(b, e):
        mid = (b + e) / 2
        up  = (e - b).rot(90) * 0.15
        return Curve(b, mid + up, e, em="Triangle")

    r1 = Relate([1,2,3,1], arrow)
    r2 = Relate([3,2,1,3], arrow)

    c = Circle(1); c.orig = Vec(10,10)
```

コンセプト





配置を先に決めて

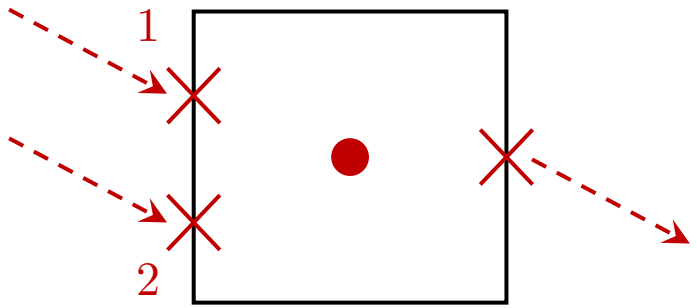
図形に化かす

図形

- **描画**を行うすべてのクラスがFigureを継承
  - draw(self, off, unit)メソッドをオーバーライド

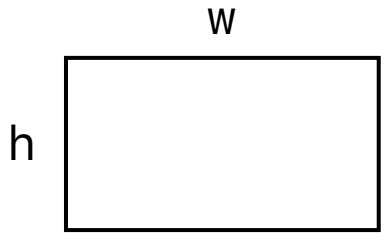
- **来点**, **行点**, **接続点**を設定できる
  - 点の指定は**相対的**な**比**の座標を用いる

```
r = Rect(1,1)
r.i = [Vec(0,1/4),Vec(0,3/4)]
r.o = [Vec(1,1/2)]
r.mg = [Vec(1/2,1/2)]
```

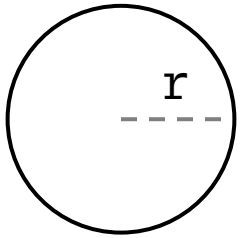


原始図形

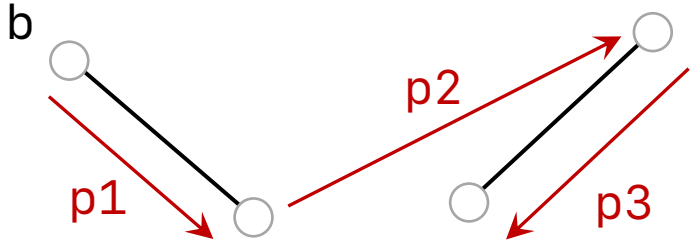
- **長方形**：Rect(width,height)



- **円**：Circle(r)



- **パス**：Path(b).l(p1).m(p2).l(p3)...

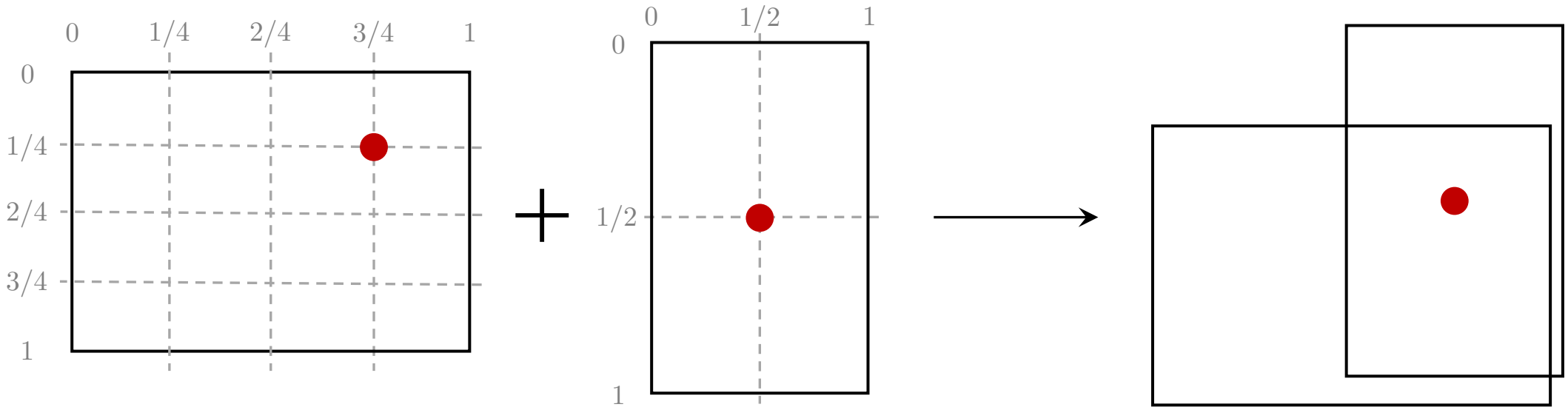


- **テキスト**：Text(text,fontsize)

- etc. (これから対応予定)

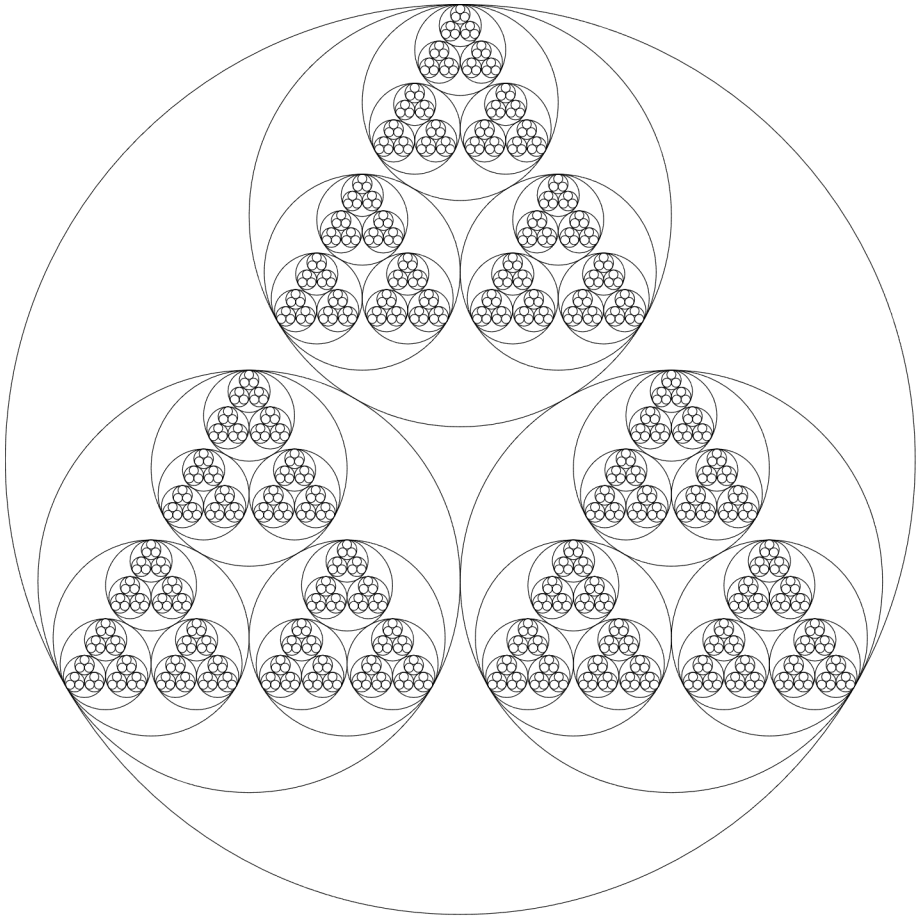
図形の合成

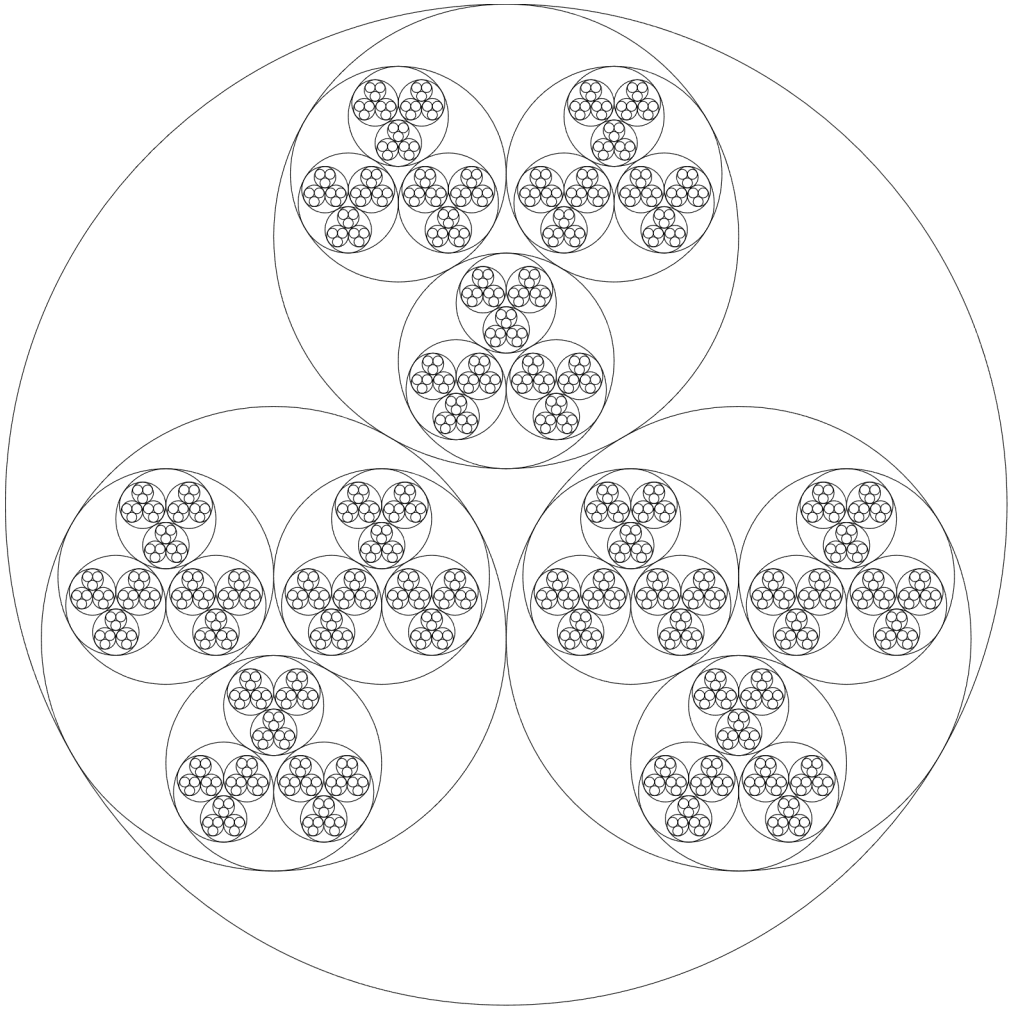
- **接続点**をもとに図形をくっつける
  - 点の指定は**相対的**な**比**の座標を用いる




```
Rect(2,1.6)[Vec(3/4,1/4)] + Rect(1,3)[Vec(1/2,1/2)]
```

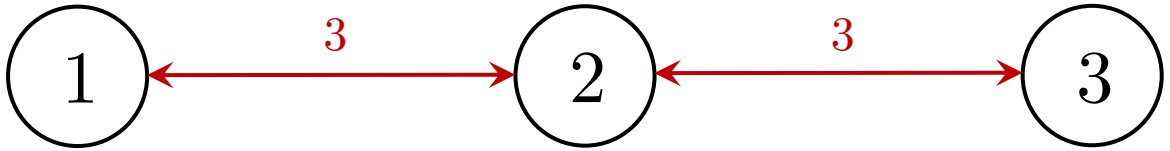
デモ①（フラクタル図形）



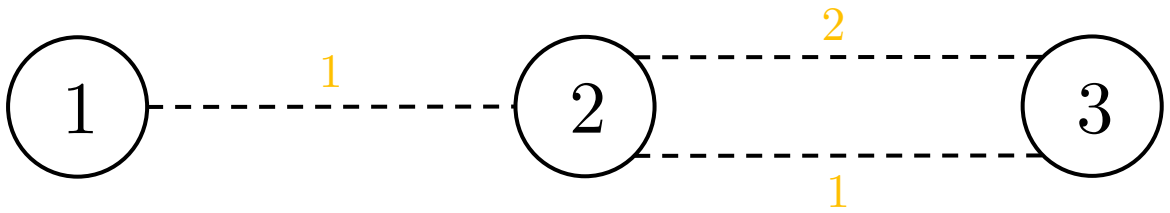


配置と関係

- 整数値のidを用いて，図形どうしの**配置**と**関係**を定義
  - `Declare([1,2,3],lambda i,N: (3,0))`



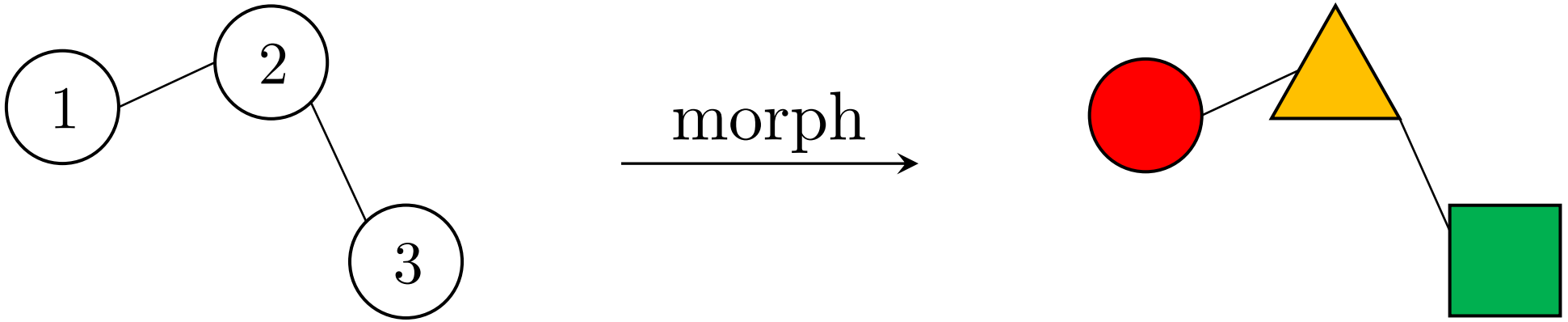
  - `Connect([1,2,3]); Connect([2,3]);`
    - 図形どうしを**直線**で繋ぐ. **実際にどの2点が繋がるのかは図形に依存する**



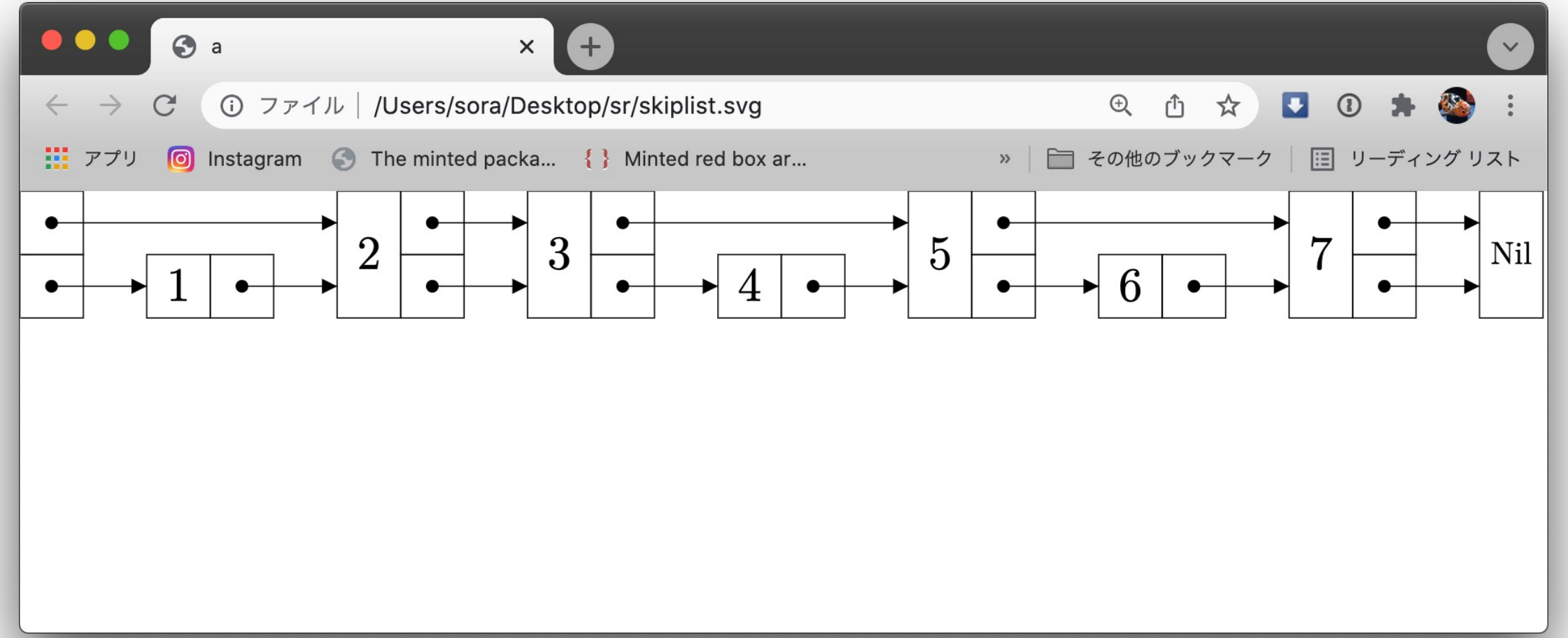
化かす

- 実は, `Figure`インスタンスは, idを引数にして**呼び出せる**
  - **例**：`r = Rect(1,2);` `r(1,2)`
  - この例では, idの1,2が, 長方形の`r`と紐づけられる

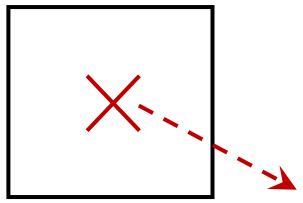
- `Declare`や`Connect`の描画時には, この対応が用いられ, idで仮置きしていた箇所が, 対応する図形に**化ける**



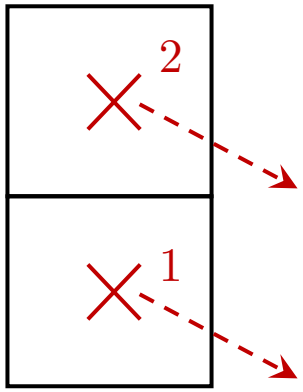
デモ②（スキップリスト）


a
ファイル
/Users/sora/Desktop/sr/skiplist.svg
アプリ
Instagram
The minted packa...
