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
  - 状相の表現の効率化
- 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 [渡邉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

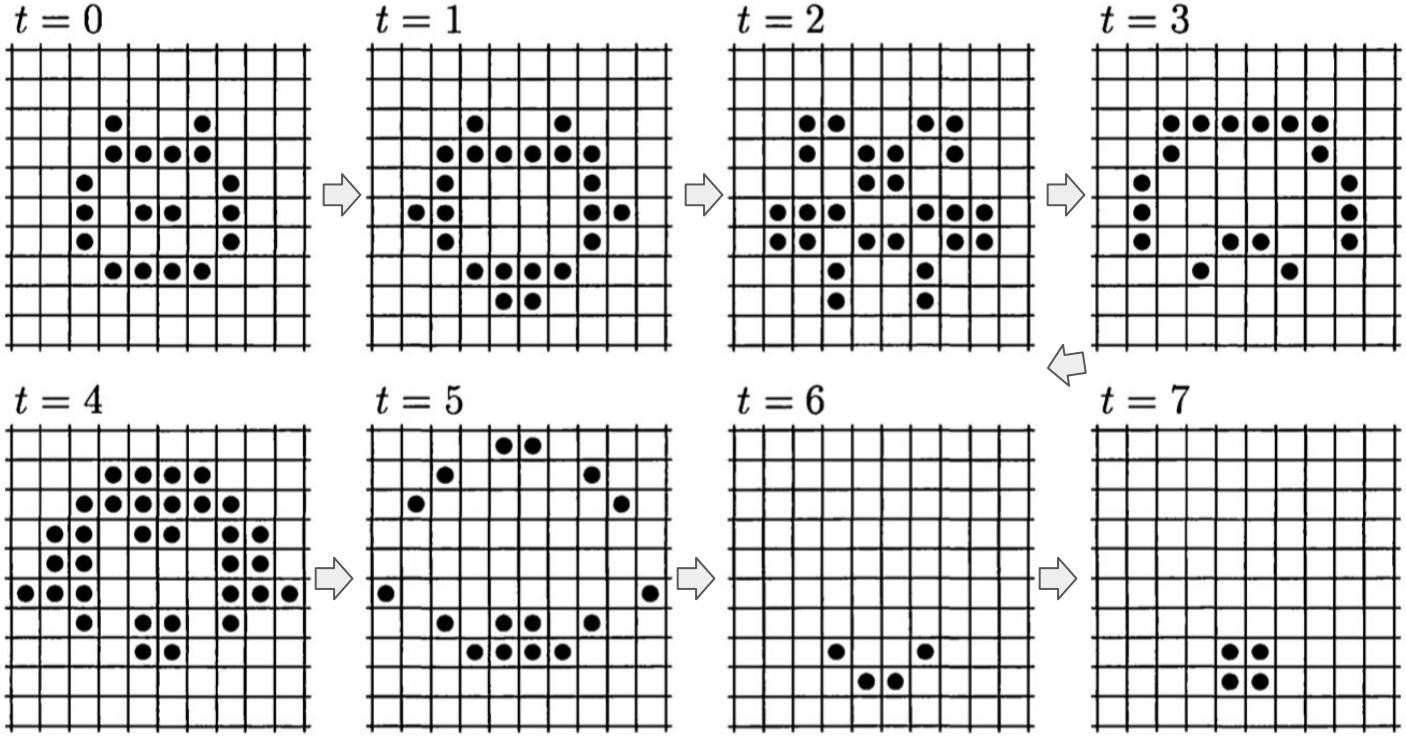
- 研究目的
  - 2D-RPCAの可逆言語上でのクリーンな実現
- 研究課題
  - 任意の2次元有限状相の表現
    - 可逆言語Janusの拡張が必要
  - 2次元状相の表現の単射的更新の実現
    - 効率的な状相の表現が必要

アプローチ

- 無限長の状相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
- 効率的な状相の表現
  - 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

