修　士　論　文

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

指導教員　横山 哲郎

2020 年 7 月

南山大学 大学院　理工学研究科 ソフトウェア工学専攻

Clean Reversible Simulation of Two-Dimensional Reversible Partitioned Cell Automata on a Reversible Programming Language

M2018SE013　YAZAWA Takumi

Supervisor　YOKOYAMA Tetsuo

July 2020

Graduate Program of Software Engineering
Graduate School of Science and Engineering
Nanzan University

要約

セルオートマトン (CA) は化学，社会システムなど幅広いシミュレーションに利用されている．CA の状相の遷移に対し，単射性の制約を加えたものが可逆セルオートマトン (RCA) である．RCA の一つである，可逆分割 CA(RPCA) はそれぞれのセルが分割されている CA である．RPCA の可逆プログラミング言語 Janus 上でのシミュレーションは，複数実現されており，周期的境界条件のあるもの [2]，周期的境界条件がなく，遷移の度に状相を表現するスタックがちょうど 2 セルぶんずつ大きくなるもの [7]，周期的境界条件がなく，状相の表現が常に効率的なものになるもの [8] が実現されている．しかし，我々の知る限りでは，2 次元 RPCA についてはまだ実現されていない．

2 次元 RCA には弾性衝突のシミュレーション [6] など，物理シミュレーションの応用例がある．可逆性は物理法則と深い関係があり，2 次元可逆セルオートマトンを実現することは様々な物理シミュレーションの基礎となる．2 次元可逆セルオートマトンの実現にあたって，無限長の状相を有限のメモリで扱う必要がある．

本研究では，1 次元 RPCA を実現した際の手法を 2 次元に拡張することで 2 次元 RPCA の実現を図る．また，2 次元の状相を表現する際に 2 次元配列を用いることで直接的にセルの遷移を行う．その際に，2 次元配列を実行中に拡大する必要があるため，配列の割り付け・解放を独立して行えるように Janus を拡張することによりこの問題の解決を図った．このようにして実現された 2 次元可逆セルオートマトンには，遷移を行うたびに状相の大きさがちょうど 1 ずつ拡大するという問題がある．我々はその問題に対し，入力消去可逆計算 [9, 10] を応用するというアプローチで解決を試みた．

Abstract

A Cellular automata (CA) is used for science and social system simmulation. A reversible cellular automaton (RCA) is a CA with an injective constraint. A reversible partitioned cellular automata is RCA which each cell is divided. The simmulation of RPCA on Janus has been implemented multiple times. One is periodic[2], the other one is not periodic and expand[7], and the other is not periodic and not expand[8]. But 2 dimmentional RPCA is not implemented.

2 dimmentional RCA has many applications such as simulations of elastic collisions[6]. Reversibility is closely related to the physics, and realization of a two-dimensional reversible cellular automaton is the basis of various physical simulations. We need to implement infinitely configulation on finite memory.

In this paper, We aim to realize 2 dimmentional RPCA by extending the method used to realize 1 dimmentional RPCA to 2 dimensions. We use 2 dimmentional array to express 2 dimmentional configulation. We expand Janus because we need to expand array. The 2 dimmentional RPCA realized in this way has the problem that the size of the array increases by exactly 1 each time a transition. We tried to solve that problem by use input-erasing reversible computation.

目次

第 1 章

はじめに

1.1 背景・目的

セルオートマトン（以下，CA）は結晶の成長などの化学システムや交通モデルなどの社会システムといった様々な研究分野においてシミュレーションに用いられている計算モデルである．CA の状相（セル空間全体の状態）の遷移を行う大域関数に単射性の制約を与え，直前の状相が一意に定まるものが可逆 CA（以下，RCA）である．可逆計算は計算に関わる消費エネルギーに深い関係があり，可逆 Turing 機械（以下，RTM）などの計算モデルや観測以外の演算が可逆性制約を満たす必要がある量子計算など様々な応用がある．非可逆の場合と同様に，可逆な計算モデルの表現能力の解析のために様々な可逆シミュレーションが実現されている．たとえば，RTM の可逆プログラミング言語による可逆シミュレーションが知られている．

本研究では，2 次元可逆分割セルオートマトン（以下，2D-RPCA）を状相のうち非静止状態がある矩形範囲のみのセルの情報を直接的に 2 次元配列でもつことで可逆シミュレーションする．