Minted red box ar...
その他のブックマーク
リーディング リスト
1
2
3
4
5
6
7
Nil

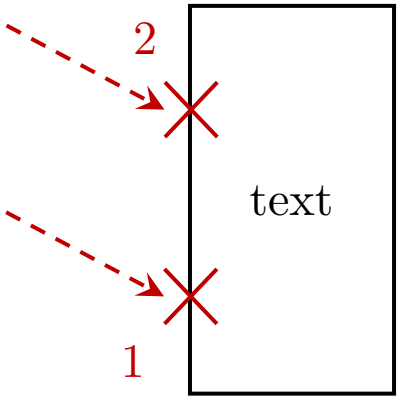
デモ②（スキップリスト）



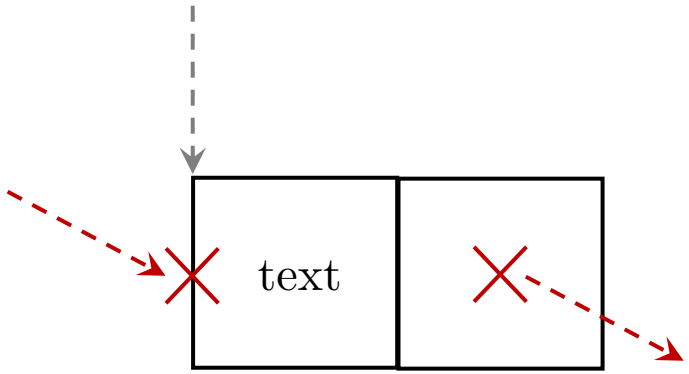
a



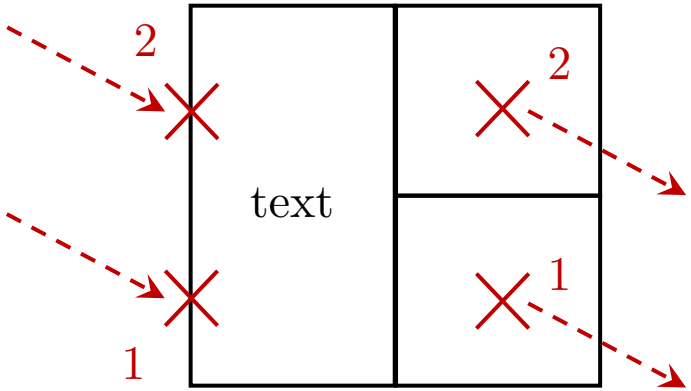
aa



r(text)



np(text)



p(text)

これから

- パスの**マーカ**の定義法の確立
- 図形の**回転**への対応
- **木構造**を扱えるように
- 図形の概形の**大きさ**を保持して、自動リサイズ
- **原始図形**を増やす

- SVG以外の形式への対応
- 専用GUIエディタの作成

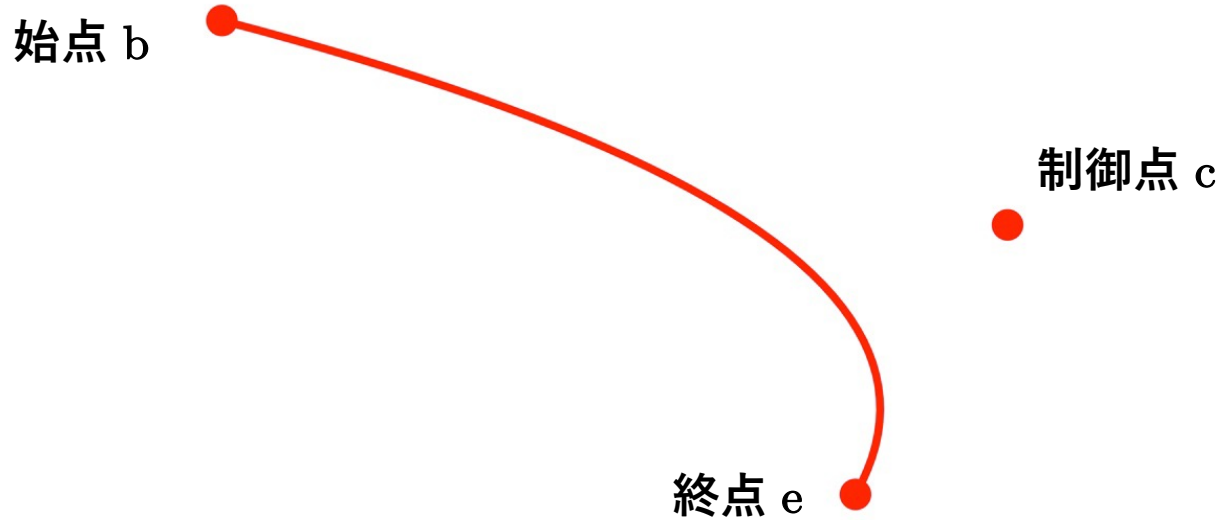
後半で扱った主なトピック

- 曲線を**直観的に**指定する
- **回転**について

曲線描画

- SVGで**曲線**を描画する方法は：
  - 3次ベジェ曲線 / **2次ベジェ曲線　→ これを使いたい**
  - 楕円弧曲線

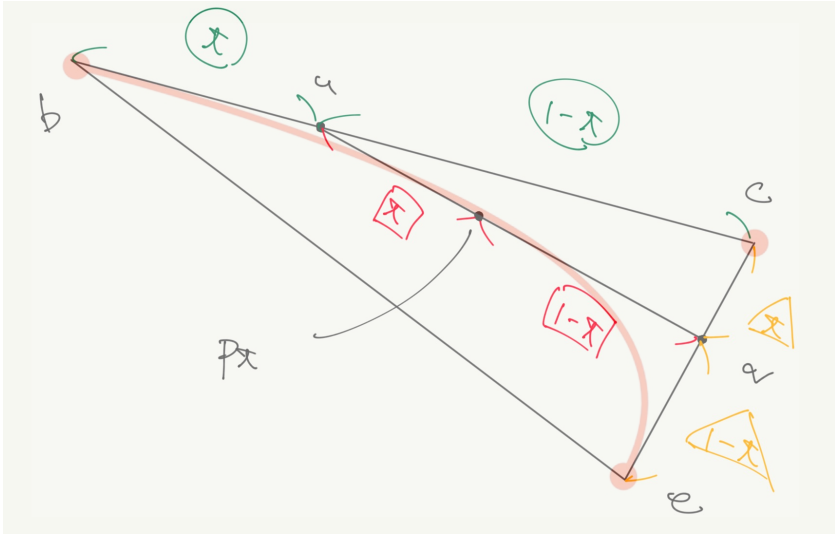
- しかし，ベジェ曲線をGUIなしに使うのは難しい（と感じた）



2次ベジェ曲線

- **2次ベジェ曲線**は左の式で定義され, その幾何的意味は右の通り

$$\mathbf{p}_t = (1-t)^2\mathbf{b} + 2t(1-t)\mathbf{c} + t^2\mathbf{e}$$