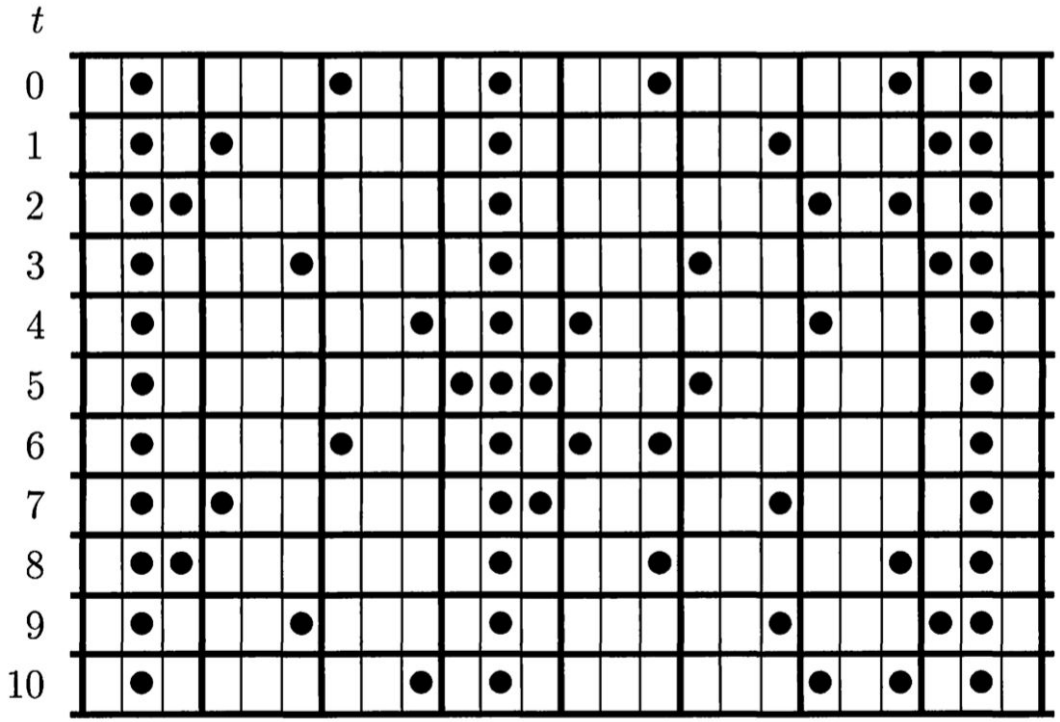


有限オートマトンを規則的に配置・接続した計算システム

2次元状相の遷移は非単射

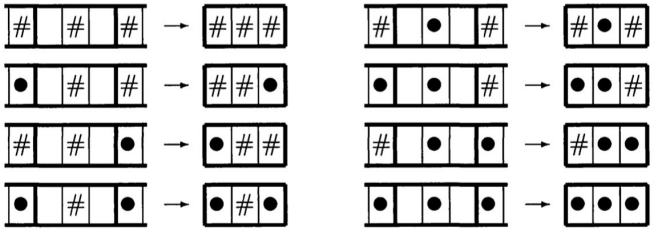
出典：[森田12]

可逆セルオートマトン1D-RPCA P_1[MoritaHarao89]



出典：[森田12]

遷移規則：



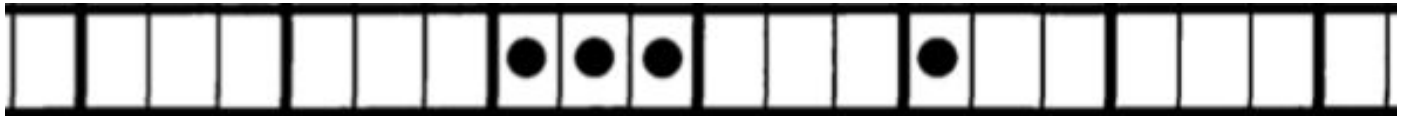
遷移規則は単射

⇓

1次元状相の遷移は単射

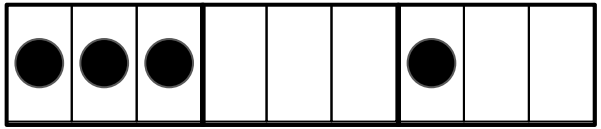
1D-RPCA P_1の クリーン可逆シミュレーション[渡邉12]

状相

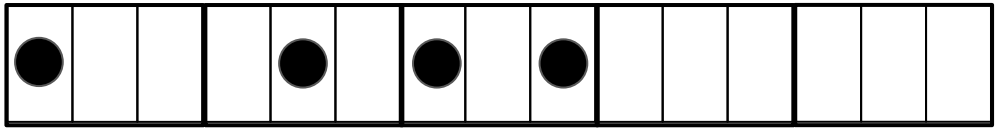


時間発展

t=0



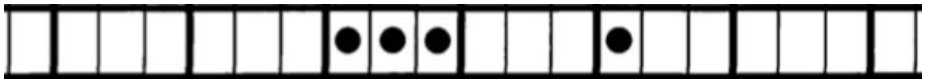
t=1

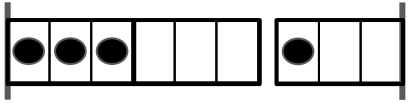


t=2

- 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

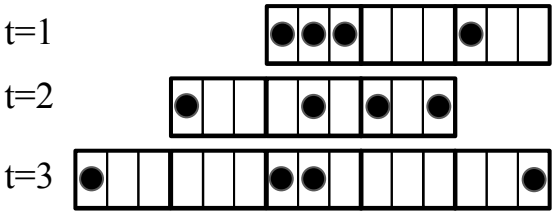
1D-RPCA P_1の クリーン可逆シミュレーション[木村,矢澤18]