1.2 研究課題

2D-RPCA の大域関数は単射であるにもかかわらず，状相の整形な表現のみを用いた場合，状相の情報をもつ 2 次元配列の更新は必ずしも単射にはならない問題がある．状相を表現する際には，無限長の 2 次元状相を有限なメモリで表現し，かつ効率的なメモリ使用量で実現を行う必要がある．実装上の問題により，プログラムの実行中に状相を表現する際に用いた配列の拡大・縮小を行う必要がある．

1.3 アプローチ

この可逆シミュレーションの実現には 2 次元配列を実行時に可逆的に割付け/解放できるよう Janus の拡張を行う．これにより，実行中に配列の拡大・縮小が可能になる．我々は論文 [8] を応用し，状相をメモリ使用量を最適にした可逆動的表で保持して無限長の状相を有限なメモリで表現する．また，入力消去可逆シミュレーション [9, 10] を応用することで効率的なメモリ使用量での実現を図る．

1.4 期待される効果

我々が知る限りでは 2D-RPCA の可逆シミュレーションは可逆プログラミング言語で効率的に実現されていない．2D-RPCA の可逆シミュレーションの Janus での実現は，Janus の r-Turing 完全性の別証明である．また，可逆性は物理制約に深い関わりがあり，可逆シミュレーションを実現することは，様々な物理学のシミュレーションを実現する際の基礎となる．

第 2 章

関連研究

2.1　セルオートマトン

CA は最も古い計算モデルの一つであり，有限オートマトンであるセルを規則的に配置・接続した離散計算モデルである．結晶の成長などの化学システムや交通モデルなどの社会システムなど，幅広い分野で利用されている．

2.2　可逆セルオートマトン

セルオートマトンに対し，単射性の制限を加えたものが可逆セルオートマトンである．RCA の大域関数の単射性と全射性は等価である．任意の 1 次元 CA が可逆であるかどうか決定するアルゴリズムが存在することが知られている [4]．d 次元 CA は $d+1$ 次元 RCA に埋め込みを行うことでシミュレートでき，RCA は計算万能性を持つことが知られている [5]．また，1989 年には 1 次元 RCA も計算万能性を持つことが文献 [12] によって示されている．2 次元 RCA の応用例として，弾性衝突の可逆シミュレーション [6] がある．逆計算を実装することにより，GPU の計算効率を上げることが可能である．

2.3　可逆分割セルオートマトン

可逆分割 CA とは，RCA を構成する手法の一つとして知られている，それぞれのセルを分割したセルオートマトンである．特に各セルが複数部分からなる k 次元分割 CA（以下，kD-PCA）が知られている．以降では，$\overline{t}$ が文脈によって組 $(t_1, \ldots, t_m)$ 又は積 $t_1 \times \cdots \times t_m$ を表すこととする．kD-PCA は $(\mathbb{Z}^k, \overline{Q}, \overline{n}, f, \overline{\#})$ と定められる．$\mathbb{Z}^k$ はセルが配置される k 次元ユークリッド空間の整数座標をもつ点の集合である．各セルは m 個の部分からなる．$Q_i (i = 1, \ldots, m)$ は，各セルの第 i 部分のとり得る内部状態の非空有限集合である．m 個組 $\overline{n}$ をセルの近傍とよぶ．関数 $f : \overline{Q} \to \overline{Q}$ は各セルの状態遷移を表す局所関数である．$\overline{\#} \in \overline{Q}$ は，$f(\overline{\#}) = \overline{\#}$ を満たす静止状態である．写像 $\alpha : \mathbb{Z}^k \to \overline{Q}$ は状相である．状相全ての集合を $Conf(\overline{Q})$ で表す．集合 $\{x \mid x \in \mathbb{Z}^k \wedge \alpha(x) \neq \overline{\#}\}$ が有限であるような状相 α を有限状相という．関数 pr_i $(i = 1, \ldots, m)$ を，任意の $\overline{q} \in \overline{Q}$ に対して，$pr_i(\overline{q}) = q_i$ となるような射影関数とする．局所関数 f から誘導される大域関数 $F : Conf(\overline{Q}) \to Conf(\overline{Q})$ を $F(\alpha)(x) = f(pr_1(\alpha(x + n_1)), \ldots, pr_m(\alpha(x + n_m)))$ とする．

