

2次元可逆分割セルオートマトンの 可逆プログラミング言語上での クリーン可逆シミュレーション

M2018SE013 矢澤拓海

目次

- はじめに
 - 背景, 関連研究, 目的・課題, アプローチ
- 可逆分割セルオートマトン(1D-RPCA, 2D-RPCA)
- 可逆言語による実現
 - 状態の表現の効率化
- おわりに
 - 結論, 参考文献

研究背景

- セルオートマトン(CA)は様々なシミュレーションに用いられる計算モデル
 - 例: 結晶の成長, 交通モデル
- 可逆計算は様々な分野で活用
 - 計算に関わる消費エネルギーに深い関係
 - RTMなどの計算モデル
 - 観測以外の演算が可逆性をもつ量子計算
- 可逆CA (RCA) = CA + 可逆制約
- 多数の可逆シミュレーション

関連研究: RPCAの可逆言語上でのクリーンな実現

- 一般解法によりRCAは可逆言語上で実現可能*¹
- 有限状相の1D-RPCAは可逆言語上で効率的に実現可能
 - 周期的境界条件 { あり [Moriyama92]
なし [渡邊13*¹][木村,矢澤18]

実装が容易

2次元に適用不可

- 有限状相の2D-RPCAは可逆言語上で効率的には未実現

*¹実行時間に比例するメモリ使用量 ⇒ 非効率

研究目的 と 研究課題

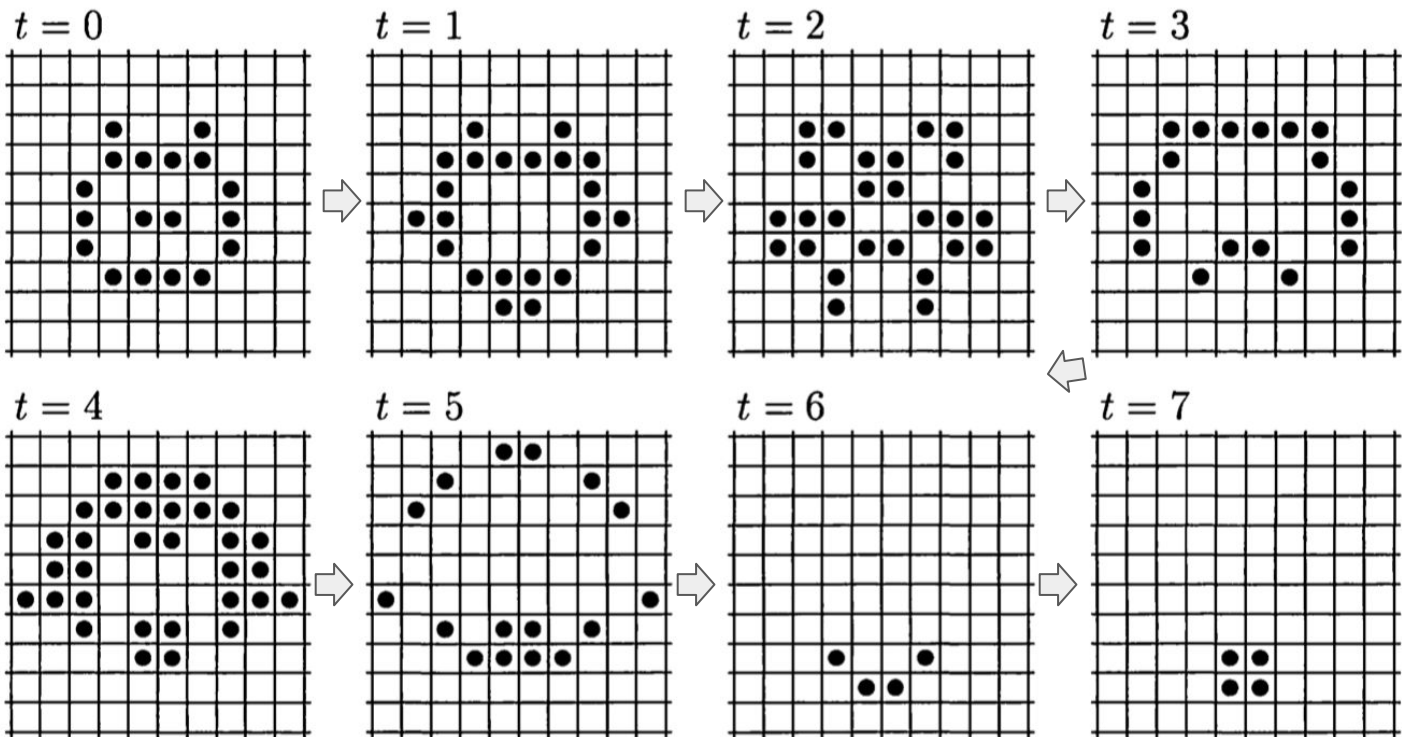
- 研究目的
 - 2D-RPCAの可逆言語上でのクリーンな実現
- 研究課題
 - 任意の2次元有限状相の表現
 - 可逆言語Janusの拡張が必要
 - 2次元状相の表現の単射的更新の実現
 - 効率的な状相の表現が必要

