PRG1-w2a
状態とその表現

脇田建

2019.10.4

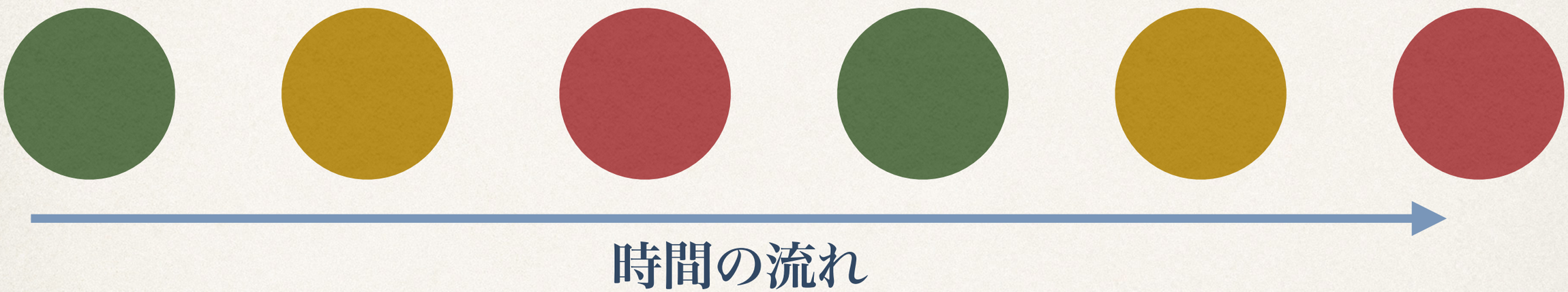
デモ：状態って何？

- タイマー(timer1.scala)
- 信号 (trafficlight.scala)
- オートマトン (automaton.scala)
- 点滅間隔が指定された信号 (scheduledtrafficlight.scala)
- ポトン (poton.scala)

状態って何でしょう？

状態の観測

- 信号の場合



- 時間とともに「色」が変化

- 「色」の変化を円の色として観測

状態の観測

- タイマーの場合
- 開始からの経過秒数 (1.23) が激しく変化している
- **経過秒数**が**ウィンドウにデジタル表示される様子**を観測

状態の観測

- オートマトンの場合

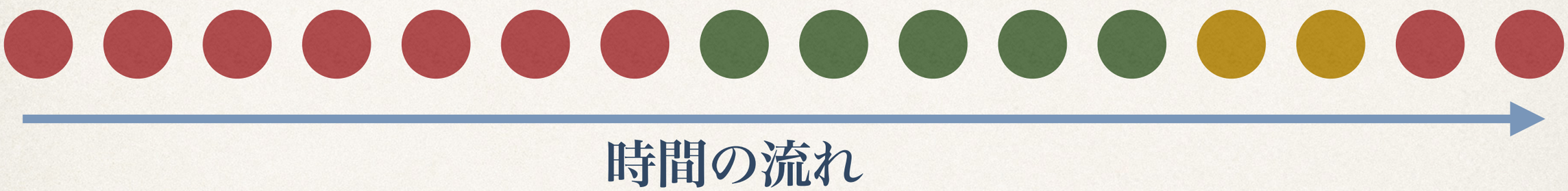
状態	入力	次状態
q	[1]100101	q
q	[1]00101	q
q	[0]0101	q0
q0	[1]01	q00
q00	[0]1	q01
q001	[1]	q001
q001	[]	q001

- オートマトンの**状態変数**と(残りの)**入力列**が変化
- **現在の状態変数**の変化を観測

状態の観測

- 点滅間隔が指定された信号の場合

  赤/緑/黄: 7/5/2秒



- 「色」の変化を円の色として観測

- 同じ色が表示されている時間間隔を時刻の経過として認知（あるいは、時計を見ながら確認）

状態の**観測**

- ポトンの場合
- **青い円の縦方向の位置**の変化として観測
- ←/→キーの操作: **青い円の横方向の位置**の変化として観測

状態と状態遷移のモデル化

決定性有限状態オートマトンにおける状態の表現

- DFA: $A = (Q, \Sigma, \delta, q_0, F)$
  - Q: 状態集合、Σ: 文字集合、 q_0 開始状態、F: 受理集合
  - $\delta : Q \times \Sigma \to Q$ 遷移関数
  - 状態遷移: $q \mapsto q'$: $\delta(q, a \in \Sigma) \mapsto q'$
    - q: 今の状態、q': 次の状態、**a**: 入力