状相 …  …

表現 

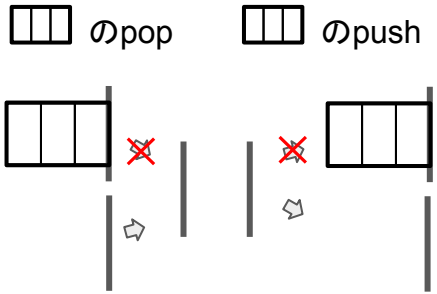
- 任意の1次元有限状相を スタック対で表現可
- メモリ使用量が削減

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

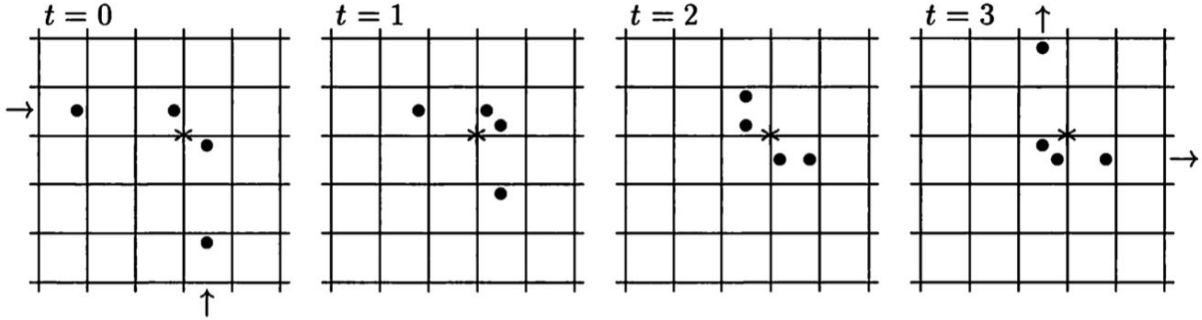


底の静止状態を禁止
⇒ スタック操作の単射化
[YokoyamaAxelsenGlück2016]

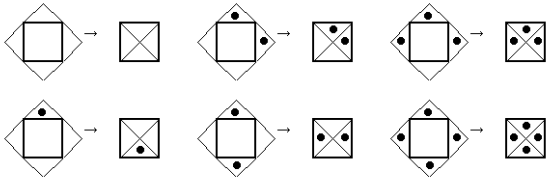
2次元に拡張したい

2D-RPCA P_2 [MoritaUeno1992]

ボール同士の反射のシミュレーション

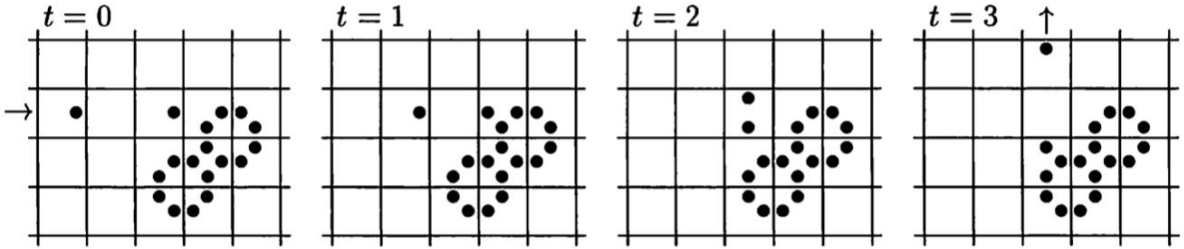


遷移規則:



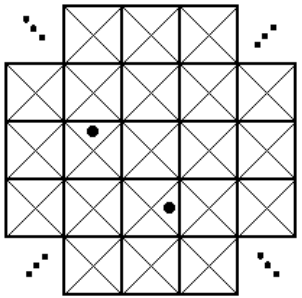
遷移規則は単射

反射板によるボールの反射のシミュレーション



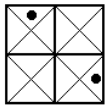
2D-RPCA P_2 の有限状相の表現の提案

有限状相



⇒ 整形化

表現



時間発展

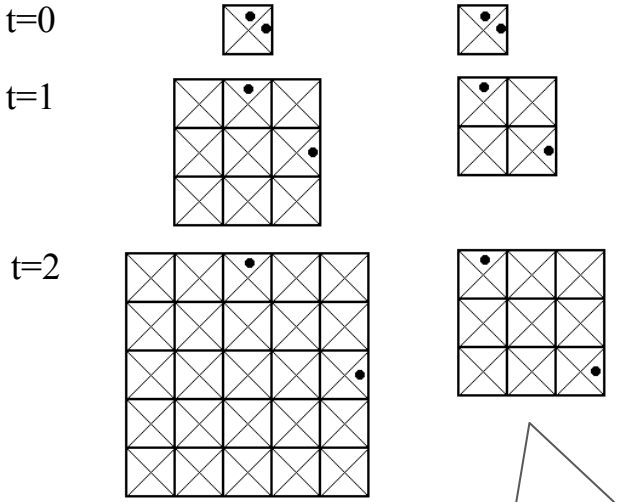
[渡邉12]の拡張

[木村,矢澤18]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

・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
  - 状相の表現の効率化
- 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