アプローチ

- 無限長の状態の有限なメモリでの表現と単射的更新
 - 1次元の手法を2次元に拡張
- プログラム実行中の2次元配列の拡大
 - 単射性を損なわないようJanusを拡張
- 効率的な状態の表現
 - 入力消去可逆計算[Bennett89]の応用

セルオートマトンの例: GoL

遷移規則: 周囲の8個のセルがちょうど3個なら誕生し, 2個か3個なら生き延びる

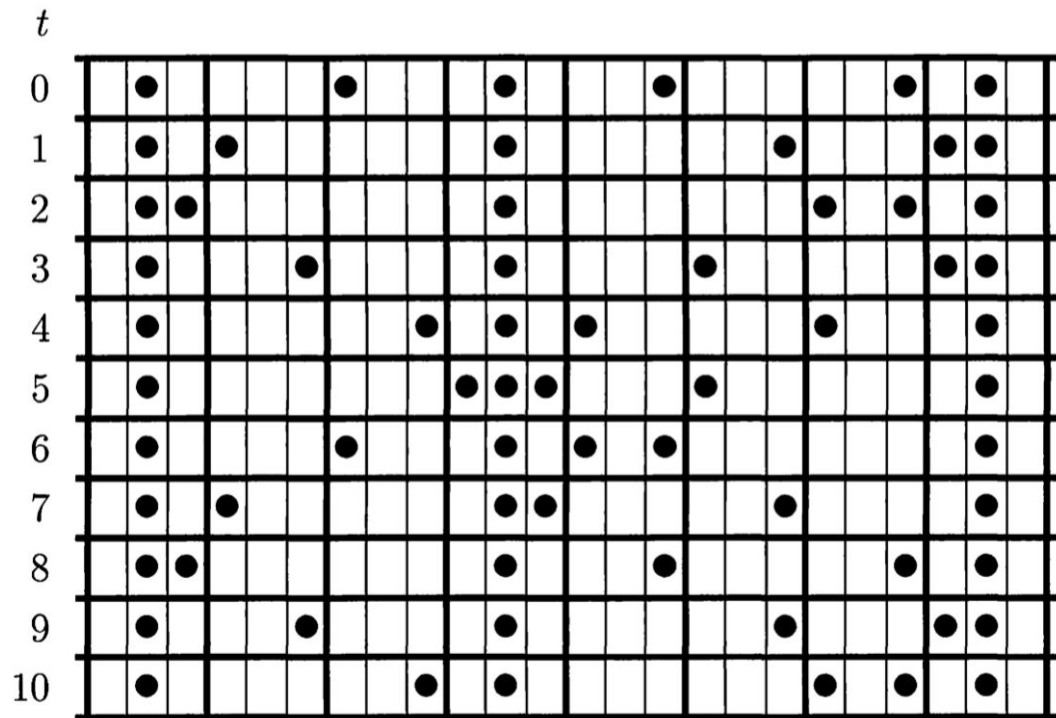


有限オートマトンを規則的に配置・接続した計算システム

2次元状相の遷移は**非単射**

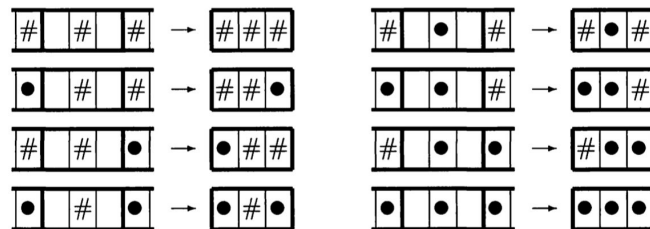
出典: [森田12]