Janus 上での 1D-RPCA のクリーン可逆シミュレーションの実現については，複数の方法が提案されている．文献 [2] において，周期的境界条件をもつ 1D-RPCA が実現されている．文献 [7] は，非周期境界条件をもつ 1 次元 RCA のクリーン可逆シミュレーションをスタック付きの Janus で実現した．この実現方法では状相の遷移ごとにメモリ使用量が増加する問題があった．文献 [8] の 1D-RPCA では，状相の整形な表現のみを用いることによりメモリ使用量を最適化した．

```
procedure CA(int t_end, stack conf, int rule[][])
    iterate int t = 1 to t_end
        call gmap(conf, rule)
    end
```

図 2.1　1D-RPCA P_1 のクリーン可逆シミュレーション

2.4　可逆プログラミング言語 Janus

Janus[1] とは可逆プログラミング言語の一種である．Janus は動的なメモリ確保ができず，r-Turing 完全ではない．文献 [3] においてスタック付きの Janus を用いて RTM のクリーン可逆シミュレーションを行うことにより，可逆スタック付きの Janus が r-Turing 完全であることが示されている．

Janus は C に似た構文をもつ手続き型の可逆プログラミング言語であり，単射な意味をもつ文のみ記述できるという意味で可逆である．Janus では単射でない単純な代入はできず，代わりに加算，減算，排他的論理和の 3 種類の複合代入演算子で代入を行う．局所変数と 2 次元配列の割付けと値の初期化は `local` で行う．`local` で割付けをした局所変数と 2 次元配列は `delocal` で保持する値を指定して解放できる．非可逆言語の if 文では，その実行後に then 節と else 節のどちらに分岐していたか判別できない．この制御の更新は単射ではなく可逆性が失われる．Janus では if 文の末尾にアサーションをもち，どちらの節に分岐が行われたかが if 文の実行直後に一意に定まるようにする．このアサーションは，then 節に分岐した場合は真を表す値を，else 節に分岐した場合は偽を表す値をもたねばならない．非可逆言語の繰返しに入る制御の合流地点においても，繰返しが開始されたのか，1 回以上繰返しが行われているのかが特定できないので可逆性が失われる．Janus では繰返し文の先頭にアサーションをもち，繰返しの内部と外部のどちらから制御がきたかを一意に定まるようにする．このアサーションは，繰返しの開始時に真を表す値を，1 回以上繰返した後である場合は偽を表す値をもたねばならない．Janus では call 文により，プロシージャを呼び出すことができる．uncall 文はプロシージャの本体を逆方向に実行する逆呼出しをする．スタックへのプッシュは，`push(`x`,`g`)` を用いて行う．スタック g に変数 x の値を格納し，その後，x の値を 0 クリアする．ポップは `pop(`x`,`g`)` を用いて行い，スタック g の先頭の値を取り出し x に格納する．このとき，変数 x の値は 0 でなければならない．`push` と `pop` は互いに逆である．

2.5　1 次元可逆分割セルオートマトン

文献 [8] で実現された 1D-RPCA は，有限個の非静止状態が含まれる状相の範囲のみを直接的にもつ整形スタックを用いることで大域関数の単射性を保ちつつメモリ使用量を最適化した．1D-RPCA は RTM を可逆的にシミュレートできるので，このプログラムは Janus の r-Turing 完全性の証明になっている．

各セルを 3 分割した 1D-RPCA $P_1 = (\mathbb{Z}, (L, C, R), f, (\#_1, \#_2, \#_3))$ を定める．ここで，$\#_1 = \#_2 = \#_3 = 0$, $L = C = R = \{0, 1\}$ であり，f は表 2.1 で定まる局所関数であり，P_1 は有限状相のみをもつものとする．