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

$$\texttt{local } t\ x = e \mid \texttt{delocal } t\ x = e$$

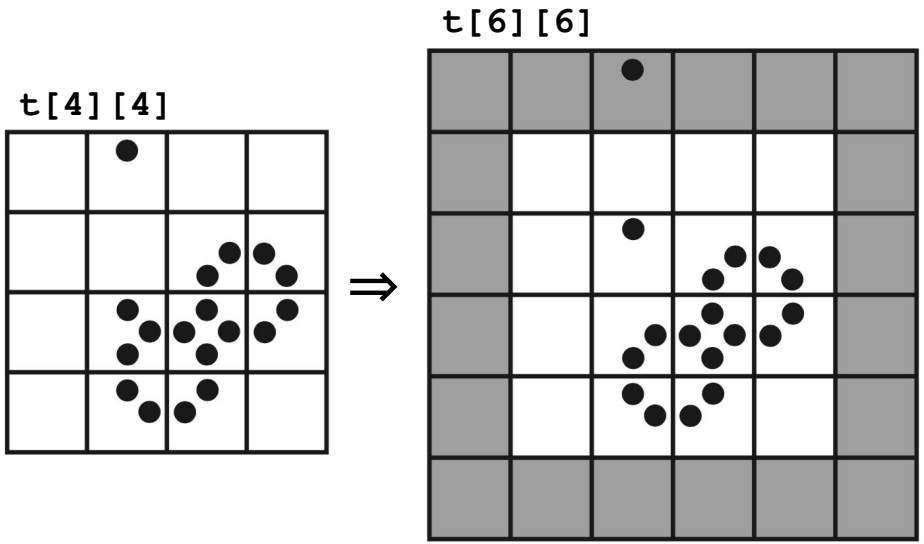
自然意味論の拡張

$$\text{Local}\ \frac{x \notin \mathrm{dom}(\sigma) \qquad \sigma \vdash_{expr} e_1 \Rightarrow v_1 \qquad \sigma \vdash_{expr} e_2 \Rightarrow v_2}{\sigma \vdash_{stmt} \texttt{local } t\ x[e_1][e_2] = \{\{0\}\} \Rightarrow \sigma[x[0][0] \mapsto 0] \cdots [x[v_1][v_2] \mapsto 0]}$$

$$\text{Delocal}\ \frac{x \notin \mathrm{dom}(\sigma) \qquad \sigma \vdash_{expr} e_1 \Rightarrow v_1 \qquad \sigma \vdash_{expr} e_2 \Rightarrow v_2}{\sigma[x[0][0] \mapsto 0] \cdots [x[v_1][v_2] \mapsto 0] \vdash_{stmt} \texttt{delocal } t\ x[e_1][e_2] = \{\{0\}\} \Rightarrow \sigma}$$

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)
delocal int t2[6][6] = {{0}}
```

配列の拡大

2D-RPCAの可逆言語上での [渡邉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
```

状相の遷移

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

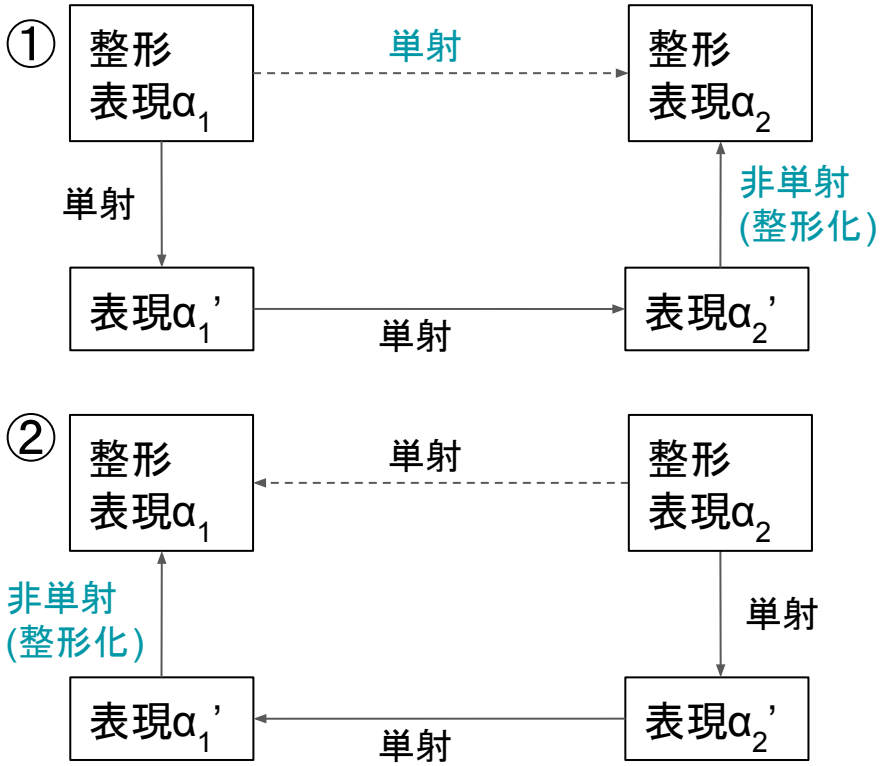
```
procedure gmap(int c1[][], int rule[][])
  local int c2[size(c1) + 2][size(c1) + 2] = {{0}},
      int u = 0, int l = 0, int r = 0, int d = 0
call table_expand(c1)
iterate int y = 0 to size(c2) - 1
  iterate int x = 0 to size(c2) - 1
    call get_ulrd(u,l,r,d,c1,x,y)
    local int t = 0
      call 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
```

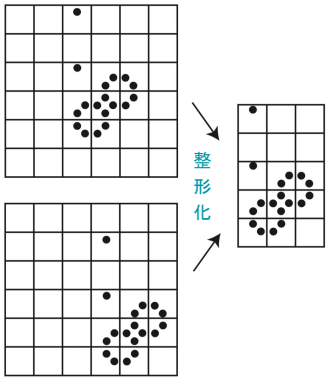
遷移規則の適用

図 6　2D-RPCA P_2 の大域関数

実行時間に比例した
メモリ使用量 ⇒ 非効率

[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] 渡邉恭平：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).