$$(0 \leqq t \leqq 1)$$



- 「一番高くなる点」を指定する方が分かりやすい（と思った）
  - 制御点 $\mathbf{c}$ の代わりに, $\mathbf{p}_t$ のうち, 線分 $\mathbf{be}$ との距離が最大となるような $\mathbf{p}_t$ を指定する → 実は $t = 1/2$ のとき最大

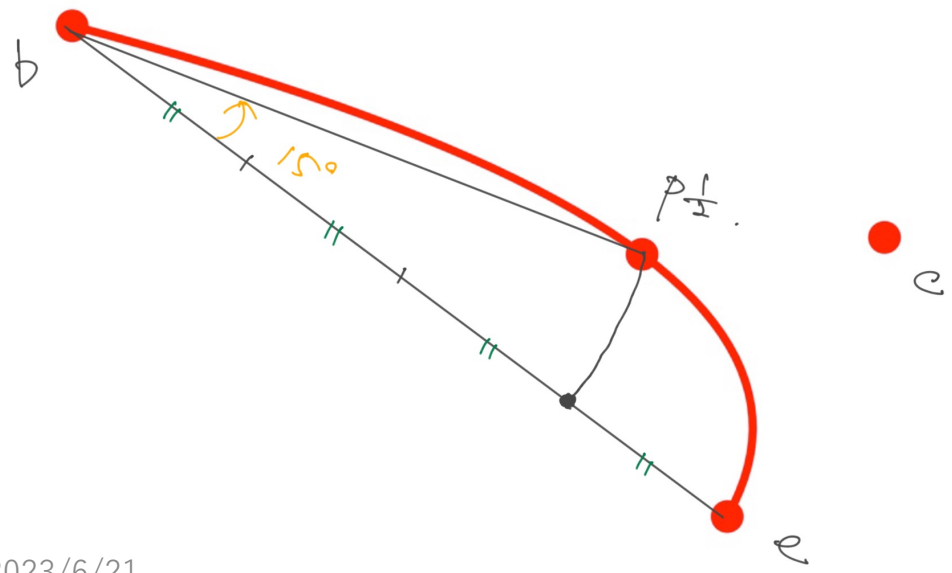
2次ベジェ曲線

$$\mathbf{p}_t = (1-t)^2\mathbf{b} + 2t(1-t)\mathbf{c} + t^2\mathbf{e}$$
$$(0 \leqq t \leqq 1)$$

- 定義式に $t = 1/2$ を代入して,

$$\mathbf{p}_{1/2} = \frac{\mathbf{b}+\mathbf{e}}{4} + \frac{\mathbf{c}}{2}. \qquad \therefore\ \mathbf{c} = 2\mathbf{p}_{1/2} - \frac{\mathbf{b}+\mathbf{e}}{2}.$$

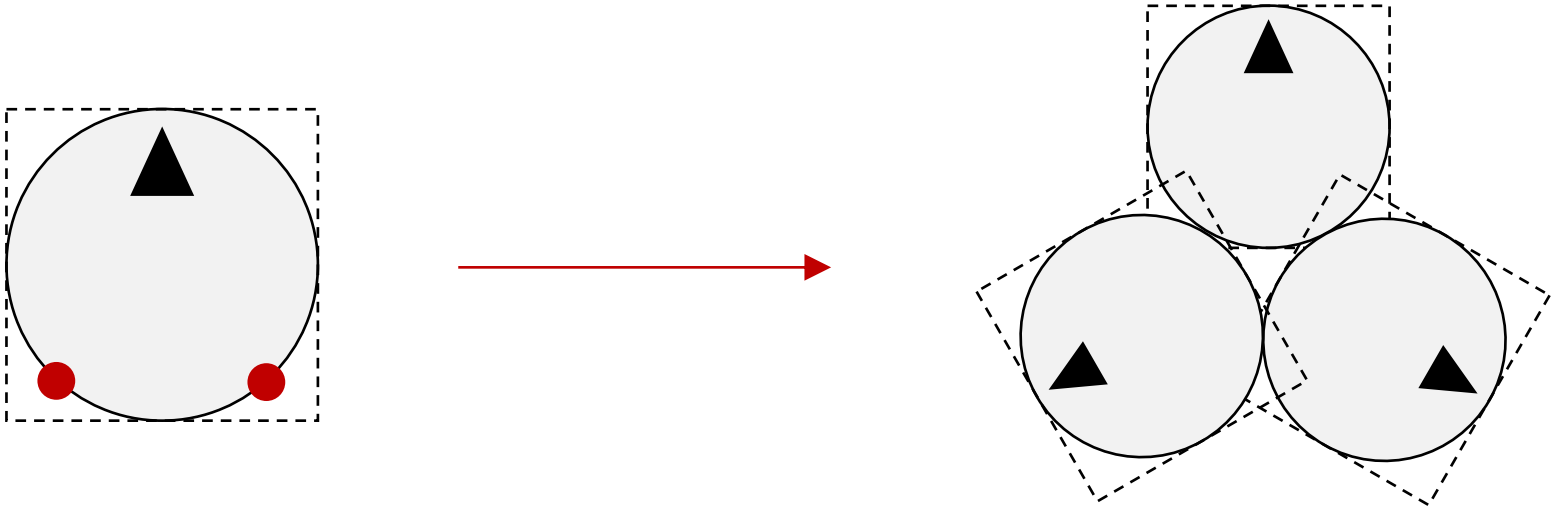
  - → つまり, 制御点 $\mathbf{c}$ の代わりに, 「最大」点 $\mathbf{m} = \mathbf{p}_{1/2}$ を与えても曲線は描ける




```python
class Bez2(Path):
    def __init__(self, b, c, e, bm=None, em=None):
        super(Bez2, self).__init__(b, bm, em)
