可逆計算の調査

田島 嘉人

1　可逆計算について

1.1　可逆計算

可逆計算とは？詳しくまとめる

1.2　可逆計算の活用

この章では，可逆コンピューティングについて概説する．「可逆」とは，入力 x に対して，$f(x)$ が出力された際，$f(x)$ から x を計算できる性質のことである．つまり，すべての計算が単射であり，初期状態と最終状態を除いたすべての状態において直前と直後の状態を一意に定めることができる．この性質を持つ計算には，次のような応用方法がある．

- マルチプロセッサなどの投機的実行が効率的に行える
- ハードウェアのエラー検出
- エディタなどで必要な部分を消去してしまった場合，復元が可能である
- もっと具体的に応用例を挙げる

以上のように活用できる可逆計算であるが，より大局的な目的として，エネルギー効率の良いコンピュータの製作が挙げられる [1]．これは，物理的可逆性と呼ばれ，「可逆プロセッサ」，「可逆言語」，「可逆アルゴリズム」の 3 つの組み合わせで構築される．ランダウアーの研究から，コンピュータが計算を行う際に消失するビットが熱を発生させることが知られている (いつかここら辺を詳しくまとめる)．非可逆計算にはビットの消失が伴い，これによってエネルギーが消費される．対して，可逆計算はビットの消失が発生せず，エネルギーの効率が良い．

現代には，このエネルギー効率が求められる製品が多く存在する．例えば，スマートフォンなどのモバイル端末はその代表例である．これらのエネルギー効率を改善することは，バッテリーの軽量化や長寿命化など様々な利点をもたらす．また，可逆計算によって CPU の性能向上も期待できる．現在 CPU の性能向上を阻害する要因として，熱の発生が挙げられる．また，〜によると CPU の性能による熱の発生は 60 年後に物理的限界を迎えると考えられている．この問題を解決する為には，可逆計算を CPU アーキテクチャに取り入れる必要があり，これらの研究が望まれる．また，中長期的研究対象として，量子計算への応用が考えられる (なぜ量子計算に用いられるのか詳しく)．

【可逆アルゴリズムがなぜ重要かを追記】

2　可逆コンピューティングについて

3　予備知識

3.1　可逆計算の計算モデル

計算モデルは，ある計算システムの計算能力について，個々のハードウェアや詳細に定義された計算方式に依らない，非常に抽象的な議論を行うために用いられる．例えば，あらゆる計算の計算可能性について検証できるチューリング機械や，プログラミング言語の分野において広く用いられるラムダ計算，交通モデルや化学システムに利用される離散的計算モデルのセル・オートマトンなどが計算モデルの代表的なものである．可逆計算の代表的な計算モデルには可逆チューリング機械がある．可逆チューリング機械は，チューリング機械の様相遷移グラフ[*1]に，すべての様相においてその様相の入次数 (他の様相から来る辺の数) と出次数 (他の様相へ向かう辺の数) は最大 1 つであるという条件を付与したものである．この条件の下では，様相遷移グラフは線形になり，ある様相から以前の様相を一意に定めることができる．つまり，可逆チューリング機械の計算は可逆であると言える．図 1，2 に，一般的なチューリング機械と可逆チューリング機械の様相遷移グラフを示した．図 1 では，最終様相が J であるとすると，任意の初期様相から最終様相へ遷移する道順は A→F→H→I→J や E→G→I→J など様々に考えられる．しかし，図 2 においては，最終様相が E であるとすると，E へ遷移する過程は任意の初期様相から一意に決まる．