図 2.1 に時間発展に伴う P の状相の変化を計算する `CA(t_end,conf,rule)` を示す．

ただし，`t_end` は終了ステップ数を表し，スタック `conf` は状相を表すものであり，スタックの底には必ず非静止状態が格納され，かつセルの分割位置が変わらないような最小数の静止状態が用いられている．`gmap` では，繰返しごとにスタックで表現されたセル数回，局所関数 f を計算する `lmap` が呼び出される．各繰返しでは，スタックから取り出した `l`, `c`, `r` の値を用いて局所関数を実行し，後にスタック `sl` にセルの 3 つの部分を順に格納する．繰返し終了後は，可逆プロシージャ `rewind` によって `sl` に格納されたセルを 3 つずつ逆順に `sr` に移す．図 2.3 に `lmap` を示す．`lmap` では，まず `l`, `c`, `r` の値を利用してその状態の際に適用する遷移

```
procedure gmap(stack sr, int rule[][])
    local stack sl=nil,int l=0,int c=0,int r=0,int next_r=0
    call mypop(l, sr)
    from empty(sl) && c=0 && r=0 && next_r=0 loop
        call lmap(l, c, r, rule)
        call mypush(l,sl) call mypush(c,sl) call mypush(r,sl)
        call mypop(c,sr)  call mypop(r,sr)  call mypop(l,sr)
        r <=> next_r
    until empty(sr) && l=0 && c=0 && r=0 && next_r=0
    call rewind(sl, sr)
    delocal stack sl=nil,int l=0,int c=0,int r=0,int next_r=0
```

図 2.2　1D-RPCA P_1 の大域関数

```
procedure lmap(int l, int c, int r, int rule[][])
    local int no = (l<<2) + (c<<1) + (r<<0)
    l ^= ((no&0x4)>>2) ^ rule[no][0]
    c ^= ((no&0x2)>>1) ^ rule[no][1]
    r ^= ((no&0x1)>>0) ^ rule[no][2]
    from l = rule[no][0] &&
         c = rule[no][1] &&
         r = rule[no][2] loop
         no -= 1 // zero clear no
    until no = 0
    delocal int no = 0
```

図 2.3　1D-RPCA P_1 の局所関数

規則の番号を計算して no に格納する．3–5 行目では，各変数を no を用いて 0 クリアして遷移規則適用後の状態に更新する．繰返しでは，更新後の状態の情報を用いて no を 0 クリアする．無限に広がる 1 次元の状相を表現する際，RTM の無限長のテープを有限メモリで表す方法 [3] を応用することができる．文献 [8] では文献 [3] と同様にプッシュとポップの拡張を行った．空のスタックに静止状態をプッシュする際には何も入れず，mypop は mypush の逆呼出しを用いて定義する．可逆プロシージャ CA(t,conf,rule) の空間計算量を考える．conf が保持するのは，非静止状態のセルを両端とする必ずしも静止状態ではないセルが存在する範囲である．これにより，無限個の静止状態のセルを有限メモリで表現できるので，提案プロシージャのメモリ使用量は小さい．

表 2.1　1D-RPCA P_1 の局所関数 $f(l, c, r) = (l', c', r')$

$l\,c\,r$	$l'c'r'$	$l\,c\,r$	$l'c'r'$
000	000	100	100
001	001	101	101
010	010	110	110
011	011	111	111

第 3 章

Janus の拡張

本研究では Janus に 2 次元配列と配列の割付け/解放が独立して行える拡張を行ったものを用いる．構文領域を次のように拡張する：

$$
\begin{array}{lr}
d ::= \cdots \mid x\texttt{[}c\texttt{]}\texttt{[}c\texttt{]} & \text{スカラと配列} \\
e ::= \cdots \mid x\texttt{[}e\texttt{]}\texttt{[}e\texttt{]} & \text{式} \\
s ::= \cdots \mid x\texttt{[}e\texttt{]}\texttt{[}e\texttt{]} \oplus\texttt{=}\ e \mid \texttt{local}\ t\ x\ \texttt{=}\ e \mid \texttt{delocal}\ t\ x\ \texttt{=}\ e & \text{文}
\end{array}
$$

状態を σ，値を v と書くことにする．式と文の判定をそれぞれ $\sigma \vdash_{expr} e \Rightarrow v$ と $\sigma \vdash_{stmt} s \Rightarrow \sigma$ とする．割付けと解放の自然意味論を定める：

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

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

`delocal` 文における 2 次元配列の解放は，正しい配列サイズが指定され，全要素が 0 をもつ場合にのみ成功する．本研究で用いる Janus においては `local` と `delocal` は入れ子の形で用いる必要は無い．このような拡張を行った Janus も可逆である．以降では拡張された Janus を単に Janus と呼ぶ．