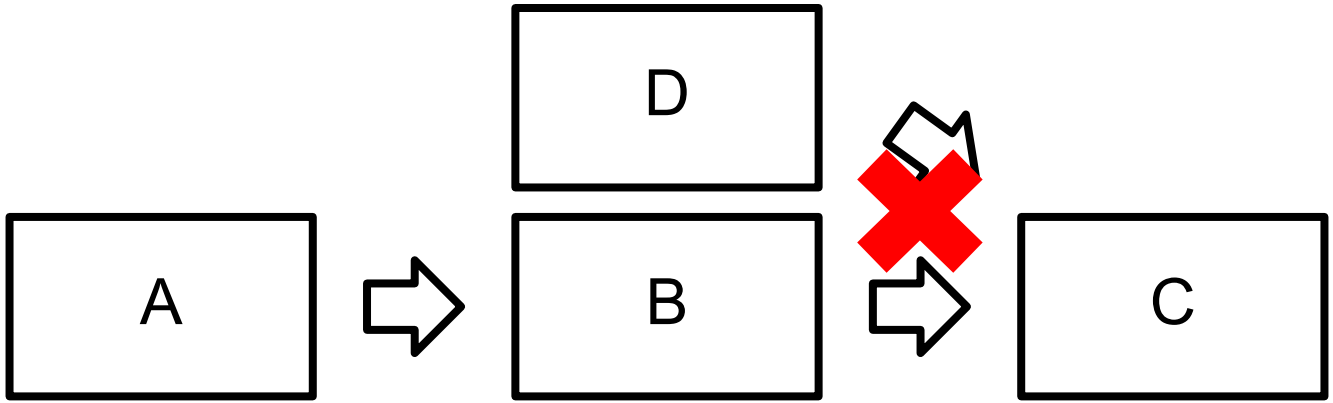
[8] Morita, K., Harao, M.: Computation Universality of one-dimensional reversible (injective) celluar automata, *Trans. IEICE Japan*, E72:758–762 (1989).

今後の課題

- 提案した 有限状相の効率的な表現 の実装
  - 実行時間に比例したメモリ使用量 ⇒ 非効率
  - 入力消去可逆シミュレーション[Bennett89]を応用(6パス)
- 1パスでクリーン可逆シミュレーションの実現

準備

可逆計算:
全ての時間の状態から直前の状態が
たかだか一つに特定できるような計算



可逆セルオートマトン

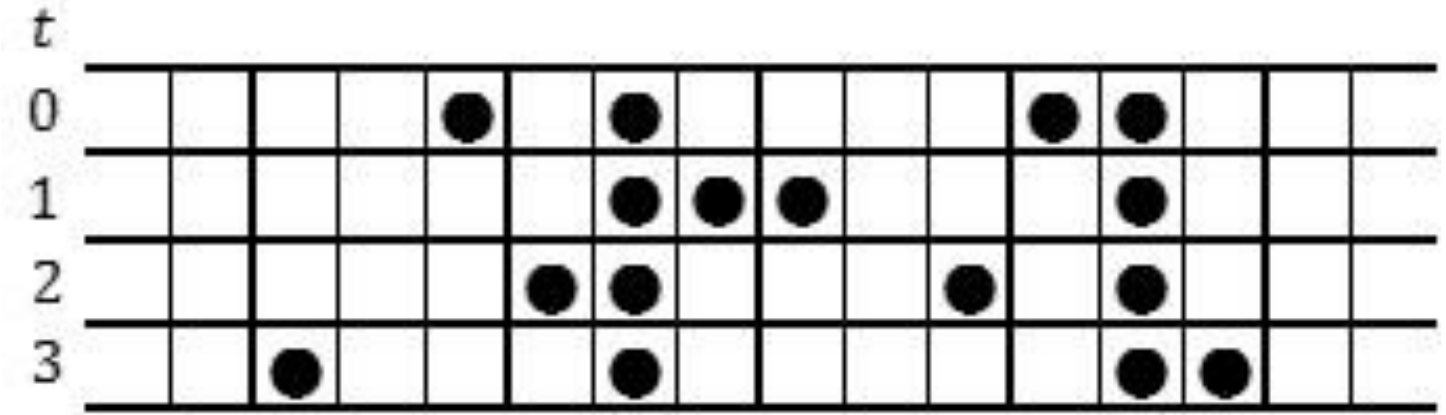
- セルオートマトン(CA)
  - 有限オートマトンを規則的に配置・接続した計算システム
- 可逆セルオートマトン(RCA)
  - 状相の遷移を行う大域関数に単射性の制約を与えたCA