可逆セルオートマトン1D-RPCA P_1 [MoritaHarao89]



出典: [森田12]

遷移規則:

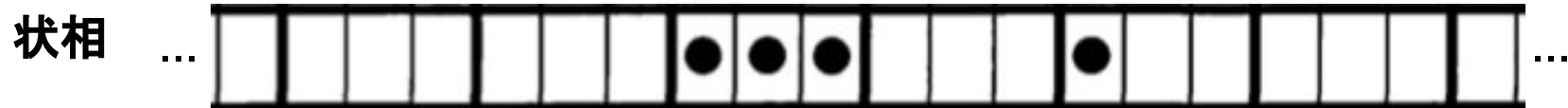


遷移規則は単射



1次元状態の遷移は単射

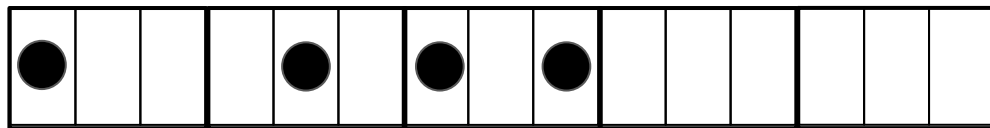
1D-RPCA P_1 の クリーン可逆シミュレーション[渡邊12]



時間発展 $t=0$



$t=1$



$t=2$

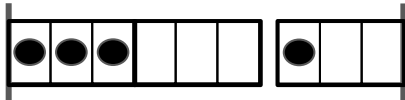


- 遷移ごとにメモリ使用量を2セル分増やす
- 実行時間に比例したメモリ使用量 ⇒ 非効率
- 実装は容易

1D-RPCA P_1 の クリーン可逆シミュレーション[木村, 矢澤18]

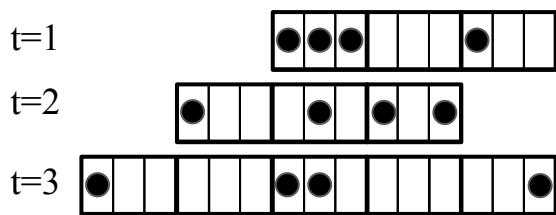
状態 ...  ...

表現



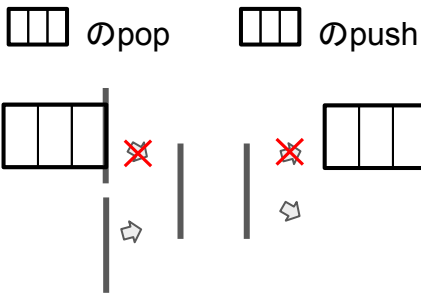
- 任意の1次元有限状態をスタック対で表現可
- メモリ使用量が削減

時間発展



非静止状態の
存在する範囲のみを保持

スタック操作

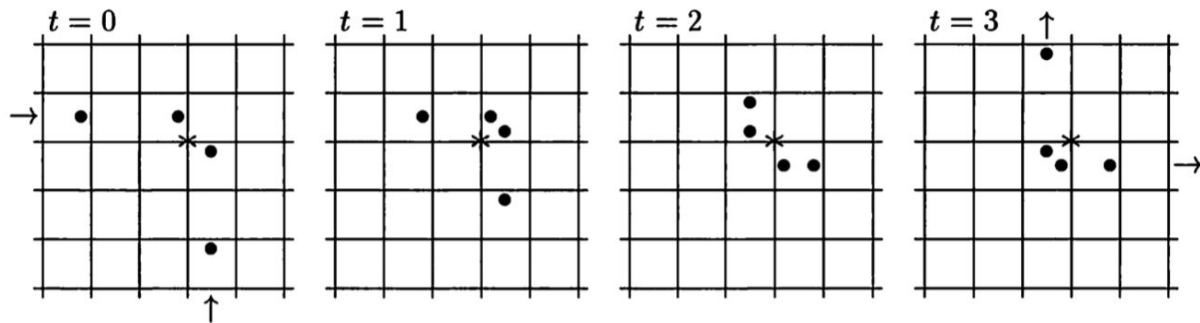


底の静止状態を禁止
⇒ スタック操作の単射化
[YokoyamaAxelsenGlück2016]