        self.Q(c, e)

class Curve(Bez2):
    def mtoc(self, b, m, e):
        return m * 2 - (b + e) / 2

    def __init__(self, b, m, e, bm=None, em=None):
        super(Curve, self).__init__(b, self.mtoc(b, m, e), e, bm, em)
```

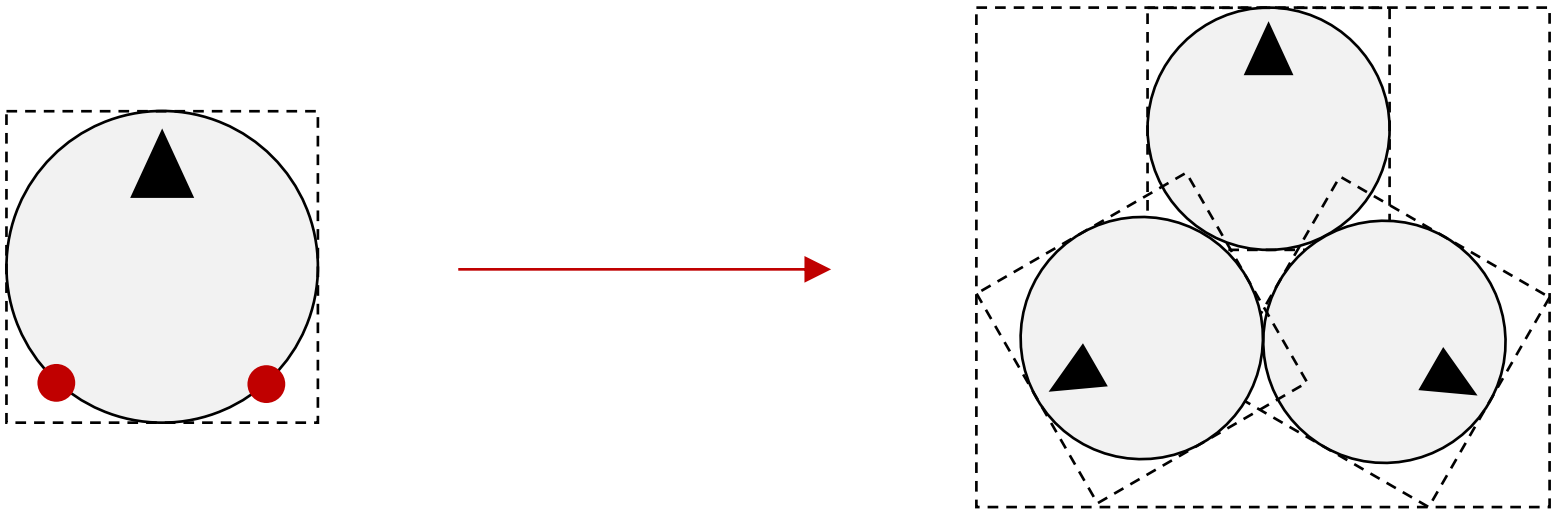
回転

- 現状, すべての図形を**長方形の外枠**で捉えている
  - あらゆる図形について、「縁」を明らかにするのが大変だから
  - → 回転した図形の外枠をどう捉えればよいか？



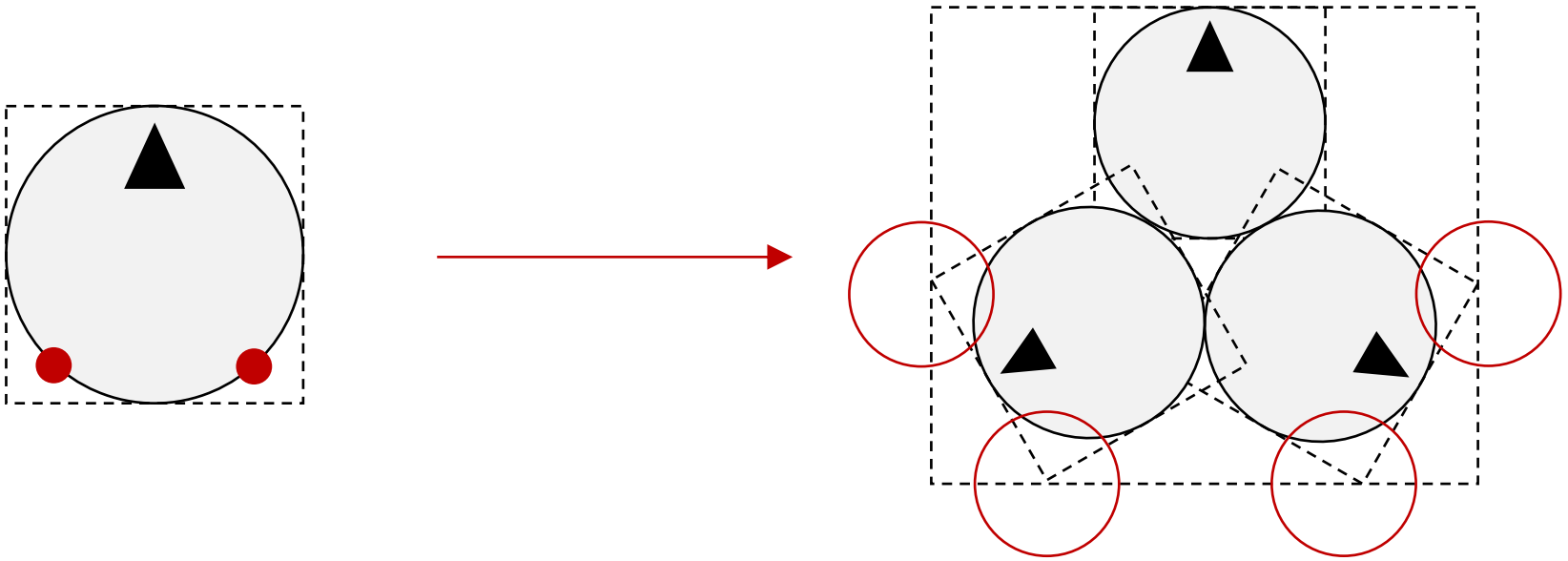
回転

- 現状, すべての図形を**長方形の外枠**で捉えている
  - あらゆる図形について、「縁」を明らかにするのが大変だから
  - → 回転した図形の外枠をどう捉えればよいか？



回転

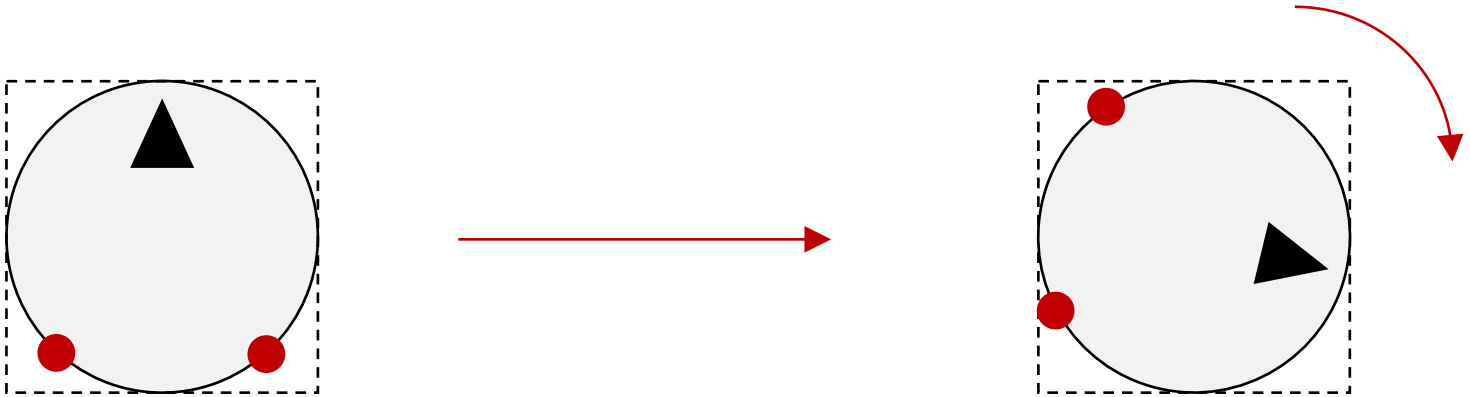
- 現状, すべての図形を**長方形の外枠**で捉えている
  - あらゆる図形について、「縁」を明らかにするのが大変だから
  - → 回転した図形の外枠をどう捉えればよいか？



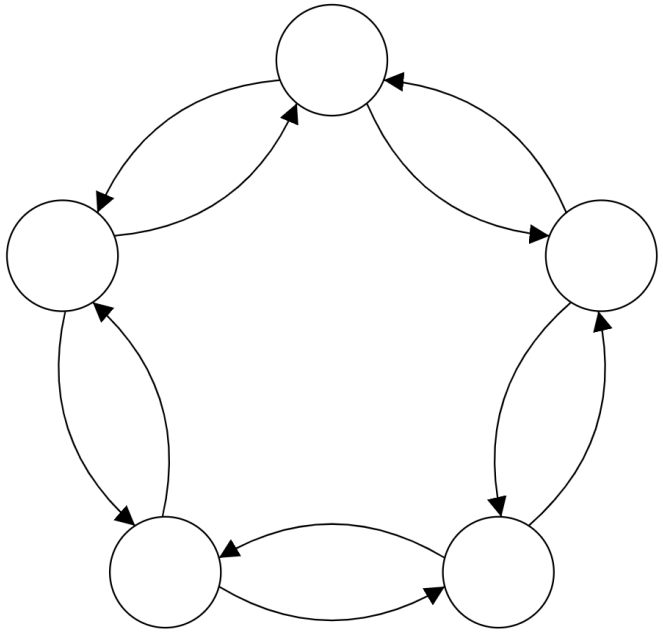
回転

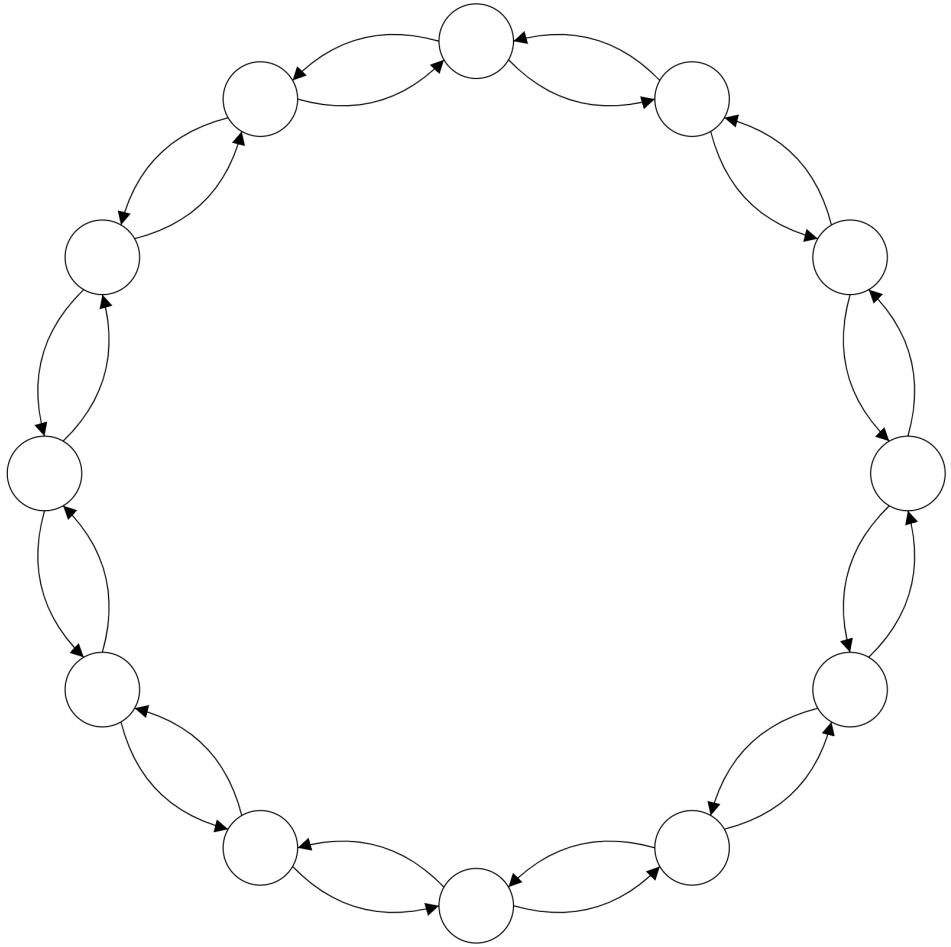