可逆分割セルオートマトン

- 分割されたセルを持つCA
- セルの遷移を行う局所関数が単射なので大域関数が単射
- プログラムでは有限状相のみ実現

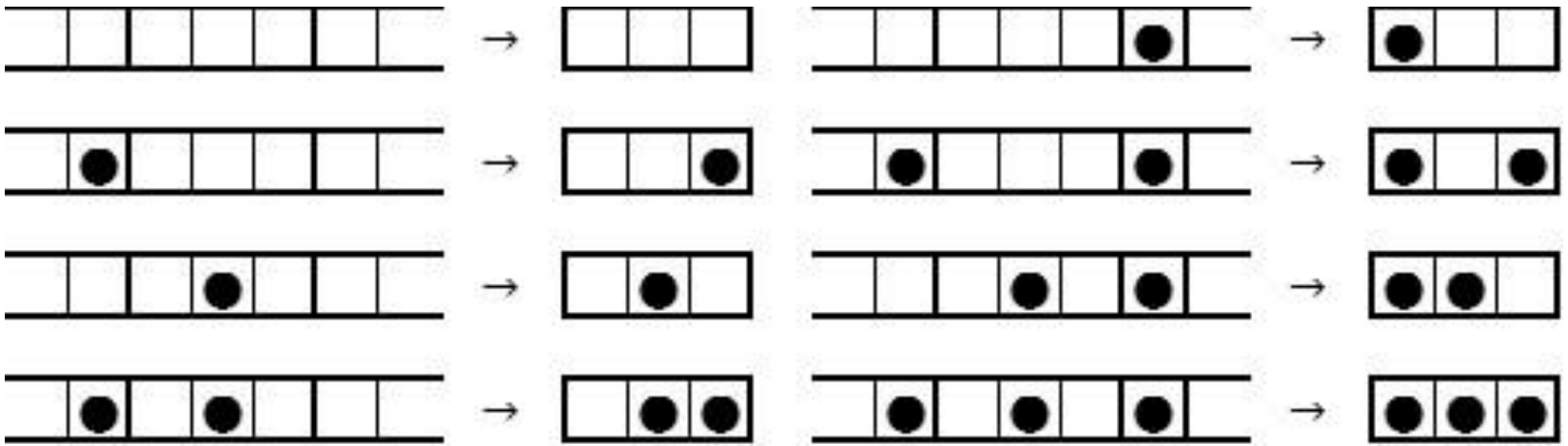
1次元可逆分割セルオートマトンの実現[木村,矢澤18]