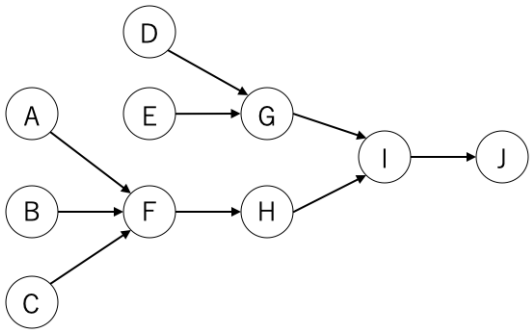

図 1　チューリング機械の様相遷移グラフ

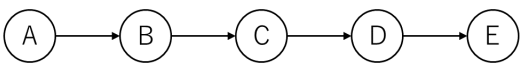

図 2　可逆チューリング機械の様相遷移グラフ

[*1] 様相については付録 A を参考にして欲しい

チューリング機械

可逆チューリング機械のモデルであるチューリング機械は，1936 年に Alan Mathieson Turing が考案した，あらゆる計算を行うことができる思考上の機械である [2]．計算可能なすべての計算はチューリング機械で計算することができ，逆に，チューリング機械で計算できる計算はすべて計算可能である (参考文献があるといいかな？)．チューリング機械の振る舞いについては，付録 A に記述した．

3.2 双方向変換

ある情報 s に対して 2 つ以上の表現 $v_1, v_2, \ldots, v_n$ がある場合，その全ての表現について一貫性を保つための方法を双方向変換[*2]という．双方向変換はデータベース，グラフ変換，ソフトウェア工学，プログラミング言語など多くの分野でその活用が研究されている [3]．例えば，ソフトウエア工学の分野では 2 つ以上のモデルの一貫性を自動的に保つようなシステムの研究が行われている [4]．

双方向変換は形式的に $get(s_1) \to v_1$ 及び $put(v_2, s_2) \to s'$ の 2 つの関数で表現される [5]．ここで s はソース，v はソースに対する表現である．つまり，関数 get はソースから対応する 1 つの表現を出力し，関数 put は 1 つの表現とソースから対応する更新されたソースを出力する．関数 get は単射である．また，関数 put は関数 get から自動的に生成することが可能である．

4 可逆プロセッサ

PISA．【コペンハーゲン大の BoB や Hall のアーキテクチャも。抽象機械は他にも考えられているとおもう。】

5 可逆言語

可逆コンピューティングには，可逆プログラムを記述するための言語が必要である．これには 2 通りの実現方法が考えられる．1 つは，既存のプログラミング言語に対して可逆コンパイラを作成することで，可逆性を持つ機械語を生成する方法である．しかし，欠点として，可逆プログラムへの変換をプログラマーがコントロールできないという問題がある．可逆プログラムには，可逆プログラムに適した計算の方法があり，それらをコントロールできない書き方では，意図せず計算効率が悪くなる可能性がある．

もう 1 つの方法は，可逆プログラミング言語を使用する方法である．可逆プログラミング言語は，その言語で記述されたプログラムが可逆性を持つことを保証する．また，可逆計算に適するように作られているので，可逆計算用のアルゴリズムを記述しやすい．可逆プログラミング言語には Janus や R がある．

また，計算を完全に可逆的にするのではなく，部分的に可逆的にすることで非可逆計算の効率化を図ろうとする試みも行われている．このような計算を実現するためのプログラミング言語に Eel がある [6]．

6 可逆アルゴリズム

この章では，可逆アルゴリズムについて概説する．
ここに書くこと．

- 可逆的な操作は，非可逆操作に比べ必ずオーバーヘッドが発生する ([1]3.4)
- その中で，可逆操作の効率を上げることは，可逆コンピューティングを利用する関心を高める？

6.1 アルゴリズムの解析手法

アルゴリズムには，その性能を客観的に測るためのいくつかの指標がある．これらの指標を用いてプログラムの性能を測ることを，アルゴリズムの解析と呼ぶ．この節では，以降の節での説明で用いるアルゴリズムの解析手法について解説する．

6.1.1 計算量の表現方法

本稿では，漸近的上界を O，漸近的下界を Ω，漸近的上界かつ漸近的下界 (タイトな限界) を Θ で表す．

6.1.2 時間計算量

時間計算量は，アルゴリズムの計算にかかる時間を表す．多くの可逆アルゴリズムの研究では，非可逆なアルゴリズムを，時間計算量が同等であり，空間計算量・ゴミ出力量を効率よく可逆的にシミュレーションすることを目標としている．また，可逆的なマシンは、不可逆的なマシンよりも時間効率が高くなることはない [1]3.3.3．しかし，空間効率を度外視すれば，時間効率が悪くなることもない．これは，Landauer により証明された [7]．ただ，この時の Landauer の方法では，全ての計算過程を保存していたため，エントロピーは非可逆計算と同様に増加していた．この履歴を削除できることに最初に気づいたのは，Lecerf であった．しかし，彼はエントロピーとの関係に気づかなかったため，最終的な貢献は Bennett が得ることとなった．