- 一般の回転をどう扱えば良いかはまだ分からない
  → ひとまず，より簡単な**円の回転**にのみ対応

- 円の中心を回転の中心とする回転移動であれば，
  移動後も円になるので扱いやすい



デモ①（円環形のオートマトン）





デモ①（円環形のオートマトン）

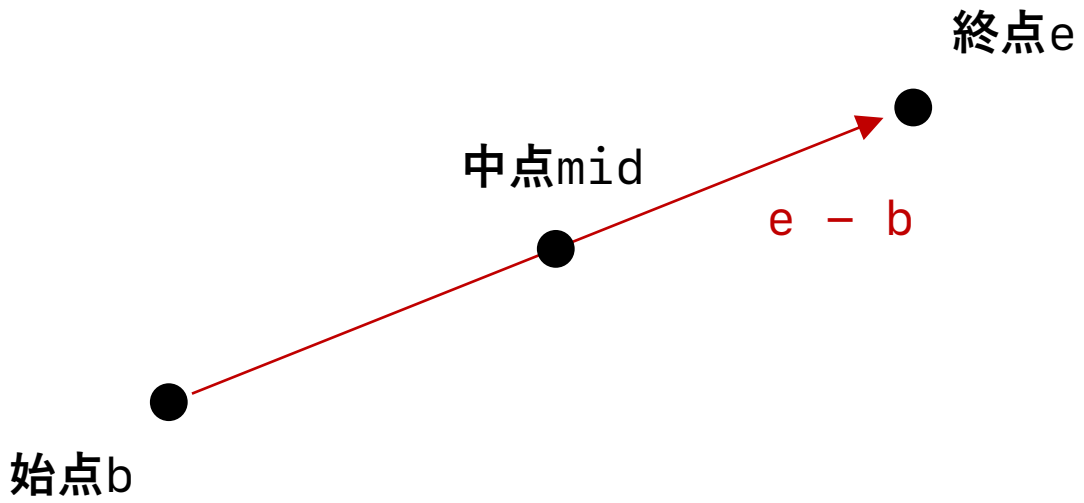
```
def arrow(b, e):
    mid = (b + e) / 2
    up = (e – b).rot(–90) * 0.15
    return Curve(b, mid + up, e, em=“Triangle”)
```

デモ①（円環形のオートマトン）

```
def arrow(b, e):
    mid = (b + e) / 2
    up = (e – b).rot(–90) * 0.15
    return Curve(b, mid + up, e, em=“Triangle”)
```

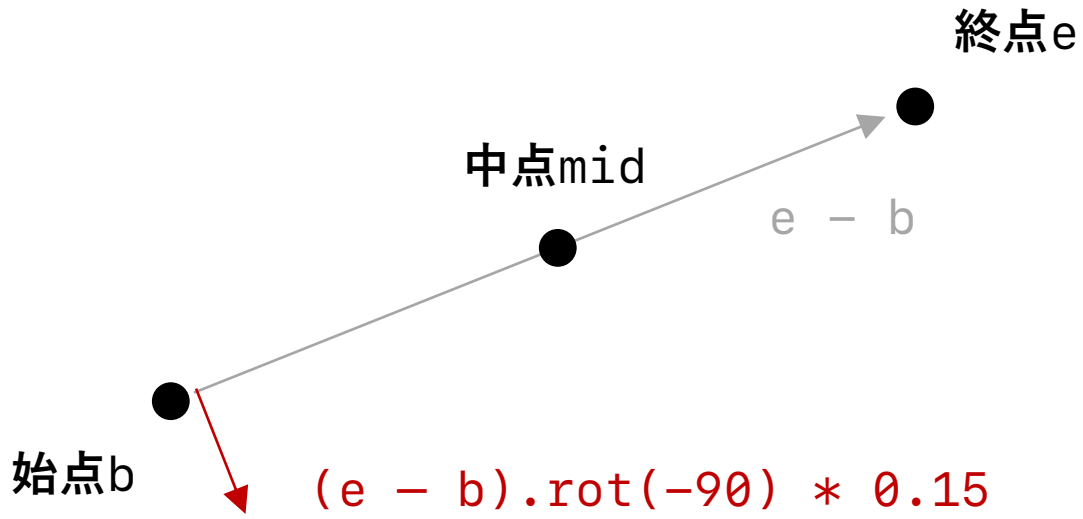
デモ①（円環形のオートマトン）

```
def arrow(b, e):
    mid = (b + e) / 2
    up = (e – b).rot(–90) * 0.15
    return Curve(b, mid + up, e, em=“Triangle”)
```



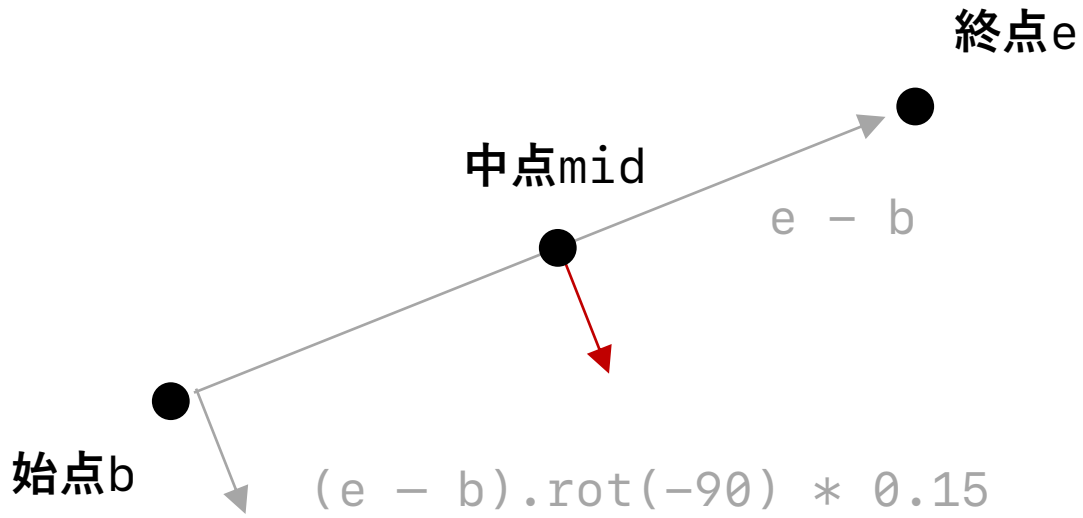
デモ①（円環形のオートマトン）

```
def arrow(b, e):
    mid = (b + e) / 2