第 4 章

2 次元可逆分割セルオートマトン

本章では，2D-RPCA とその可逆シミュレーションの方法を述べる．

4.1　2 次元可逆分割セルオートマトン

2D-RPCA $P_2 = (\mathbb{Z}^2, \overline{Q}, \overline{N}, f, \overline{\#})$ を定める．ここで，近傍は $\overline{N} = ((0,-1),(-1,0),(0,1),(1,0))$，$Q_i = \{0,1\}$，$\#_i = 0$ $(i = 1,2,3,4)$, $f\,(u,r,d,l) = (u',r',d',l')$ は表 4.1 に示す遷移規則の集合である．f の右辺が全て異なるので単射であり，P_2 は可逆である．

4.2　2 次元配列による有限状相の表現

P_2 の状相を 2 次元可逆動的配列（以下単に 2 次元配列）c で表す．分割されたセルの状態 Q_i を 0/1 で表し，$Q_i(i = 1,2,3,4)$ をそれぞれ $1,2,4,8$ との積の和で効率的に状態を表す．2 次元配列の端以外にあるセル c[x][y] の近傍は c[$x \pm 1$][$y \pm 1$] というように直接的に表せる（復号任意）．無限のセルをもつ状相を c で直接的に表すことはできない．我々は 1D-RPCA の有限状相をスタックで表現した方法 [8] を応用し，2D-RPCA の有限状相のセルが存在しうる矩形領域のみを c で直接的にもつ．

4.3　大域関数の実現

P_2 の大域関数 F は定義より単射である．しかし，c を更新する計算は単射では無い．なぜなら，上記の矩形領域の定め方により境界に静止状態をもつ無数の 2 次元配列は同一の状相を表し，c を更新する計算は多対多の関係になるからである．

単純な解決法として c を更新するときに必ず周囲に 1 要素ずつ静止状態を追加することで計算を単射にすることが考えられる．この方法だと実装が比較的簡単になる．

別の解決法も考えられる．境界に非静止状態をもつ 2 次元配列を整形とよぶ．F を表す計算を 2 次元配列の整形性を保つように入力消去可逆計算 [9, 10] で実現できる．この実現には非可逆な計算とその逆計算の可逆

表 4.1　2D-RPCA P_2 の局所関数 $f(u,r,d,l) = (u',r',d',l')$

$urdl$	$u'r'd'l'$	$urdl$	$u'r'd'l'$	$urdl$	$u'r'd'l'$	$urdl$	$u'r'd'l'$
0000	0000	0100	0100	1000	1000	1100	0011
0001	0001	0101	1010	1001	0110	1101	0111
0010	0010	0110	1001	1010	0101	1110	1011
0011	1100	0111	1101	1011	1110	1111	1111

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

図 4.1　2D-RPCA P_2 のクリーン可逆シミュレーション

```
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
```

図 4.2　2D-RPCA P_2 の大域関数

的な実現が必要である．その両者を整形の c からその矩形領域の周囲にたかだか 1 要素ずつ追加した非整形の c' へそれぞれ F と F^{-1} を表す計算で非可逆的に実現できる．しかし，この方法では実装が複雑になるという欠点がある．

4.4　可逆プログラミング言語による実現

P_2 の可逆シミュレーションは図 4.1 のプロシージャ CA により行う．CA は初期状相 conf に大域関数 gmap を t_end 回適用して更新する．

図 4.2 に大域関数を計算する gmap を示す．まず遷移後の状相の表現を一時保存用の 2 次元配列 c2 を割り付ける．次に図 4.3 の table_expand で遷移前の状相の表現をもつ c1 の上下左右の境界を 1 セルずつ拡張する．この拡張には local と delocal を独立に使用する必要がある．したがって既存の Janus では実現できない．続く繰返しでは各セルの近傍をプロシージャ get_ulrd で一時変数 u, l, r, d に移し，それらを引数としたプロシージャ lmap で局所関数をクリーン可逆シミュレーションした結果をそれぞれ c2 に格納する．局所関数は図 2.3 と同様である．swap は c2 と全要素が 0 クリアされた c1 の全要素を入れ替える．図 4.4 に状相を表す 2 次元配列の遷移の例を示す．遷移ごとに 2 次元配列の行と列が 2 ずつ拡大する．要素数 n^2 の 2 次元配列からの遷移において，メモリ使用量は $4n + 4$ 増え，計算ステップ数は $\mathrm{O}(n^2)$ である．

```
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}}
```