6.1.3 空間計算量

空間計算量は，実行したアルゴリズムが使用する記憶領域の量を表す．一般的には，空間計算量より時間計算量が重視される場合が多い．しかし，可逆計算においては，量子コンピュータなど，空間計算量がより重視される場合がある．

6.1.4 ゴミ出力量

ゴミ出力量は，可逆アルゴリズム特有の指標である．アルゴリズムを実行した際に出力されるデータ中で，問題の解決に必要な情報を保持していないものはゴミ情報と呼ばれる．なぜ不必要な情報が出力されるのかというと，可逆アルゴリズムでは，逆実行するために，これらが必要とな

[*2] 双方向変換 (Bidirectional Transformations) は bx と略される

るからである．ゴミ情報の内，実行途中で出るゴミ情報を中間ゴミ，実行終了後に出るゴミ情報を最終ゴミと呼ぶ．可逆システムによる可逆計算の実現を行うには，計算後に元の入出力以外がないことが望ましい．これは可逆システムにおいてデータを消去することができないためである．

6.2 非可逆アルゴリズムの可逆シミュレーション

非可逆計算で用いられているアルゴリズムは，代入や分岐など単射ではない計算を含んでいるため，可逆計算として扱うためには，可逆アルゴリズムへ変換する必要がある．この節では，非可逆アルゴリズムを可逆的にシミュレーションする方法について説明する．

6.2.1 Landauer 法

全ての計算履歴を埋め込む方法 [7]．埋め込み法とも呼ばれる．

6.2.2 Bennett 法

Landauer 法によって埋め込まれた計算履歴は計算を逆実行することによって消去できることが，Bennett によって発見された[*3]．この性質を利用した可逆シミュレーションが Bennett 法 [8] である．この方法は，非可逆な計算を Landauer 法によって実行し，その計算結果を有効なメモリに保存した後，計算結果と一緒に出力された計算履歴を，計算を逆実行することによって消去する．つまり，最終的な出力は計算結果のみとなる．しかし，Landauer 法で計算した場合に比べて実行時間が 2 倍になる．

6.2.3 Bennett 小石法

Bennett 法を，空間計算量の観点でより効率化したものが，Bennett 小石法である [9]．この方法では，計算ステップを $k(\geq 2)$ 個ごとに分割する．そして，計算過程が k 個進むごとに，Bennett 法を使用し，その時点までの最終的な計算結果と初期情報以外の計算履歴情報を消去する．この方法は，時間計算量 T 及び空間計算量 S の非可逆計算を時間計算量 $\mathrm{O}(T^{1+\epsilon})$ 及び空間計算量 $O(S \log T)$ で可逆計算可能である．ただし，ϵ は k の値が増加すると減少する．また，k の値が増加すると空間計算量の定数部分が増加する．

6.2.4 Lange-Mckenzie-Tapp 法

ここまでの方法は，計算の履歴を保存する事で非可逆計算を可逆的にシミュレーションしていた．対して，この節で紹介する LMT-Search[*4][10] では，計算の履歴を保存せずに可逆シミュレーションを行うことが可能である．

通常，非可逆アルゴリズムは決定的[*5]であり，また，計算は必ず終了する．つまり，非可逆アルゴリズムの考えられる全ての計算過程をグラフ化すると，そのグラフは計算終了時を根とする木構造になる．LMT-Search は，この木構造のグラフに対して迷路攻略手法として用いられる右手法を適用する．LMT-Search の考え方を直感的に表した図が図 3 である．図 3 は，J が計算終了時となる非可逆アルゴリズムにおいて考えられる全ての計算過程をグラフ化したものである．このグラフ中で A が計算開始時である場合を考える．この場合，この非可逆アルゴリズムは A→E→H→I→J と計算を進める．対して LMT-Search の計算過程は破線で表される，A→E→H→F→B→F→C→F→H→I→J である．このように右手法則に従えば，グラフ中のどこから計算を開始したとしても，計算終了時までの計算過程を一意に定めることができる．