    up = (e – b).rot(–90) * 0.15
    return Curve(b, mid + up, e, em=“Triangle”)
```



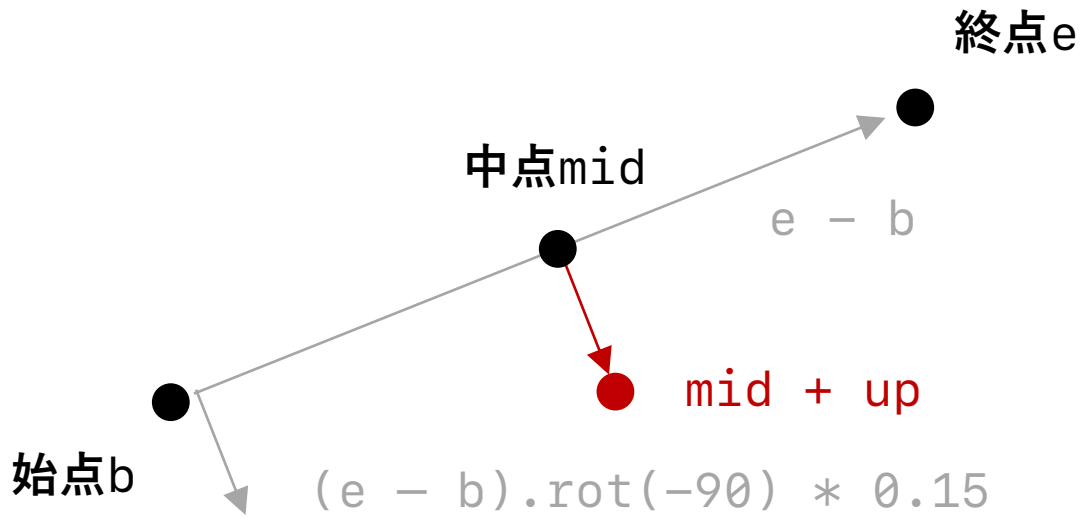
デモ①（円環形のオートマトン）

```
def arrow(b, e):
    mid = (b + e) / 2
    up = (e – b).rot(–90) * 0.15
    return Curve(b, mid + up, e, em=“Triangle”)
```



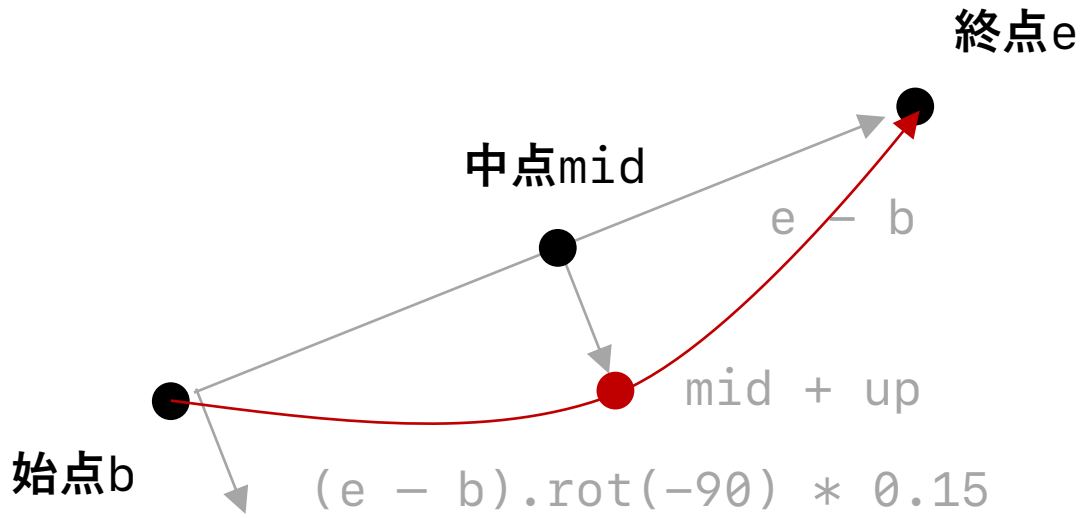
デモ①（円環形のオートマトン）

```
def arrow(b, e):
    mid = (b + e) / 2
    up = (e – b).rot(–90) * 0.15
    return Curve(b, mid + up, e, em=“Triangle”)
```

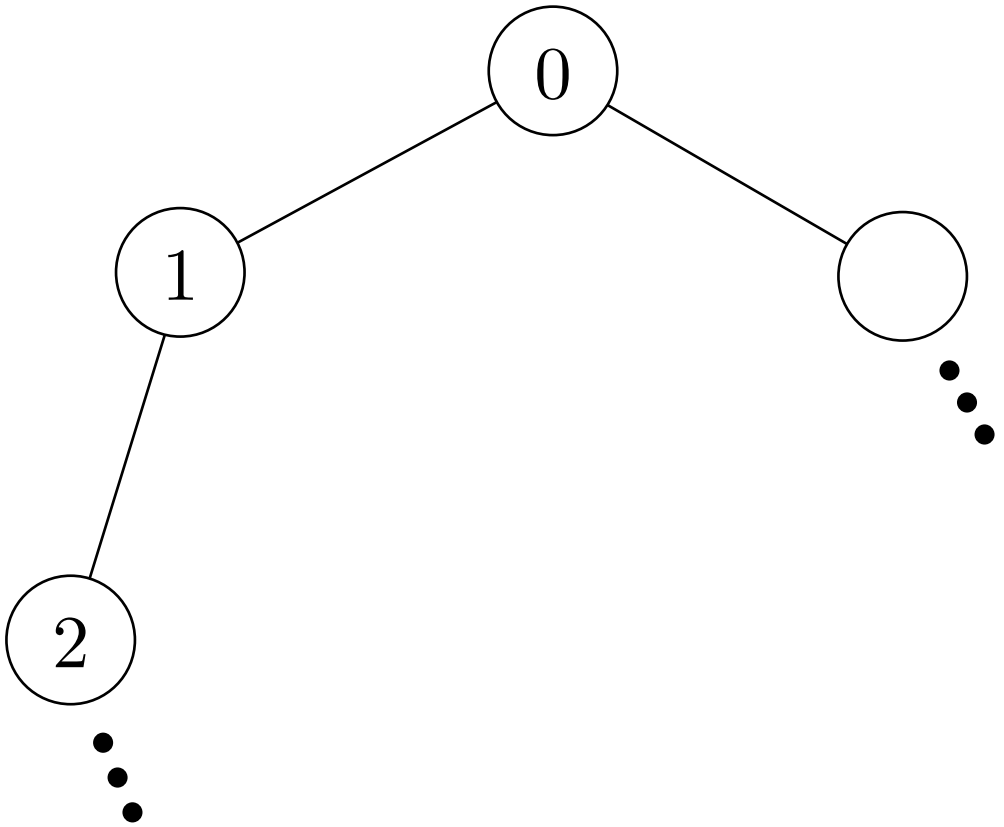


デモ①（円環形のオートマトン）

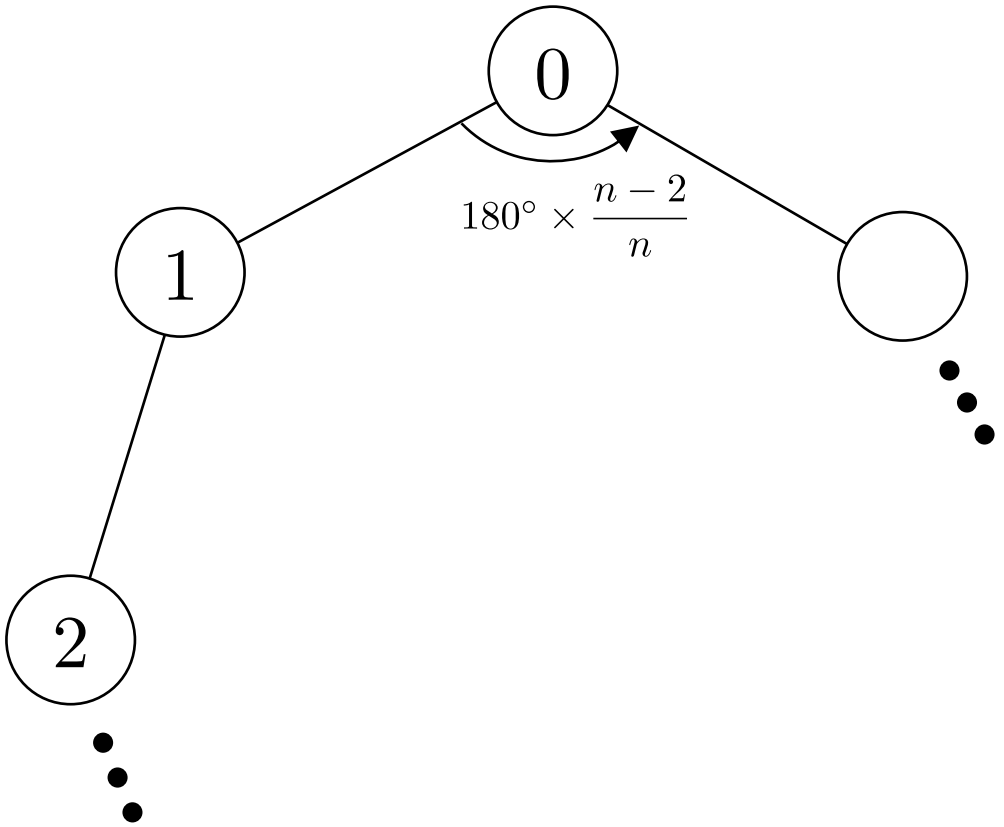
```
def arrow(b, e):
    mid = (b + e) / 2
    up = (e – b).rot(–90) * 0.15
    return Curve(b, mid + up, e, em=“Triangle”)
```



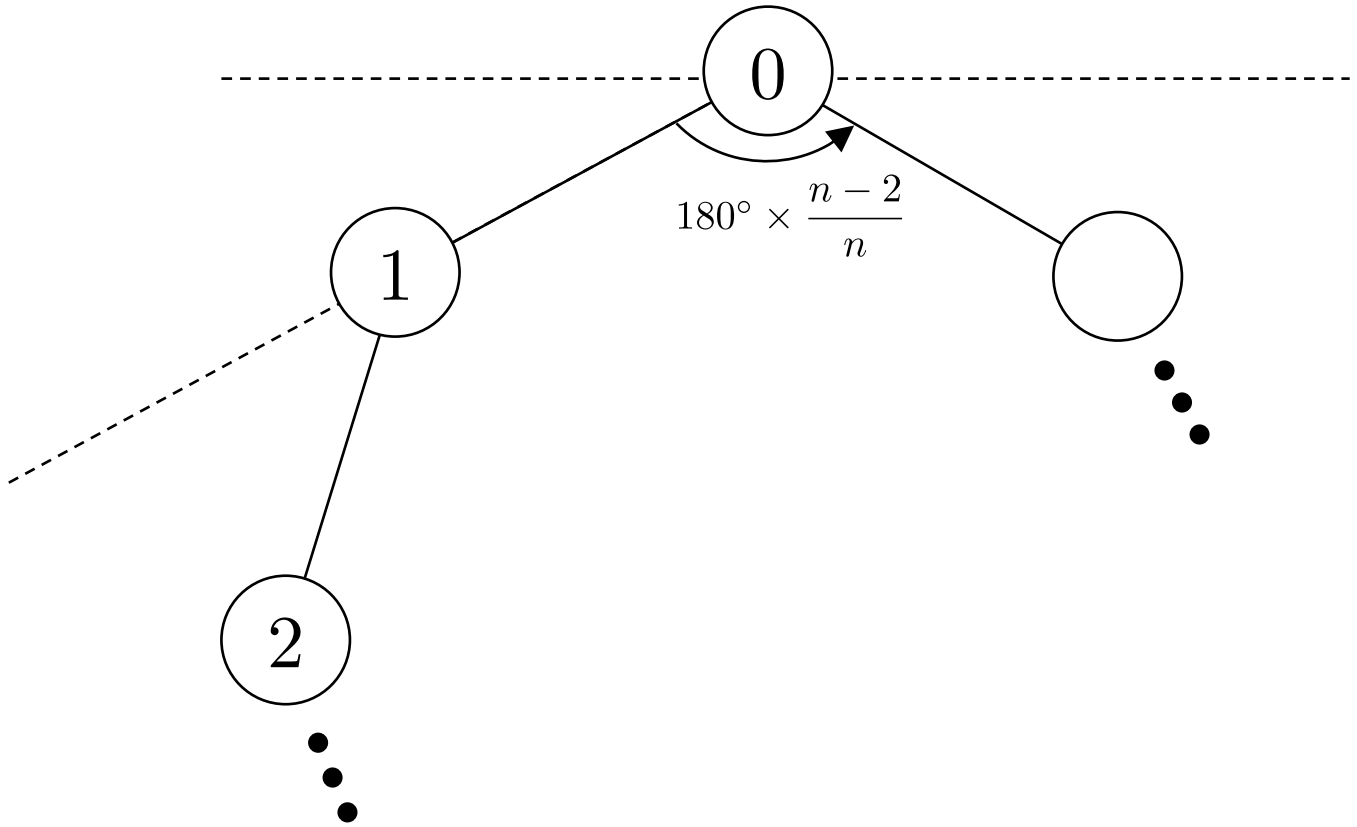