図 4.3　配列の拡大

```
{{0,12},   {{0, 0, 0, 0},       {{0, 0, 0, 0, 0, 0},
  3, 0}}    {0, 0,12, 0},        {0, 0, 0, 0, 0, 0},
            {0, 3, 0, 0},        {0, 0, 6, 0, 0, 0},
            {0, 0, 0, 0}}        {0, 0, 0, 9, 0, 0},
                                 {0, 0, 0, 0, 0, 0},
                                 {0, 0, 0, 0, 0, 0}}
```

図 4.4　左から初期状態，遷移前，遷移後の 2 次元配列

4.5　効率的な状相の表現方法

有限状相を表す 2 次元配列は，配列の先頭と末尾の行と列にそれぞれ非静止状態が含まれる場合，整形という．例えば，図 4.4 の初期状態の配列と遷移前の配列は同じ状相を表している．したがって，整形な配列に写す大域関数を適用すると遷移後の状相が同じになり，単射性が失われる．4.4 節では非整形の配列に写す大域関数を用いることで単射性を実現していた．本節では図 4.4 の初期状態の配列のように，整形な配列のみを用いて単射性を実現する．

4.6　入力消去可逆計算を用いた実現

効率的な遷移前後の状相をそれぞれ α_1, β_1, 非効率な遷移前後の状相をそれぞれ α_2, β_2 として，これらの状相の変換を可逆的に行うことができるかについて考察する．今回実装したいプログラムは α_1 から β_1 への遷移であり，これは，2D-RPCA の定義から単射であり，可逆である．よって，入力消去可逆計算 [9, 10] を応用し，ゴミ出力を消去することが可能である．なお，ここでは状相の表現に必要な出力以外の出力をゴミ出力とみなす．4.4 節で示したプログラムは，入力を α_1 に限定することで，α_1 から β_2 への遷移を行うゴミ出力を伴わないプログラムとみなすことができる．このように遷移された β_2 からゴミ出力を伴いながら β_1 に変換を行っても，入力消去可逆計算を利用してゴミ出力が消去可能，つまり，α_1 から β_1 へのゴミ出力を伴わない遷移を行うプログラムが実現可能である．

第 5 章

おわりに

本研究では，可逆動的表を用いた 2D-RPCA の Janus 上での可逆シミュレーションと，状相の表現の最適化について検討を行った．我々は，非静止状態のセルが存在する矩形領域を直接的に表すメモリのみで 2D-RPCA の可逆シミュレーションの実現方法を示した．これは Janus の r-Turing 完全性を証明したことになる．しかし，実装上の問題が解決できず，2D-RPCA の可逆シミュレーションは遷移ごとにメモリ使用量が増えるものとなった．

謝辞

本研究を進めるにあたり，指導教員であり横山哲郎教授には修士論文の研究計画を行う上で助言を頂きました．また，予稿，本稿，要旨に関しても査読と多くの助言を頂きました．横山教授の多くのご指導に深く感謝いたします．また，青山幹雄教授と吉田敦教授には，中間審査において研究に関するコメントを頂いたことと，本論文をご精読いただいたことに心から感謝いたします．最後に，共に研究を行い，予稿，本稿，要旨の査読，またそれに限らず研究の前段階から多くの助言をいただいた横山研究室の皆さまに感謝いたします．

参考文献

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

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

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

[4] Sutner, K.: De Bruijn graphs and linear cellular automata. Complex Syst. 5(1), 19–30 (1991).

[5] Toffoli, T.: Computation and construction universality of reversible cellular automata. J. Comput. Syst. Sci. 15(2), 213–231 (1977).

[6] Kalyan S. Perumalla and Vladimir A. Protopopescu.: Reversible simulations of elastic collisions. ACM Trans. Model. Comput. Simul. Vol.23, No.2, Article 12 (2013).

[7] 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).

[8] 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).

[9] Bennett, C.H.: Logical Reversibility of Computation, *IBM J. Res. Dev.*, Vol.17, No.6, pp.525–532 (1973).

[10] Bennett, C.H.: Time/space trade-offs for reversible computation, Vol.18, No.4, pp.766–776 (1989).

[11] Morita, K.: Computation-universality of one-dimensional one-way reversible cellular automata, *IPL*, Vol.42, No.6, pp.325–329 (1992).

[12] Morita, K. and Harao, M.: Computation Universality of One-Dimensional Reversible (Injective) Cellular Automata, *Trans. IEICE Japan*, Vol.E72, No.6, pp.758–762 (1989).