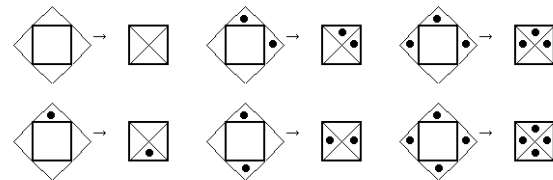
2次元に拡張したい

2D-RPCA P_2 [MoritaUeno1992]

ボール同士の反射のシミュレーション

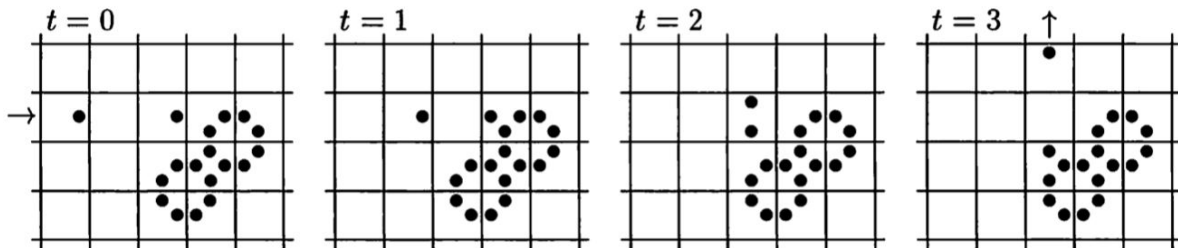


遷移規則:



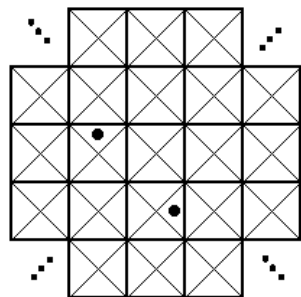
遷移規則は単射

反射板によるボールの反射のシミュレーション



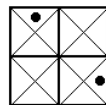
2D-RPCA P_2 の有限状相の表現の提案

有限状相



⇒
整形化

表現



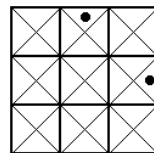
時間発展

[渡邊12]の拡張

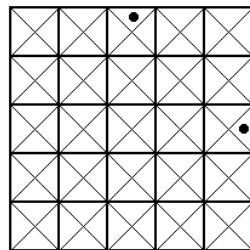
t=0



t=1

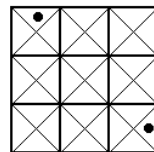
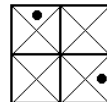


t=2



実行時間の2乗に比例した
メモリ使用量 ⇒ 非効率

[木村, 矢澤 18]の拡張



・2次元配列で整形な表現
・メモリ使用量が削減

目次

- はじめに
 - 背景, 関連研究, 目的・課題
- 可逆分割セルオートマトン(1D-RPCA, 2D-RPCA)
- 可逆言語による実現
 - 状態の表現の効率化
- おわりに
 - 結論, 今後の課題, 参考文献

可逆言語Janus[Lutz86][Yokoyama+08]

- Cに似た構文をもつ手続き型言語
- 可逆性を保つための構文の制約
 - 単純代入は不使用, 可逆条件分岐・可逆繰返し
- プロシージャの逆呼出しが可能: 非通常なモジュール性
- 配列の割付け/解放が入れ子構造
 - 欠点: 実行中に配列の大きさを変えられない ⇒ 要拡張
- スタック付きJanusは r -Turing完全
 - スタック付きJanus言語=単射計算可能関数のクラス