デモ①（円環形のオートマトン）



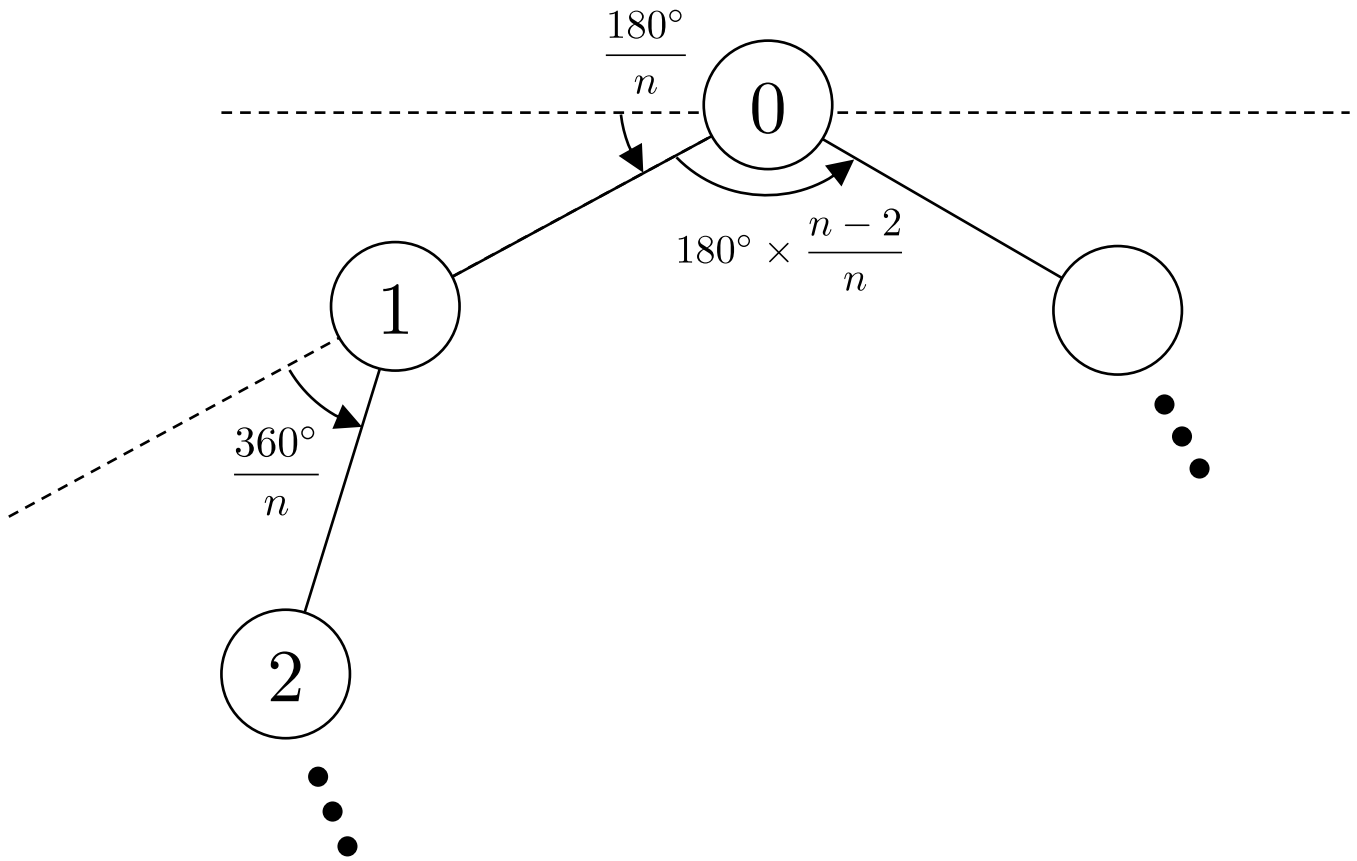
デモ①（円環形のオートマトン）



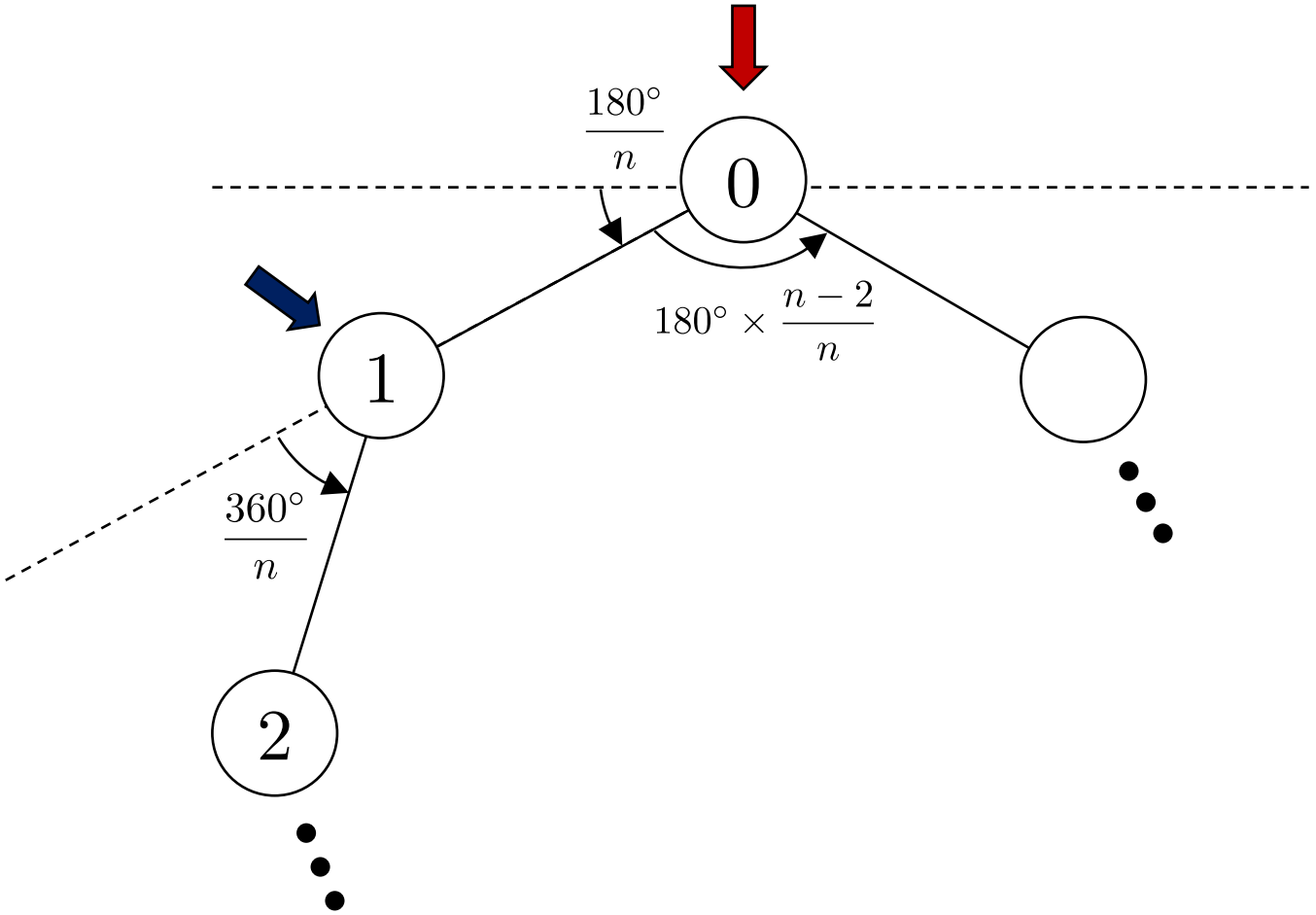
デモ①（円環形のオートマトン）



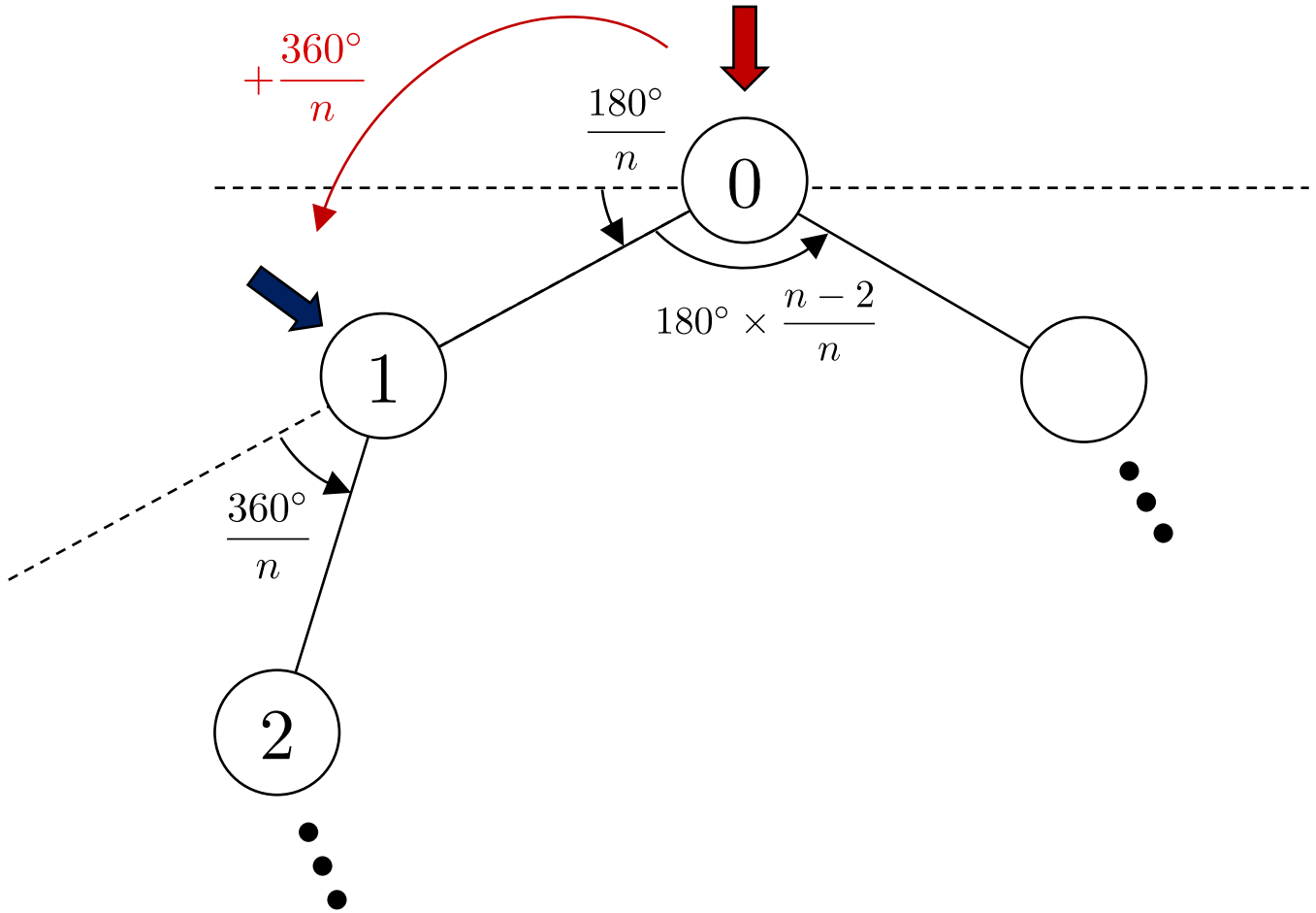
デモ①（円環形のオートマトン）



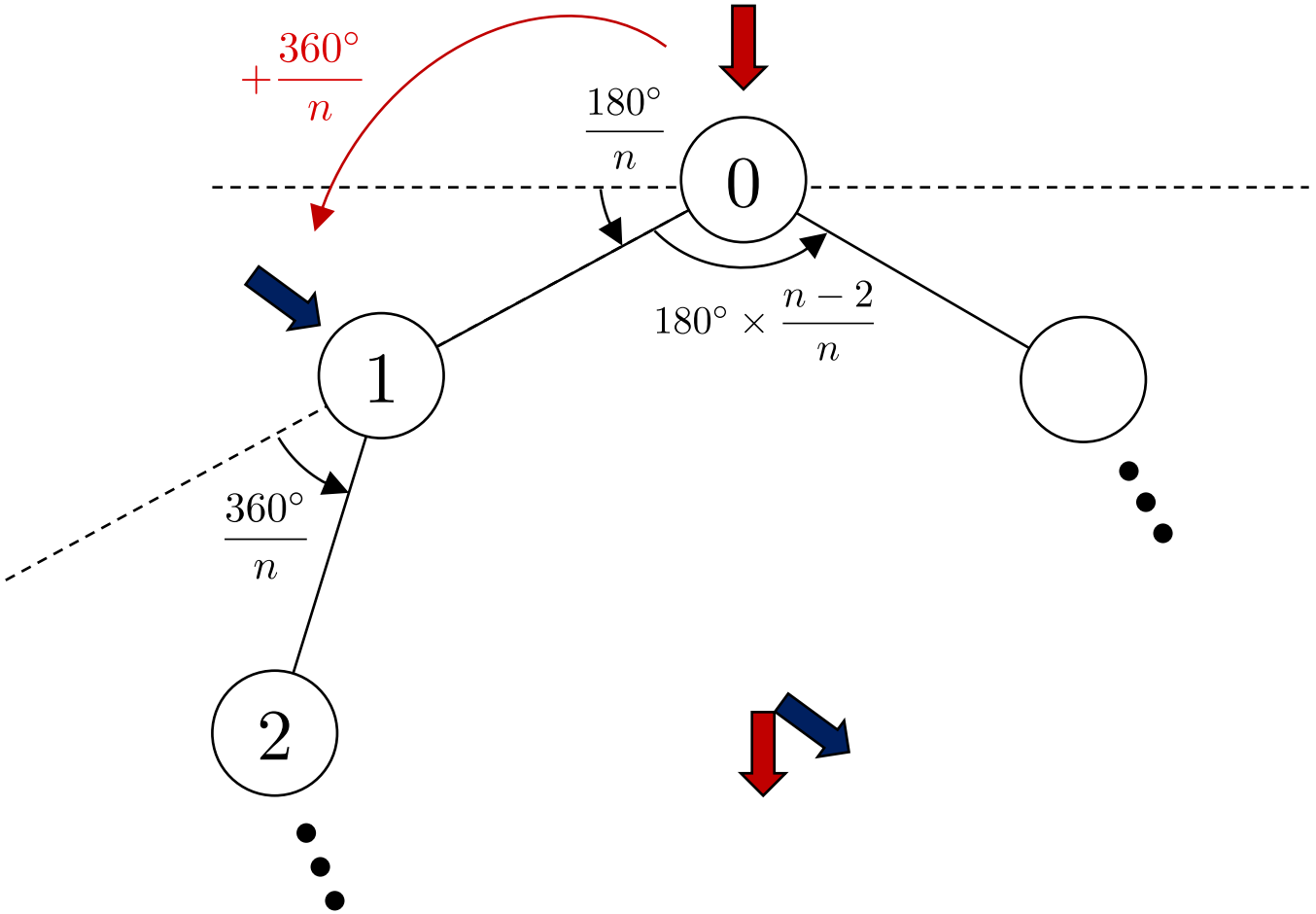
デモ①（円環形のオートマトン）



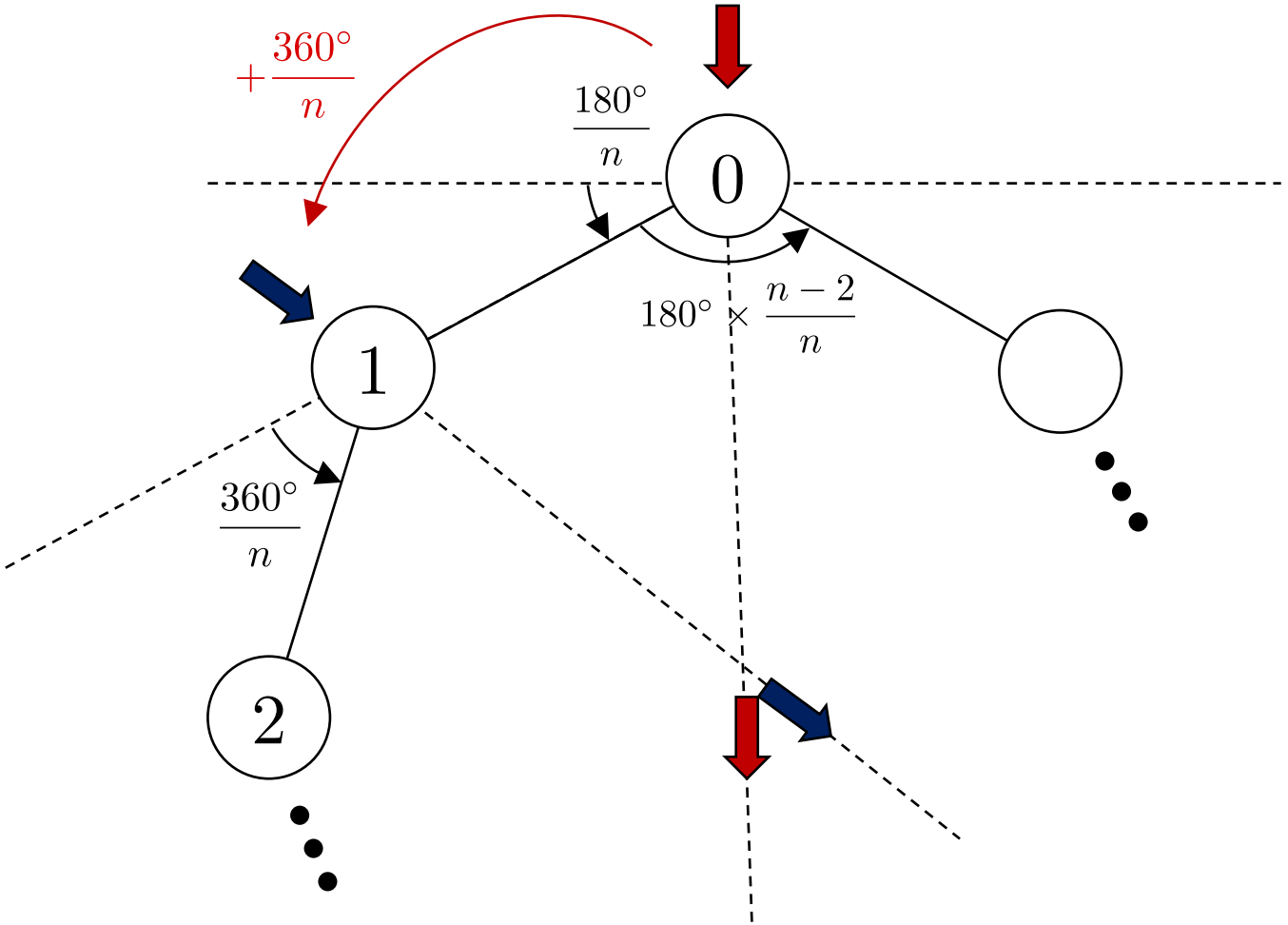
デモ①（円環形のオートマトン）



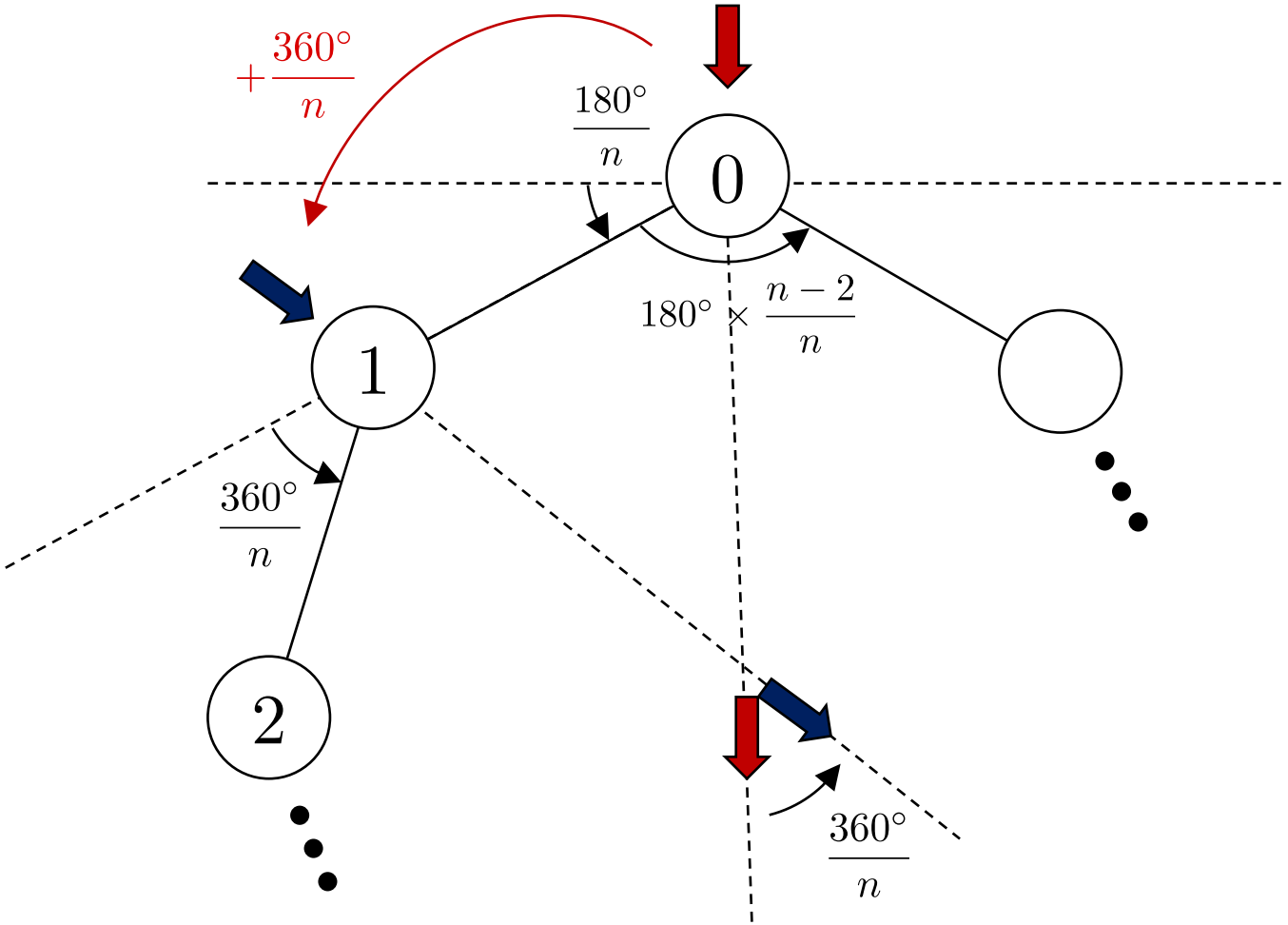
デモ①（円環形のオートマトン）



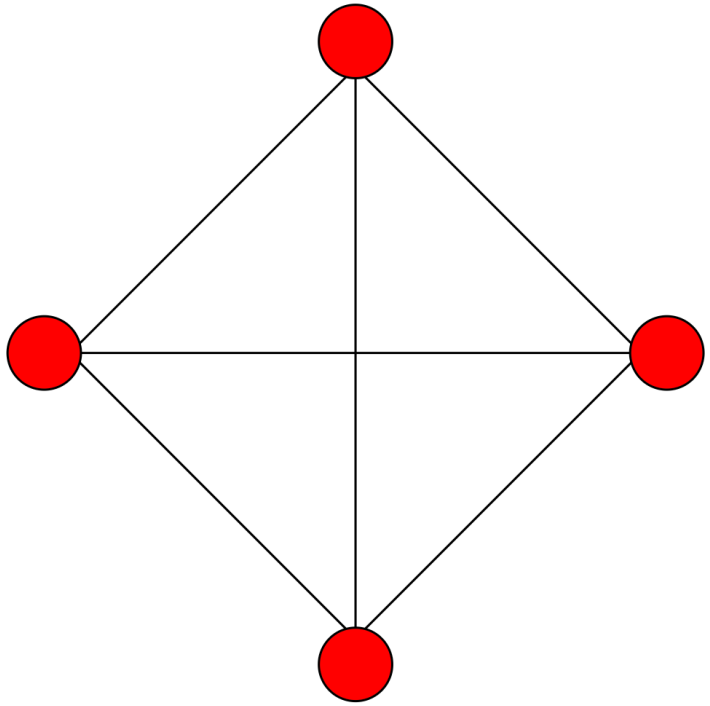
デモ①（円環形のオートマトン）

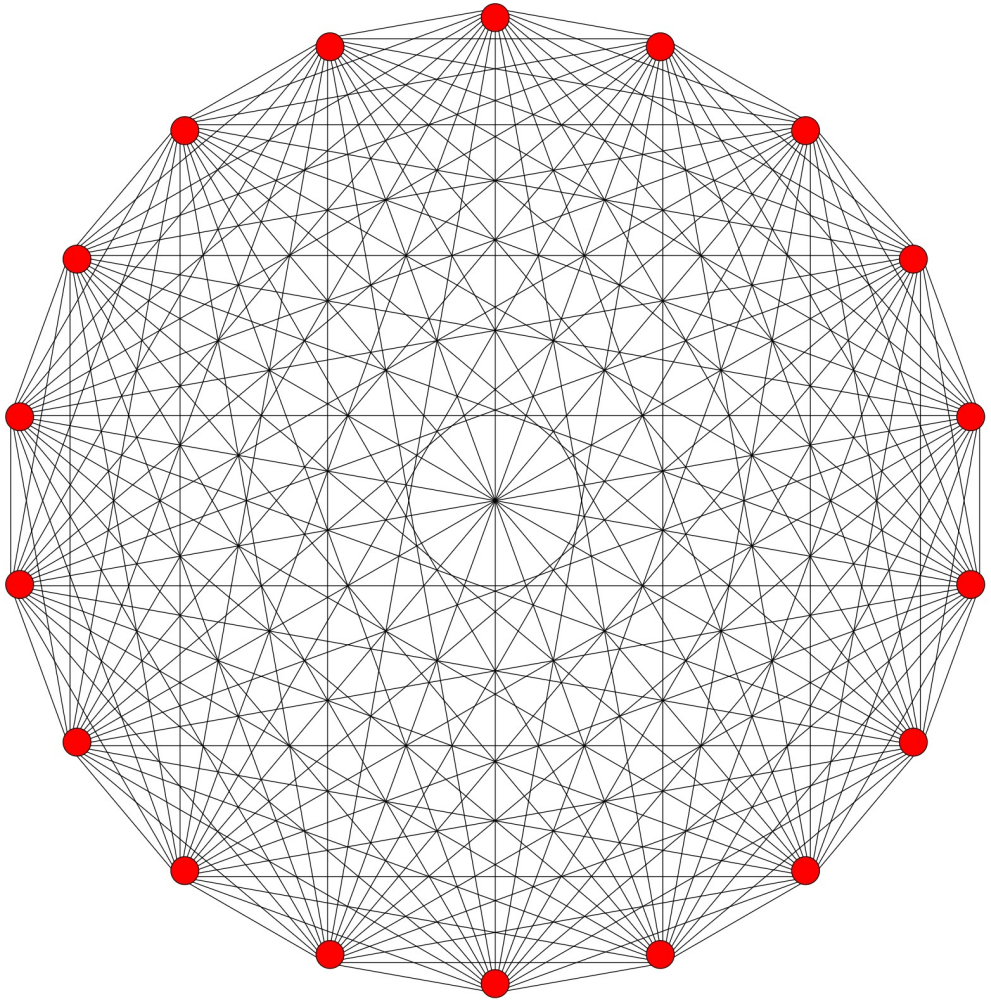


デモ①（円環形のオートマトン）

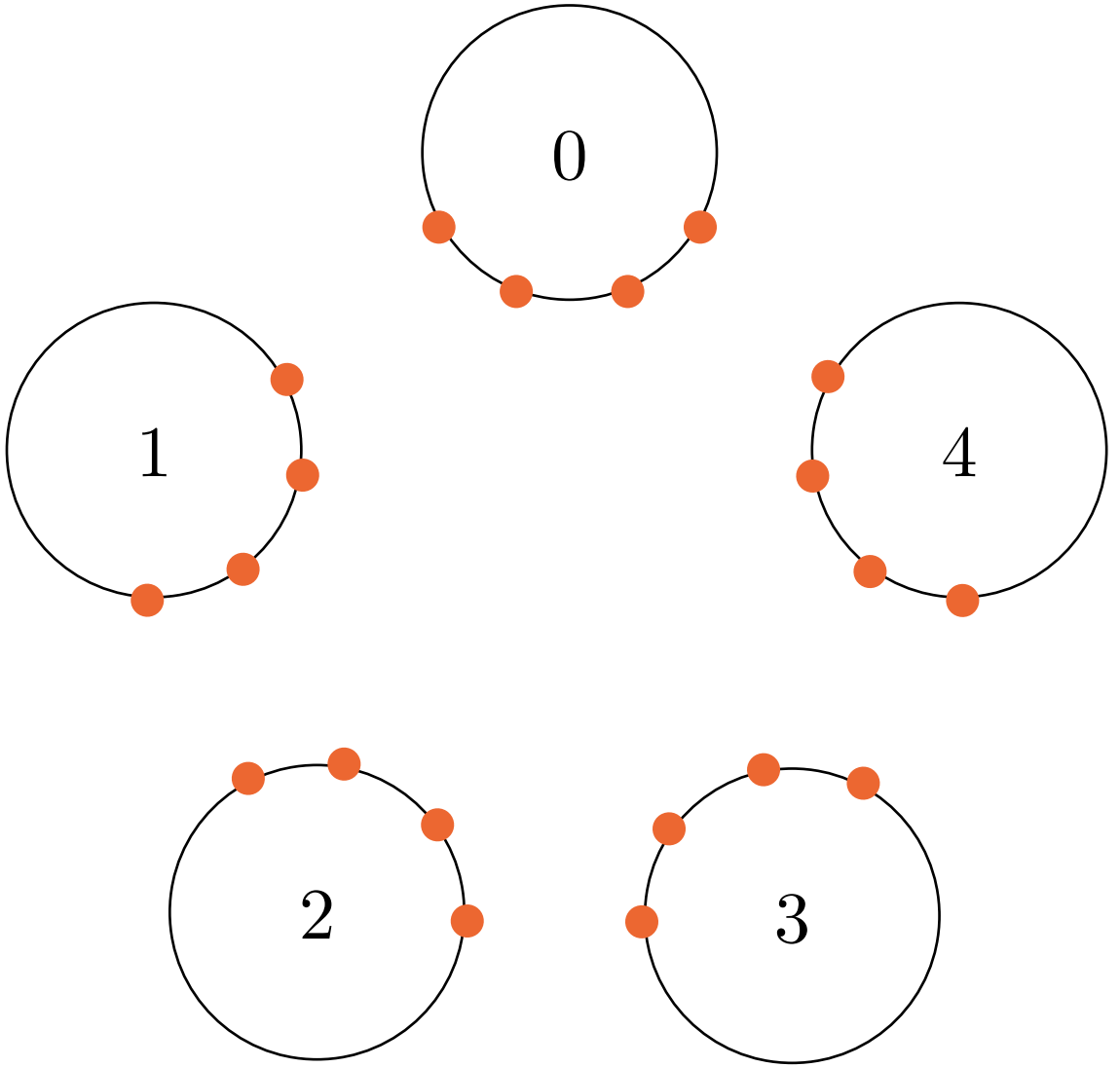


デモ②　（完全グラフ）

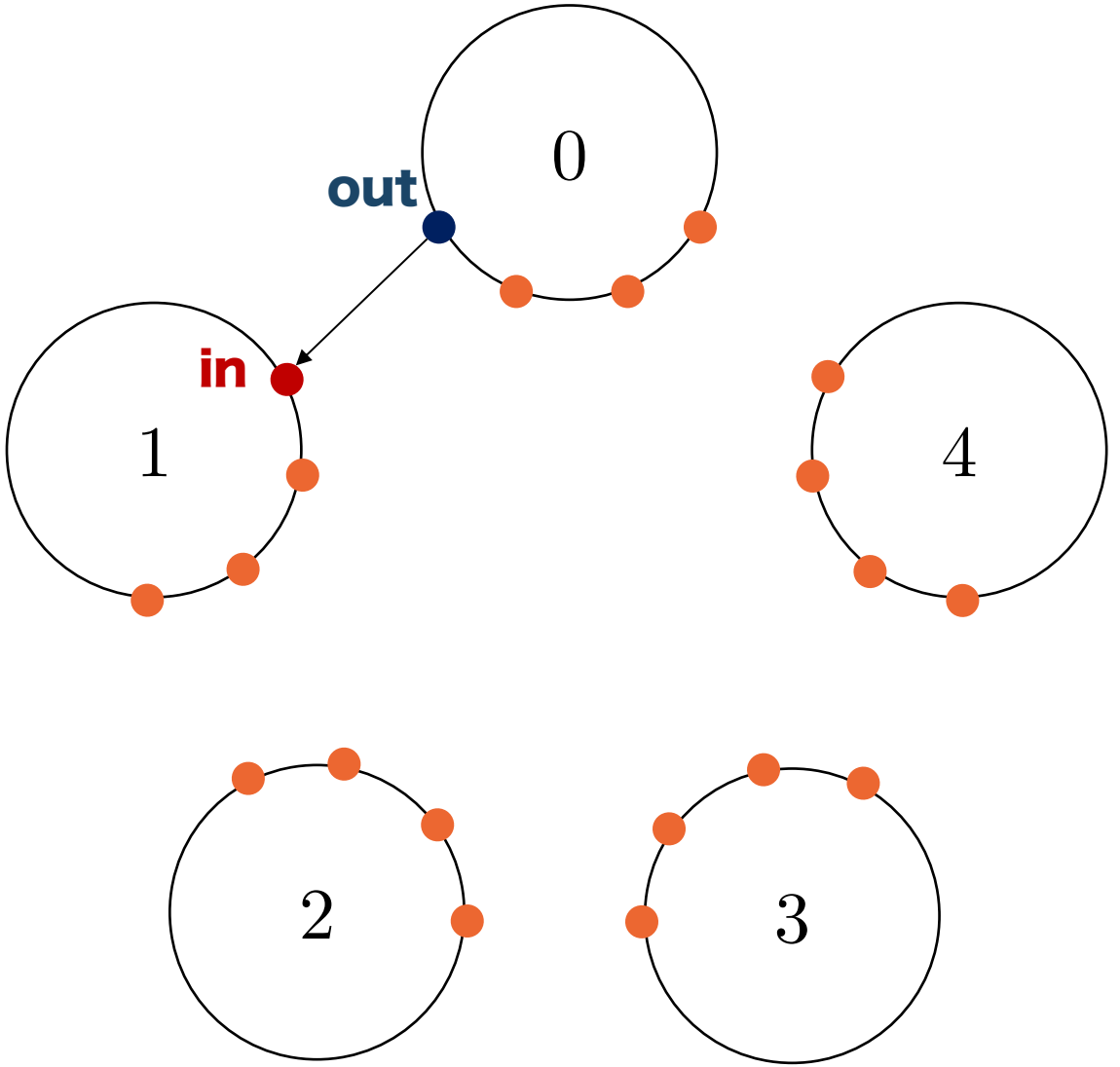




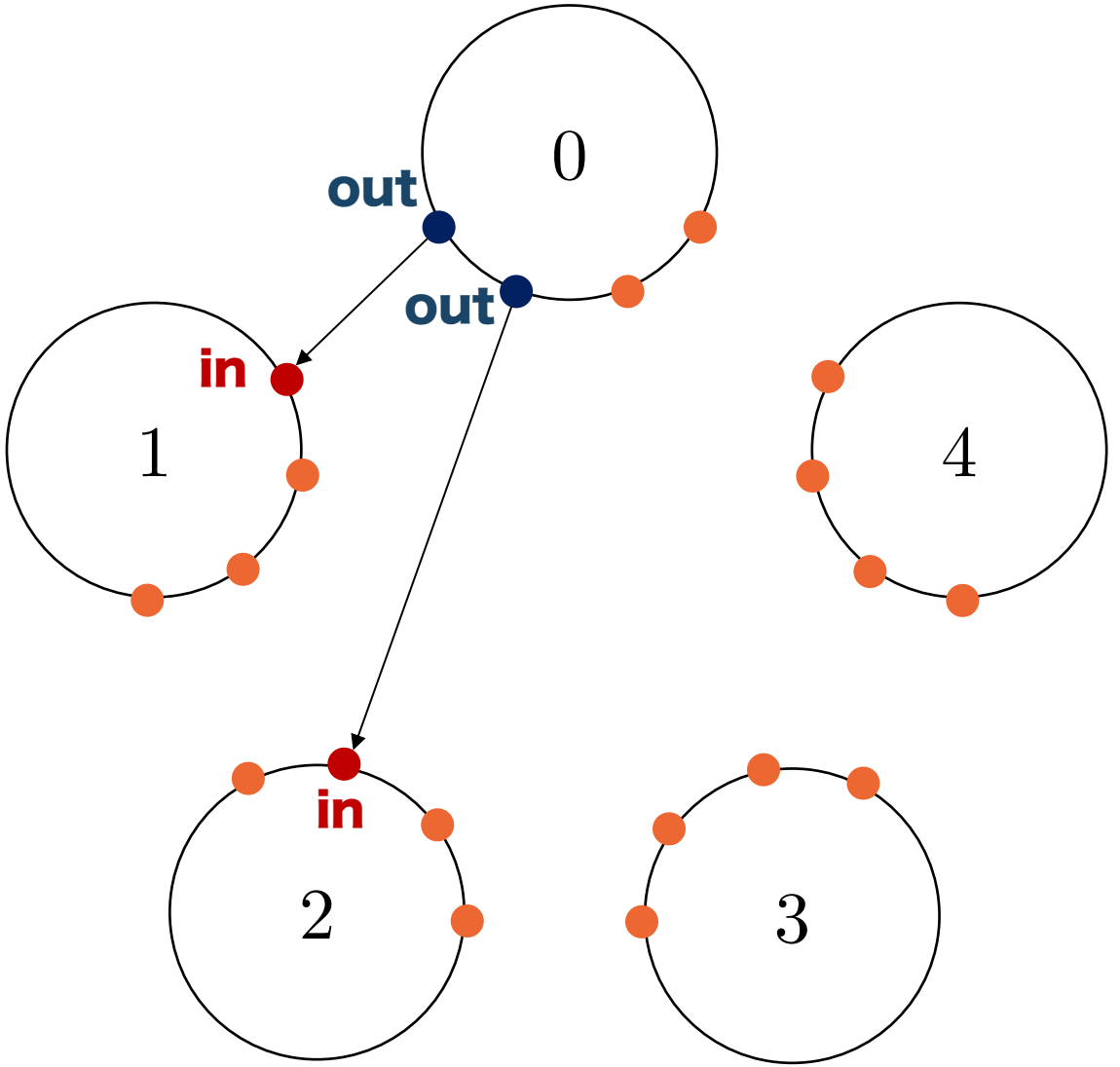
デモ②（完全グラフ）



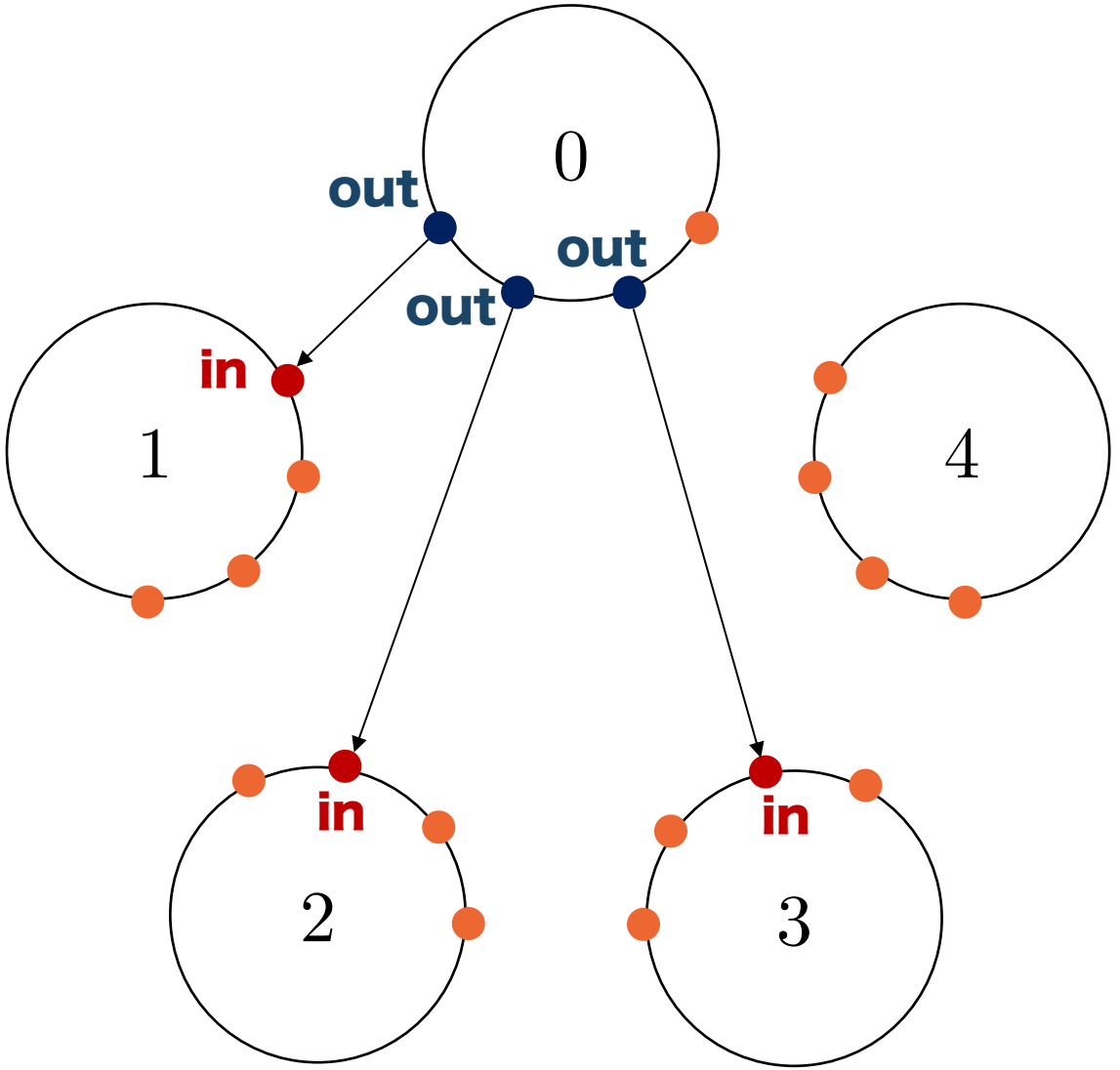
デモ②（完全グラフ）



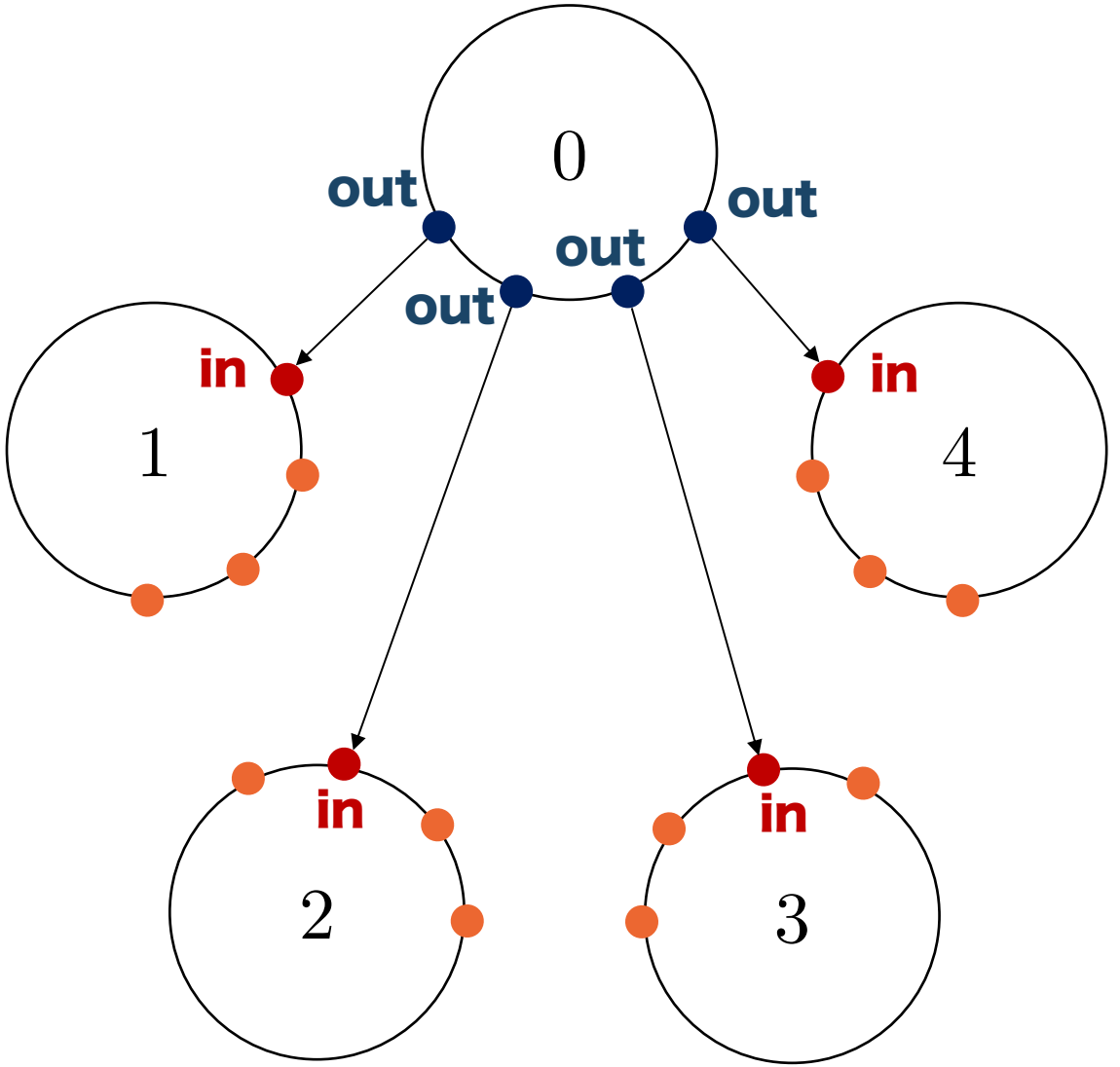