- 無限に広がる状相を表現
- 状相の表現の最適化



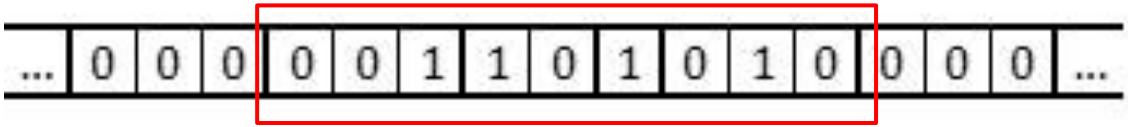
1次元可逆分割セルオートマトンの実現

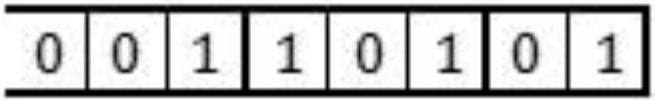
局所関数



1次元可逆分割セルオートマトンの実現

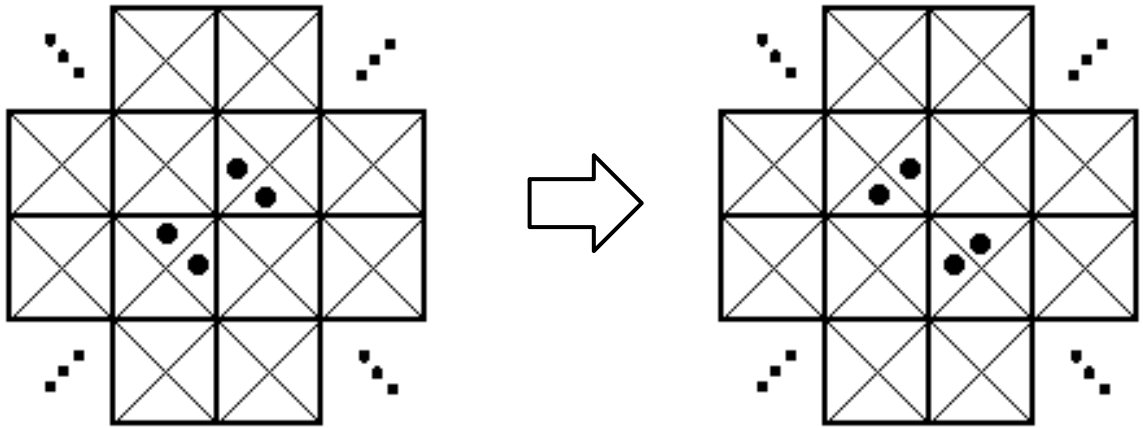
- 分割位置を保ち，静止状態の数が少ない表現





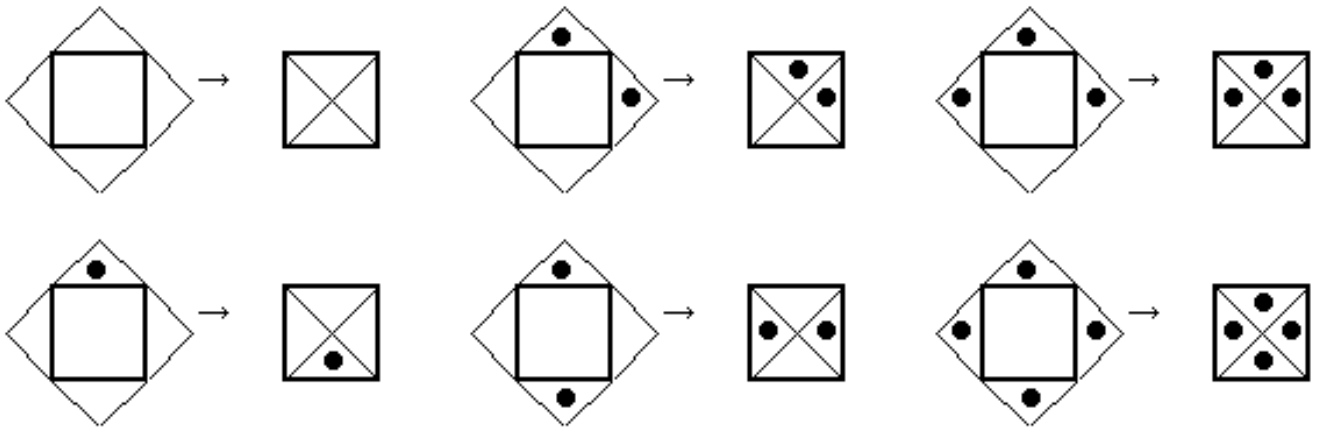
2次元可逆分割セルオートマトン[森田12]の実現

可逆動的表を用いて実現



2次元可逆分割セルオートマトン

遷移規則



配列を拡張するプログラム

配列の解放・割り付けが独立

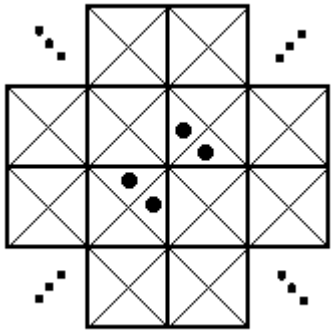
```
procedure table_expand(int t[][])
  local int t2[size(t)+2][size(t)+2] = {{0}}
  call move(t, t2)    // zero clear t
  delocal int t[size(t2)-2][size(t2)-2] = {{0}}
  local int t[size(t2)][size(t2)] = {{0}}
  call swap(t2, t)    // zero clear t2
  delocal int t2[size(t)][size(t)] = {{0}}
```

大域関数のプログラム

```
procedure global_map(int sr[][], int rule[][])
    local int sl[size(sr) + 2][size(sr) + 2] = {{0}},
        int u = 0, int l = 0, int r = 0, int d = 0
    call table_expand(sr)
    iterate int y = 0 to size(sl) - 1
        iterate int x = 0 to size(sl) - 1
            call get_ulrd(u,l,r,d,sr,x,y)
            local int t = 0
                call local_map(t, u, l, r, d, rule)
                t <=> sl[x][y]
            delocal int t = 0
        end
    end
    call move2(sl, sr)
    delocal int sl[size(sr)][size(sr)] = {{0}}, int u
        = 0, int l = 0, int r = 0, int d = 0
```

現在のプログラムにおける状相の表現

- 同一の状相の異なる表現を別のものとして扱う
- 遷移の際に表現が必ず1だけ拡大

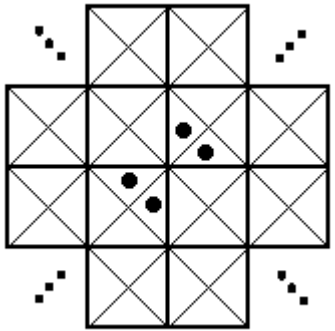


```
{{0, 12},
 3, 0}}
```

```
{{0, 0, 0, 0},
 {0, 0, 12, 0},
 {0, 3, 0, 0},
 {0, 0, 0, 0}}
```

効率的な状相の表現

- 行・列それぞれの先頭と末尾に静止状態以外を格納
- 整形な配列



```
{{0, 12},
 3, 0}}
```

```
{{0, 0, 0, 0},
 {0, 0, 12, 0},
 {0, 3, 0, 0},
 {0, 0, 0, 0}}
```

参考文献

[1] Moriyama, K.: Reversible Cellular Automata from a Programming Language Perspective, Master's thesis, DIKU, University of Copenhagen (2010).

[2] 渡邉恭平：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).

参考文献

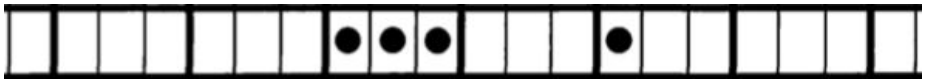
[4]Lutz, C.: Janus: A time-reversible language, Letter to R. Landauer (1986).

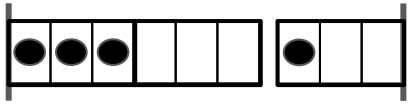
[5]Yokoyama, T., Axelsen, H.B. and Gl¨uck, R.: Principles of a Reversible Programming Language, Proc.CF, ACM Press, pp.43–54 (2008).

[6] 森田憲一:可逆計算，ナチュラルコンピューティング・シリーズVol. 5, p. 87-118，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).