この手法では，入力サイズ n に対して空間計算量 $S(n)$ の非可逆アルゴリズムを，空間計算量 $S(n)$ で可逆シミュレーションすることが可能である．しかし，時間計算量は指数関数的に増加する．

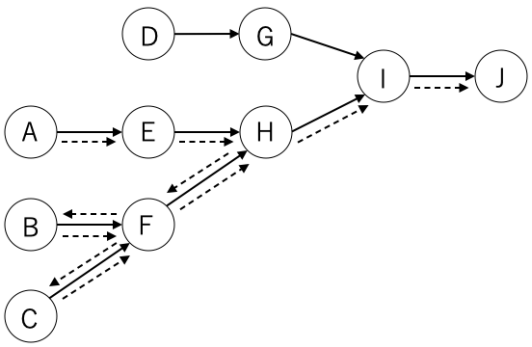

図 3 LMT Search の計算過程

6.2.5 個別解法

6.2.2，6.2.4 節において任意の非可逆アルゴリズムは線形時間または線形空間で可逆的にシミュレーションできることを述べた．同時に，これらの解法による可逆シミュレーションは，それぞれ対となる計算量 (時間計算量に対する空間計算量及び空間計算量に対する時間計算量) の漸近的オーバーヘッドがあることも述べた．現在のところ，非可逆アルゴリズムを線形時間かつ線形空間で可逆シミュレーションできる方法は知られていない．しかし，非可逆アルゴリズムに対して，そのアルゴリズムの特徴と可逆計算の特性から，効率の良い可逆アルゴリズムの開発が可能である．

このアルゴリズム個別の解法の性能を表すための概念に次のものがある．忠実と衛生の概念は [11] で導入された．

- クリーン
  非可逆アルゴリズムと変換後のアルゴリズムの引数・

[*3] 正確には，Lecerf が最初に発見したが，Lecerf はこの性質の可逆シミュレーションにおける有用性に気づかなかった

[*4] Lange，Mckenzie，Tapp の頭文字を取り，Lange-Mckenzie-Tapp 法のことを LMT Search と呼ぶ．

[*5] ある計算は必ず同じ計算結果を出力するという性質のこと．この性質を持つ計算では，何回同じ計算をしても計算結果が異なる事はない．

出力が同じものをクリーンなアルゴリズムと呼ぶ．よって，必ずゴミを出力しなければならないソートアルゴリズムは，クリーンになり得ない．

- 忠実
  漸近的なゴミ出力量が一定の値に抑えられ，変換前のアルゴリズムと同じ時間計算量であり，変換前のアルゴリズムから増えるスペース使用量は全てゴミ操作のためであるものを忠実アルゴリズムと呼ぶ．
- 衛生
  ゴミ出力が最適化されており理論的にそれ以上減らせないアルゴリズムで，忠実であるものを衛生アルゴリズムと呼ぶ．

6.2.6 非可逆アルゴリズムの可逆シミュレーション例

深さ優先探索法 (木構造) 木構造における深さ優先探索では，与えられた木は一意に定まる順序 $v_1, v_2, \ldots, v_n$ で走査され，任意の走査中の節点 v_i とその直前に走査した節点 v_{i-1} からその更に直前に走査した節点 v_{i-2} と走査中の節点の次に走査する節点 v_{i+1} はそれぞれ一意に定まる．つまり，この走査法では，走査中の節点とその直前に任意の時点において直前に走査してきた節点を探すのに，どの節点を走査してきたかの履歴を用いる必要がない．

例えば，図 4 では，木の走査は 11, 22, 33, 44, 33, 22, 55 の順で行われる．ここで，ある走査過程において走査中の節点が 22，直前に走査した節点が 33 であれば，次に走査する節点は 55 ただ一意に決まる．