Janusの拡張: 配列の割付け/解放を独立に

構文の拡張

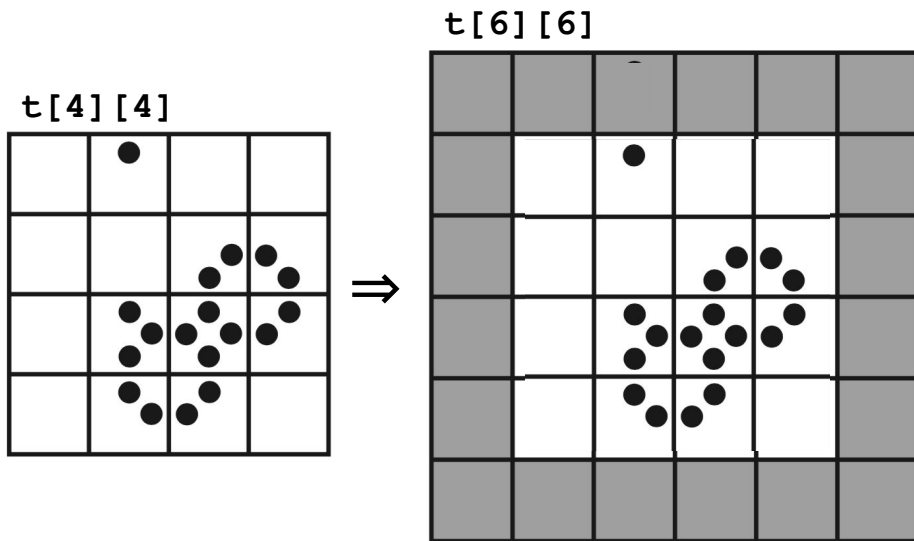
$$\begin{aligned}
 s ::= & x \odot = e \mid x[e] \odot = e \\
 & \text{if } e \text{ then } s \text{ else } s \text{ fi } e \mid \\
 & \text{from } e \text{ do } s \text{ loop } s \text{ until } e \mid \\
 & \text{push}(x, x) \mid \text{pop}(x, x) \mid \\
 & \boxed{\text{local } t x = e \ s \ \text{delocal } t x = e} \mid \boxed{\text{local } t x = e \mid \text{delocal } t x = e} \\
 & \text{call } q(x, \dots, x) \mid \text{uncall } q(x, \dots, x) \mid \\
 & \text{skip} \mid s \ s
 \end{aligned}$$

自然意味論の拡張

$$\begin{array}{c}
 \text{Local} \frac{x \notin \text{dom}(\sigma) \quad \sigma \vdash_{\text{expr}} e_1 \Rightarrow v_1 \quad \sigma \vdash_{\text{expr}} e_2 \Rightarrow v_2}{\sigma \vdash_{\text{stmt}} \text{local } t x[e_1][e_2] = \{\{0\}\} \Rightarrow \sigma[x[0][0] \mapsto 0] \cdots [x[v_1][v_2] \mapsto 0]} \\
 \\
 \text{Delocal} \frac{x \notin \text{dom}(\sigma) \quad \sigma \vdash_{\text{expr}} e_1 \Rightarrow v_1 \quad \sigma \vdash_{\text{expr}} e_2 \Rightarrow v_2}{\sigma[x[0][0] \mapsto 0] \cdots [x[v_1][v_2] \mapsto 0] \vdash_{\text{stmt}} \text{delocal } t x[e_1][e_2] = \{\{0\}\} \Rightarrow \sigma}
 \end{array}$$

可逆性は損なわれない
(厳密には規則帰納法による証明が必要)

配列の割付け/解放の独立 ⇒ 配列の拡大が可能



```
local int t2[6][6] = {{0}}  
call swap(t,t2)
```

```
delocal int t[4][4] = {{0}}  
local int t[6][6] = {{0}}
```

```
call swap(t,t2) 配列の拡大  
delocal int t2[6][6] = {{0}}
```

2D-RPCAの可逆言語上での [渡邊12]の拡張 の実現

```
procedure CA(int t_end, int conf[][], int rule[][])  
  iterate int t = 0 to t_end  
    call gmap(conf, rule)  
  end
```