タイマーにおける状態の表現

- 状態集合（経過時間）:
  - $T = \{\ 0.00, 0.01, 0.02, \ldots, 1.01, 1.02, \ldots\}$
- 入力文字集合:
  - $\Sigma = \{\text{tick}\}$: 時計のチクタク (0.01秒間隔)
- 状態遷移関数: $\delta(t \in T, \text{tick}) = t + 0.01$

信号における状態の表現

- 状態集合（信号の色の集合）：
  - Q = { 赤、緑、黄 }
- 入力文字集合: Σ = {tick}
- 状態遷移関数:　δ(赤) = 緑, δ(緑) = 黄, δ(黄) = 赤

点滅間隔つき信号における状態の表現 (1)

- 状態集合（信号の色の集合）：
  - Q = { **赤7**、**赤6**、**赤5**、**赤4**、**赤3**、**赤2**、**赤1**、**緑5**、**緑4**、**緑3**、**緑2**、**緑1**、**黄2**、**黄1**}
- 入力文字集合: Σ = {tick}
- 状態遷移関数:
  - δ(**赤1**, tick) = **緑5**,　δ(**緑1**, tick) = **黄2**,　δ(**黄1**, tick) = **赤7**
  - δ(**赤*i*+1**, tick)=**赤*i***,　δ(**緑*i*+1**, tick)=**緑*i***,　δ(**黄*i*+1**, tick)=**黄*i***　for all *i* in [1,7]

点滅間隔つき信号における状態の表現 (2)

- さすがにさっきの表現は状態集合と状態遷移関数が大きくてやっかい
- 状態集合（信号の色の集合）:
  - Q = 色 × 残り時間 = { 赤、緑、黄 } × { 1, 2, …, 7 }
- 入力文字集合: Σ = {tick}
- 状態遷移関数:
  - δ((色, r+1), tick) = (色, r)
  - δ((色, 1), tick) = (next(色), r), where next(赤 / 緑 / 黄) = 緑 / 黄 / 赤

ポトンにおける状態の表現

- 状態集合: $Q = X \times Y$ (円の位置を表す座標)
  - X in [0, Width], Y in [0, Height]
- 入力文字集合: $\Sigma = \{ \text{tick}, \leftarrow, \rightarrow \}$
- $\delta((x, y), \text{tick}) = (x, y+1)$
- $\delta((x, y), \leftarrow) = (x-1, y)$ / $\delta((x, y), \rightarrow) = (x+1, y)$

状態遷移の実現方法

サンプルプログラムをよく学んで下さい

状態遷移コードの概略（タイマー）

- 状態

  case class **TimerWorld1**(**t: Int**) extends World() { … }

- 初期状態

  TimerWorld1(**0**).bigBang(50, 50, 0.01)

- 時間の経過 (tick) – 状態遷移

  def tick(): World = { **TimerWorld1(t + 1)** }

- 時刻の観測: draw 関数

状態遷移コードの概略（信号）

- 状態: case class **Light**(**color: Color**) extends World() { … }
- 初期状態: **Light**(**Red**).bigBang(300, 300, 2)
- 時間の経過 – 状態遷移

```
def tick(): World = {
  Light(color match {
   case Red => Green        case Green => Yellow        case Yellow => Red
  })
 }
```

- 時刻の観測: **draw** 関数 – canvas.drawDisk(Pos(150, 150), 120, **color**)

状態遷移コードの概略（点滅信号）

- 状態:

  case class **ScheduledLight**(**color: Color, waitFor: Int**) extends World() { … }

- 初期状態: **ScheduledLight**(**Red, 7**)

- 時間の経過 – 状態遷移

```
def tick(): World = {
  (waitFor, color) match {
    case (0, Red) => ScheduledLight(Green, SECS_FOR_GREEN)
    …
    case _ => ScheduledLight(color, waitFor -1)}
```

- }時刻の観測: draw 関数 – canvas.drawDisk(Pos(150, 150), 120, **color**)

状態遷移コードの概略（ポトン）

- 状態: case class **PotonWorld**(**centerX: Int, centerY: Int**) extends World() { … }
- 初期状態: **PotonWorld**(**WorldWidth / 2, BallRadius * 2**)
- 状態遷移
  - tick(): **PotonWorld**(**centerX, min(centerY + 5, MaxY)**)
  - keyEvent(**key**):

```
key match {
    case "LEFT" | "h" => PotonWorld(max(MinX, centerX - 5), centerY)
    case "RIGHT" | "l" => PotonWorld(min(centerX + 5, MaxX), centerY)
    … }
```

まとめ

- 状態を持つソフトウェアの記述は状態変化を記述した状態遷移関数を用いて表現できる。
- 複雑な状態は状態変数を複合させることで記述できる。
- 時間の経過とインタラクションはそれぞれ時間ステップ(tick)とインタラクションの種類を状態機械への入力と捉えることができる。