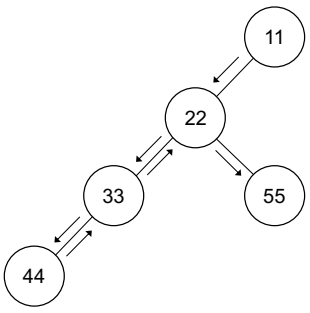

図 4 55 を深さ優先探索する過程

この走査法を命令型可逆プログラミング言語 Janus で実装したものがプログラム 1 である．簡単のため木構造は 2 分木を用いる．配列 l，r，p，v で木構造を表す．ここで，各節点 i に対して，l[i], r[i], p[i] はそれぞれその左の子，右の子，親の節点を表し，v[i] はその節点のもつ値を表す．また，節点 0 はダミーの節点であり，その左の子が根を表し，l, r, p の要素 0 は対応する節点がないことを表す．例えば，l が $[1,2,3,0,0]$，r が $[0,4,0,0,0]$，p が $[0,0,1,2,1]$，v が $[-1,22,33,44,55]$ であった場合，節点 2 に注目すると，その節点のもつ値 v[2] は 33 であり，左の子節点は 3，右の子節点はなく，親節点は 1 である．ここで v[0] の -1 は任意の探索をする値と異なるダミーの値である．整数 c は走査中の節点，pre は直前に走査した節点，k は探索する値を表す．

手続き step は走査中の節点を表す c と直前に走査した節点を表す pre の値から次に走査する節点を定め，c を次に走査する節点を表す値に，pre を走査中の節点を表す値に変更する．

手続き search は，根から走査を開始し，走査中の木の節点がキーと一致するか全節点を走査して番兵であるダミーの節点に達すると探索を終了する．二分木を表す配列 l, r, p, v，探索する値 k, 及び根である節点を表す値をもつ c が渡されて実行され，実行後に c のみを探索で見つかった節点（非ゼロ）に書き換える．ただし，探索する値 k が見つからなかった場合は，そのことを表すゼロを c に保持する．search は，非可逆な深さ優先探索法の元の入出力以外の出力をもたない．

プログラム 1 可逆深さ優先探索 (木構造)

```
 1 procedure step(int l[], int r[], int p[], int c, int pre)
 2   local int next = 0
 3   if p[c] = pre then
 4     if l[c] = 0 && r[c] = 0 then
 5       pre <=> next
 6     else if l[c] = 0 then next ^= r[c]
 7                      else next ^= l[c]
 8       fi l[c] = 0
 9       pre ^= p[c]
10     fi l[c] = 0 && r[c] = 0
11   else if r[c] != 0 then
12     if r[c] = pre then next ^= p[c]
13                    else next ^= r[c]
14     fi p[c] = next
15     pre ^= r[c]
16   else
17     next ^= p[c]
18     pre ^= l[c]
19   fi r[c] != 0
20   fi (p[next]=c && (l[c]=next || l[c]=0)) || (l[c]=0 && r[c
         ]=0)
21   pre <=> c
22   c <=> next // zero clear next
23   delocal int next = 0
24
25 procedure search(int l[],int r[],int p[],int v[],int k,int c)
26   local int pre = 0
27   from pre = 0 loop
28     call step(l, r, p, c, pre)
29   until c = 0 || v[c] = k
30   if cur != 0 then pre ^= parent[cur] // zero clear prev
31               else pre ^= 1
32   fi cur != 0
33   delocal int pre = 0
```

6.3 可逆アルゴリズムの例

- ソーティングアルゴリズム
  可逆計算では，配列を並び替える際，元の順番が失われないように，それらをゴミ情報として保存する必要がある．
  - 挿入ソート
    計算時間 $\Theta(n^2)$ のアルゴリズム．Hall[12] によって提案されたアルゴリズムでは $O(n \log n)$ のゴミ出力を用いてこれを実現できる．
  - クイックソート