状態の遷移

図5 2D-RPCA P_2 のクリーン可逆シミュレーション

```
procedure table_expand(int t[][])  
  local int t2[size(t)+2][size(t)+2] = {{0}}  
  call move(t, t2) // zero clear t  
  delocal int t[size(t2)-2][size(t2)-2] = {{0}}  
  local int t[size(t2)][size(t2)] = {{0}}  
  call swap(t2, t) // zero clear t2  
  delocal int t2[size(t)][size(t)] = {{0}}
```

配列の拡大

図7 配列の拡大

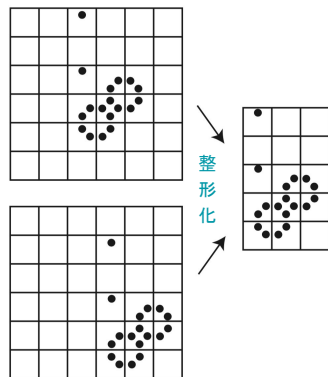
実行時間に比例した
メモリ使用量 ⇒ 非効率

```
procedure gmap(int c1[][], int rule[][])  
  local int c2[size(c1) + 2][size(c1) + 2] = {{0}},  
    int u = 0, int l = 0, int r = 0, int d = 0  
  call table_expand(c1)  
  iterate int y = 0 to size(c2) - 1  
    iterate int x = 0 to size(c2) - 1  
      call get_ulrd(u,l,r,d,c1,x,y)  
      local int t = 0  
      call lmap(t, u, l, r, d, rule)  
      t <=> c2[x][y]  
    delocal int t = 0  
  end  
end  
call swap(c2, c1) // zero clear c2  
delocal int c2[size(c1)][size(c1)] = {{0}}, int u =  
0, int l = 0, int r = 0, int d = 0
```

遷移規則の適用

図6 2D-RPCA P_2 の大域関数

[木村,矢澤18]の拡張の実装に向けて



入力消去可逆計算[Bennett89]を応用

1. ①で α_1 から α_2 とゴミ $_1$ を得る
2. α_2 をコピーする
3. ②で α_2 から α_1 とゴミ $_2$ を得る
4. ①の逆で α_2 とゴミ $_1$ から α_1 を得る
5. 2つの α_1 を1つにする
6. ②の逆で α_1 とゴミ $_2$ から α_2 を得る

未実装

目次

- はじめに
 - 背景, 関連研究, 目的・課題
- 可逆分割セルオートマトン(1D-RPCA, 2D-RPCA)
- 可逆言語による実現
 - 状相の表現の効率化
- おわりに
 - 結論, 今後の課題, 参考文献

結論

- 2D-RPCAの可逆言語上での実現
 - 任意の2次元有限状相の表現の提案
 - Janusの構文と自然意味論の**拡張**:
可逆動的表, 割付/解放の独立化
 - 2次元状相の表現の**単射的**更新の提案
 - r -Turing完全性の別証明
- 任意の有限状相の配列を用いた**整形**表現を提案

参考文献

- [1] Moriyama, K.: Reversible Cellular Automata from a Programming Language Perspective, Master's thesis, DIKU, University of Copenhagen (2010).
- [2] 渡邊恭平: 可逆スタックを用いた可逆セル・オートマトンのクリーン可逆シミュレーション, 南山大学2013年度卒業論文(2014).
- [3] 木村孝大, 矢澤拓海: 1次元可逆セル・オートマトンのクリーン可逆シミュレーションの実現, 南山大学2017年度卒業論文(2018).
- [4] Lutz, C.: Janus: A time-reversible language, Letter to R. Landauer (1986).
- [5] Yokoyama, T., Axelsen, H.B. and Glück, R.: Principles of a Reversible Programming Language, *Proc.CF*, ACM Press, pp.43–54 (2008).
- [6] 森田憲一: 可逆計算, 近代科学社 (2012).
- [7] Bennett, C.H.: Time/space trade-offs for reversible computation, Vol.18, No.4, pp.766–776 (1989).
- [8] Morita, K., Harao, M.: Computation Universality of one-dimensional reversible (injective) cellular automata, *Trans. IEICE Japan*, E72:758–762 (1989).