デモ②（完全グラフ）



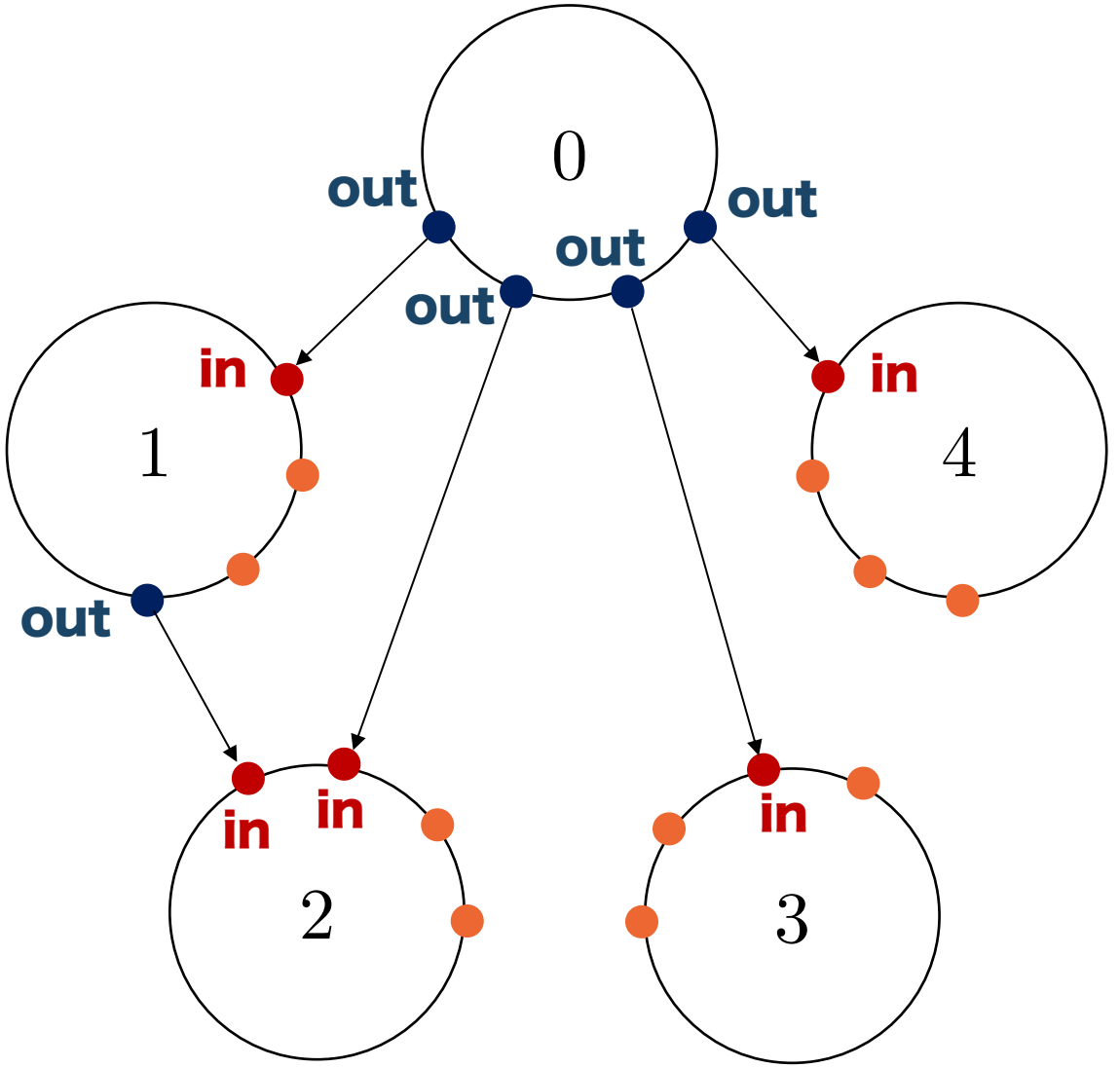
デモ②（完全グラフ）



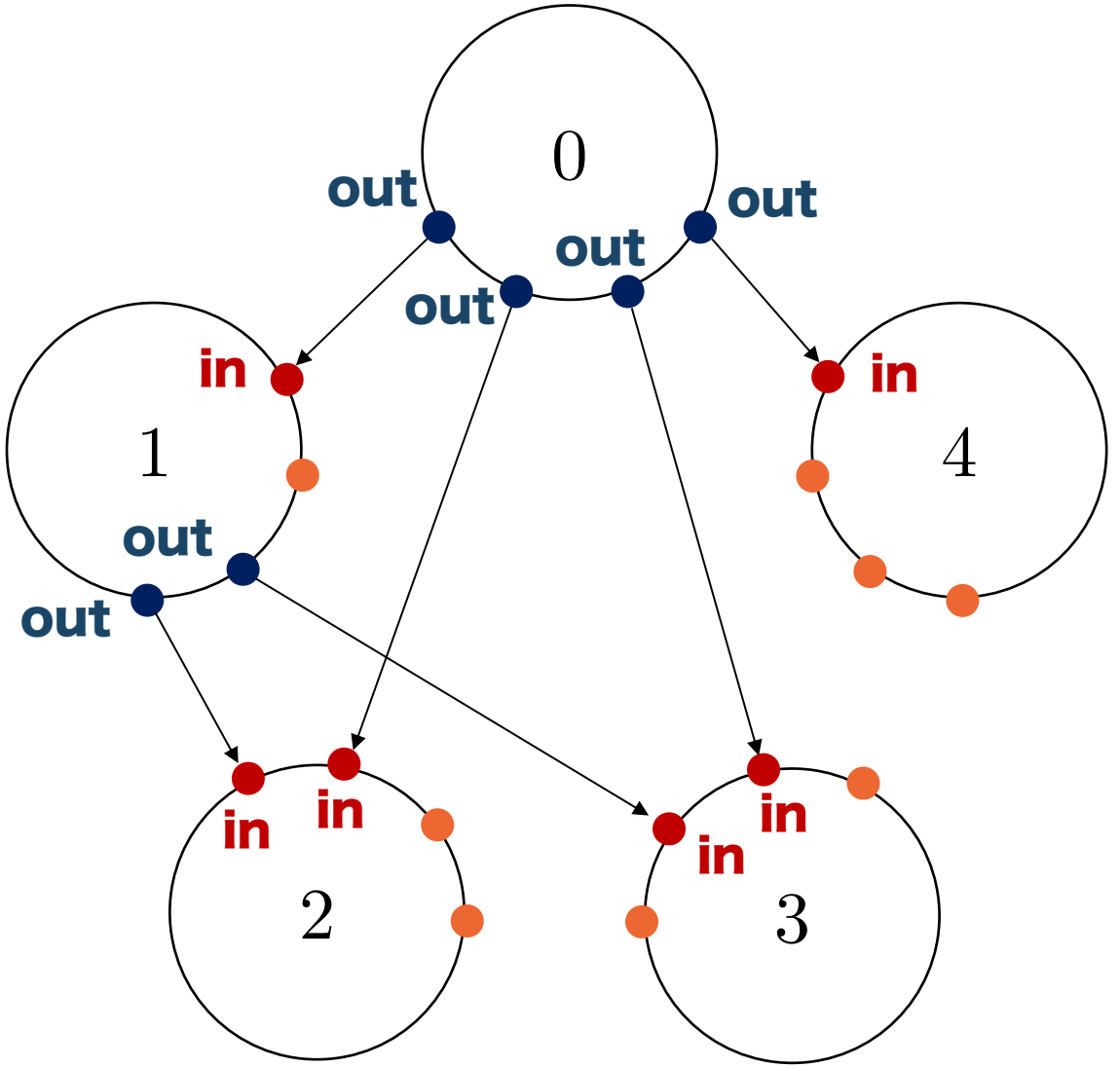
デモ②（完全グラフ）



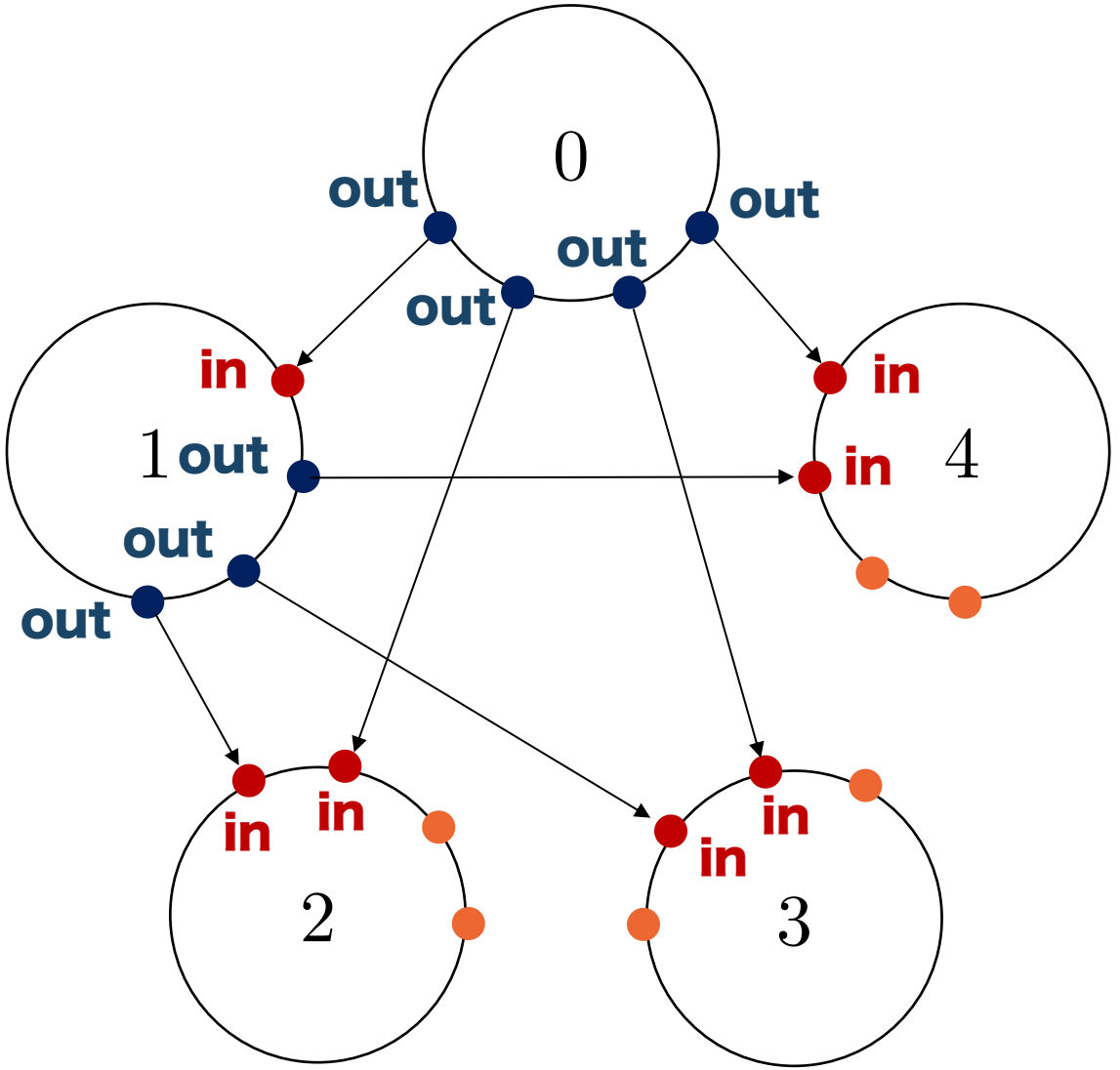
デモ②（完全グラフ）



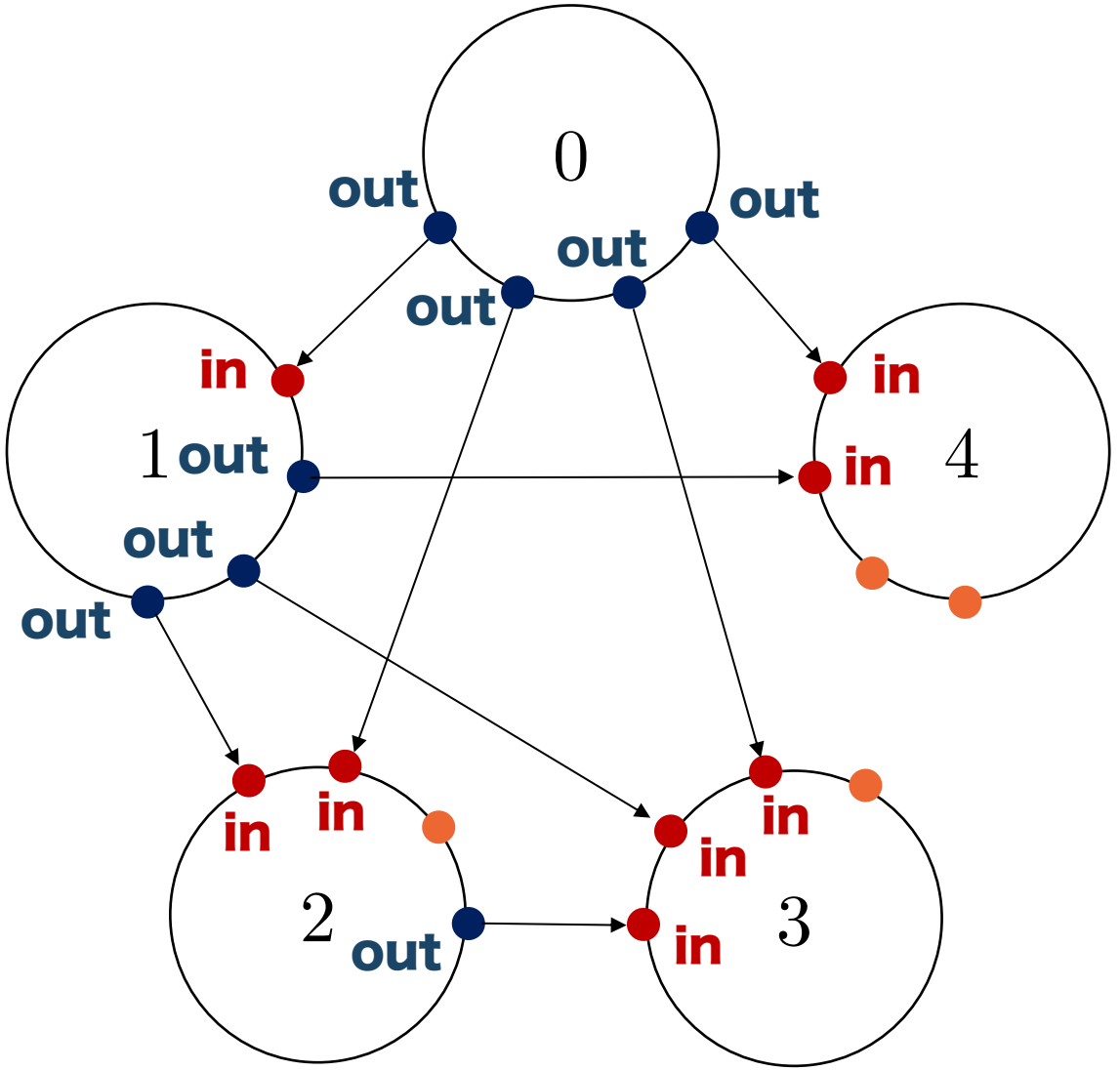
デモ②（完全グラフ）



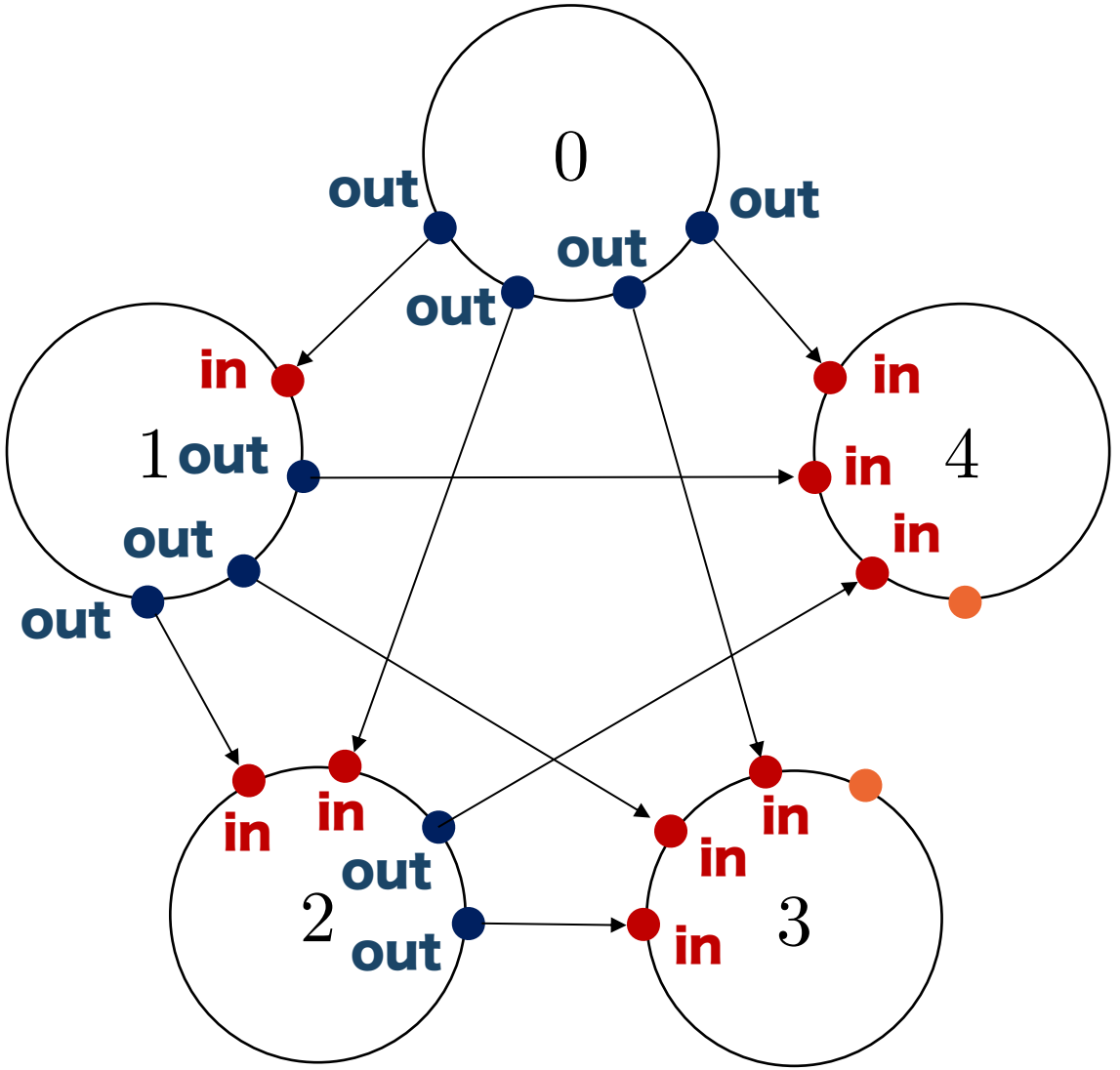
デモ②（完全グラフ）



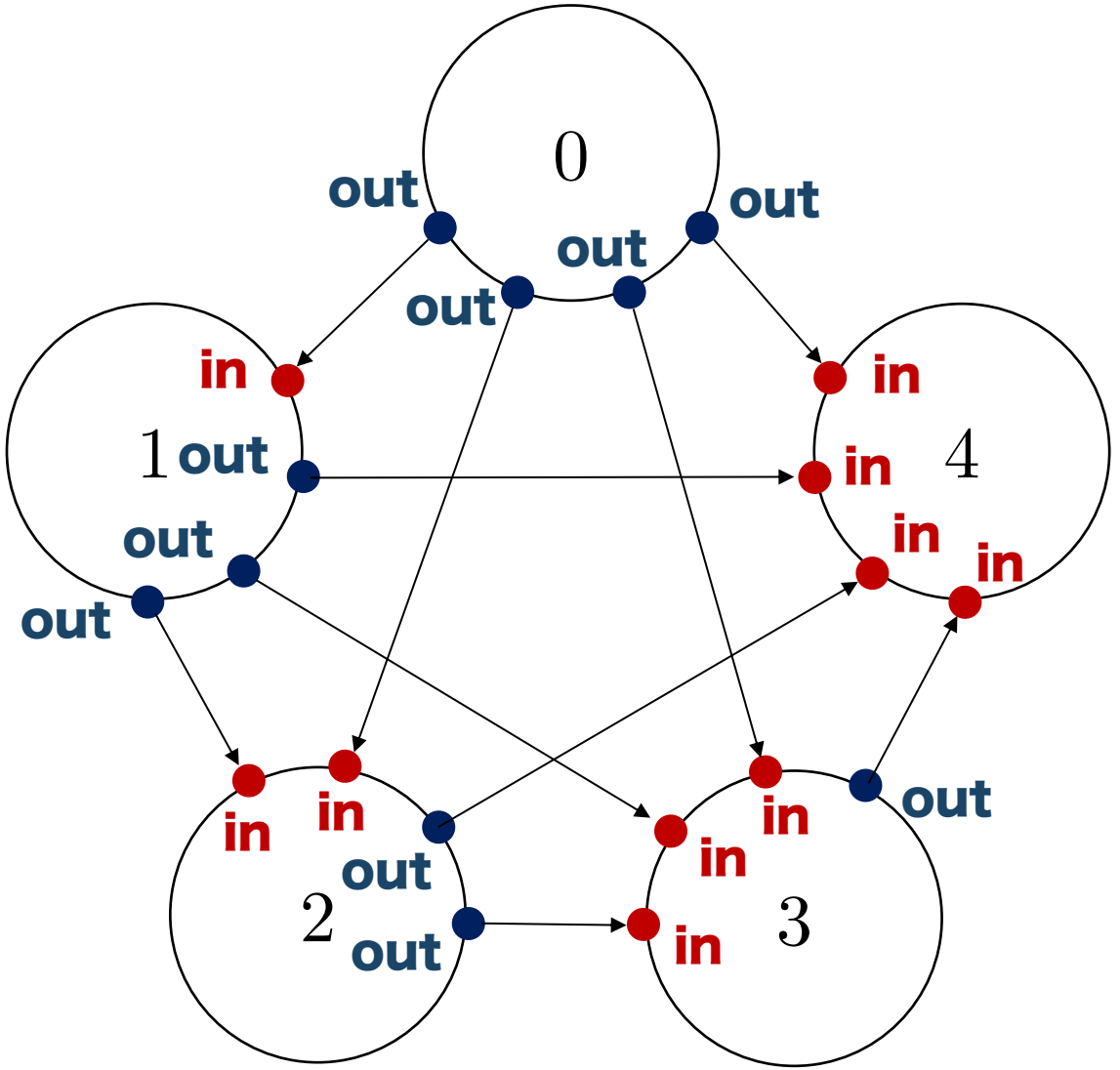
デモ②（完全グラフ）



デモ②（完全グラフ）


0
out
out
out
out
1
in
out
out
out
4
in
in
in
2
in
in
out
out
3
in
in
in

デモ②（完全グラフ）


0
out
out
out
out
in
in
1
out
in
4
out
out
in
in
out
in
in
in
in
out
in
2
out
out
in
3
in

これから

- パスの**マーカ**の定義法の確立
- 図形の**回転**への対応 → **円のみ対応**
- **木構造**を扱えるように
- 図形の概形の**大きさ**を保持して、自動リサイズ
- **原始図形**を増やす → **むしろ限るべきか？**

- SVG以外の形式への対応 → **既存ツールが存在（librsvg）**
- 出力するSVGコードの最適化
- 専用GUIエディタの作成