クイックソートに代表される時間計算量 $\Theta(\log n!)$ のアルゴリズムは，実装が容易で，各比較ごとの結果を保存することで実現できる．
  - 基数ソート
    時間計算量 $\Theta(n)$, ゴミ出力量 $\Theta(n)$ で実現できる．
- 算術演算
  n ビットの可逆的な加算・減算には $\Theta(n)$，乗算には $\Theta(n^2)$ の時間計算量が必要である．
- 行列演算
  $n \times n$ 行列の時間計算量は $\Theta(n^3)$ である．
- 探索アルゴリズム
  深さ優先探索や幅優先探索などの，要素が特定の順番で調査され，以前調査されたノードが現在調査しているノードから特定できるようなアルゴリズムは，可逆的に実行できる．
  - 深さ優先探索
    [13] では，与えられた入力以外のゴミ出力量が 0 で，時間計算量 $\Theta(n)$，空間計算量 $\mathrm{O}(n)$ また $\Omega(\log n)$ のアルゴリズムが提案された．
- グラフアルゴリズム
  最短経路問題のアルゴリズムとして，非可逆な計算では Floyd-Warshall のアルゴリズムが知られている．このアルゴリズムは，非可逆な計算では，時間計算量 $\Theta(n^3)$，空間計算量 $\Theta(n^2)$ で動作するが，可逆計算では，空間計算量が $\Theta(n^3)$ 必要になる．[1] では，可逆計算に適した最短経路問題アルゴリズムとして，時間計算量 $\Theta(n^3 \log n)$，空間計算量 $\Theta(n^2 \log n)$ のアルゴリズムが提案された．このアルゴリズムは，時間計算量は Floyd-Warshall を上回るものの，空間計算量の効率が良い．
- 物理シミュレーション
  可逆計算は，物理シミュレーションに適している．[1] には，その例として，Schrödinger の波動方程式のシミュレーション結果が示されている．

7 文献調査

7.1 Reversible Computation and Reversible Programming Languages[14]

この論文は，可逆計算及び可逆プログラミング言語に対して，その特徴や特性を分かりやすくまとめたものである．可逆プログラミング言語の例として，Janus を取り上げており，Janus の構文や制御構造，操作的意味論について解説している．また，可逆計算や可逆プログラミングのサーベイ論文でもある．

7.2 Reversible Space Equals Deterministic Space[10]

時間計算量 $T(n)$，空間計算量 $S(n)$ で計算できる非可逆なアルゴリズムは，空間計算量を制限しなければ時間計算量 $O(T)$ で可逆シミュレーションできることが知られていた．また，時間計算量が延びることと引き換えに，空間計算量は $\Omega(S \log T)$ まで削減できると考えられていた．この論文では，この論文以前まで考えられていたこととは異なり，空間計算量 $S(n)$ で計算できる非可逆なアルゴリズムは，空間計算量 $S(n)$ でシミュレーションできることを可逆チューリング機械を用いて証明した．

7.3 Programming techniques for reversible comparison sorts[11]

長さ n の配列をソートする可逆アルゴリズムにおいては，必要な中間・最終ゴミ出力量は理論上 $\Theta(n \log n)$ にできることが知られている．しかし，埋め込み法などの一般解法を使用して非可逆なソートアルゴリズムを可逆アルゴリズムに変換する限りでは，中間・最終ゴミ出力量はこれに当てはまらない．この論文では，可逆計算独特の手法とアルゴリズムの特性から，非可逆ないくつかの比較ソートアルゴリズムに対して，時間計算量の効率が良く，中間・最終ゴミ出力量が理論上の限界となる可逆アルゴリズムを提案した．

7.4 Reversibility for Efficient Computing[1]

この論文は，可逆計算の意義・可逆論理回路・可逆プログラミング言語・可逆アルゴリズムなど可逆コンピューティング分野全般について，幅広く丁寧にまとめている．現在調査中。【9.5 Reversible algorithms を読んでまとめましょう】

【CIS 6930.3753X Spr.'02 Readings for Part V: Reversible Computing, Lecture 34 を参照する】