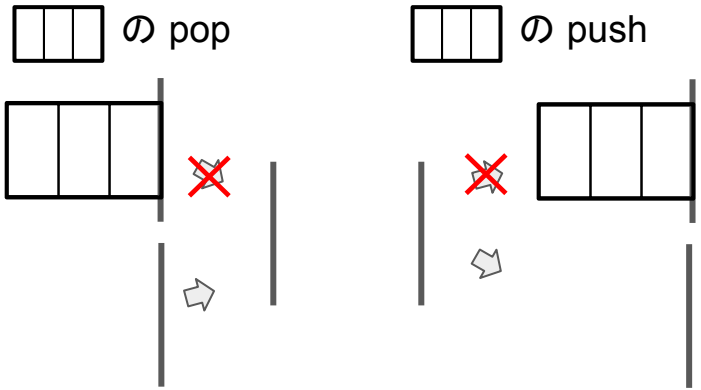
1D-RPCA P_1の クリーン可逆シミュレーション[木村,矢澤18]

状相 …  …

表現 

- 任意の1次元有限状相を スタック対で表現可
- メモリ使用量が削減

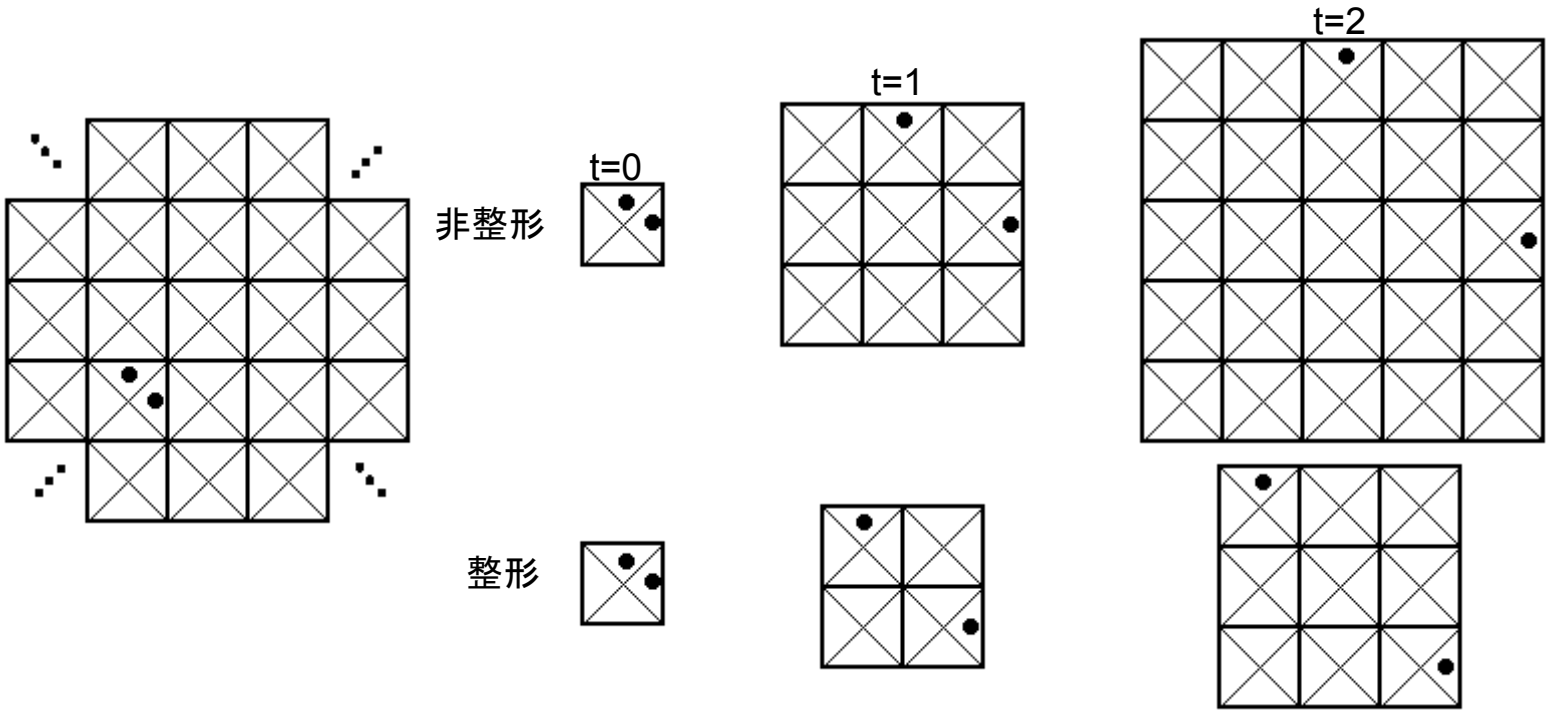
スタック操作



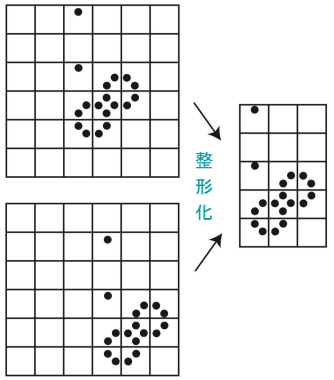
底の静止状態を禁止
⇒ スタック操作の単射化
[YokoyamaAxelsenGlück2016]

2次元に拡張したい

P_2 の状相と配列表現



[木村,矢澤18]の拡張 の実装に向けて



- 状相表現の整形化は非単射
  - ゴミ出力を伴えば実装可能
- 入力消去可逆計算[Bennett89]を応用
  - 単射な計算ならばゴミ出力と入力を消去可能
- 2D-RPCAの状相の遷移は単射

状相の表現の最適化の考察

- 整形な遷移前後の状相をそれぞれ α_1, α_2
- 非整形な遷移前後の状相をそれぞれ β_1, β_2
- プログラムは α_1 から β_2 を出力
- β_2からゴミ出力を伴い β_1を出力
- α_1から β_1の遷移は単射
- 入力消去可逆計算でゴミ出力と入力を消去可能