7.5 Toward an Energy Efficient Language and Compiler for (Partially) Reversible Algorithms[6]

従来，Janus や R などの可逆コンピューティングで扱われてきた可逆言語は，全ての操作の可逆性を保証しており，可逆性を持つ操作しか実装できなかった。同様に，そのような完全な可逆性を持つようなアルゴリズムについての研究が多く，非可逆な操作と可逆な操作両方を組み合わせたアルゴリズムの性能についての議論は行われてこなかった．この論文では，非可逆な操作と可逆な操作両方を組み合わせたハイブリットなアルゴリズムについて議論するため，非可逆な操作と可逆な操作の両方を実装できるプログラミング言語 Eel を作成した．

7.6 効率的な可逆線形探索と木構造の可逆深さ優先探索の設計と解析 [13]

この研究では，効率的な可逆線形探索と可逆深さ優先探索の設計・実装を行うため，次の 3 つについて確かめた．

- 効率的な可逆線形探索において，入力ファイルのデータ構造や出力によって空間計算量は変化する．
- ゴミ出力量が 0 という条件下で，効率的な可逆線形探

索の時間・空間計算量にトレードオフ関係が存在し，また，探索の成否によって入力ファイルの操作回数に変化が生じる．

- Bennett 法を用いたものよりも効率的な可逆深さ優先探索のアルゴリズムの存在．

7.7　可逆線形探索の解析と可逆深さ優先探索の設計 [15]

この研究では，現在提案されている可逆深さ優先探索に対してより効率化されたアルゴリズムを提案した．

7.8　二分木のランク計算のクリーン可逆シミュレーション [16]

この論文では，二分木のランク・アンランク計算を行う可逆アルゴリズムを提案している．

付録 A　チューリング機械

A.1　チューリング機械の概要

チューリング機械は，ます目に分割された無限長のテープ，有限制御部，テープ上のます目を指すヘッドの 3 点で構成される．計算は，ヘッドが指し示す，ます目の記号を読み取るところから開始される．読み取った記号と有限制御部の状態 (内部状態) を定められた規則に照らし合わせ，照らし合わせた結果に従ってテープに記された値の書き換えやヘッド位置の移動を行う．その後，再びヘッドの値を読み取り，先ほどと同様の処理を繰り返す．

以下では，チューリング機械の振る舞いについて，足し算を行うチューリング機械を用いて説明する．ただし，振る舞いを簡単にするために，足し算には + の記号が一度しか現れず，計算開始時はヘッドが必ず「1」を指しているものとする．図 5 は，このチューリング機械の動作規則である．内部状態には「初期状態」,「1 を読み取った状態」,「最終状態」があり，テープのます目に書かれている記号には，空白文字を表す「#」，数字の 1 を表す「1」，足し算の記号 + を表す「+」がある．

この動作規則によって 2 + 1 の足し算が行われる過程を図 6，図 7，図 8，図 9 に示す．図 6 では，内部状態は「初期状態」であり，ヘッドは記号「1」を指している．よって，[規則 1] に従い，内部状態を「1 を読み取った状態」へ変化させ，ヘッドが示す記号「1」を「#」に書き換え，ヘッドの位置を右に移動させる．以上の処理を行った結果が図 7 である．図 7 では，内部状態は「1 を読み取った状態」であり，ヘッドは記号「1」を指している．よって，[規則 3] に従い，ヘッドの位置を右へ移動させる．これにより，チューリング機械の様子は図 8 のようになる．図 8 では，内部状態は「1 を読み取った状態」であり，ヘッドは記号「+」を指しているので，[規則 2] に従い，内部状態を「最終状態」へ変化させ，ヘッドが示す記号「+」を「#」に書き換える．ここで，内部状態が「最終状態」になったので計算は終了し，計算結果として図 9 が出力される．

最後にチューリング機械の様相について説明する．チューリング機械の様相は，テープの記号列，内部状態，ヘッドの位置で定義される．例えば，図 6 においてチューリング機械の様相は，テープの記号は左から#，1，1，+，1，#であり，内部状態は「初期状態」で，ヘッドはテープの左から 2 番目の位置を指している．また，図 6 に対して計算を 1 ステップ進めた結果である図 7 では，テープの記号は左から#，#，1，+，1，#であり，内部状態は「1 を読み取った状態」で，ヘッドはテープの左から 3 番目の位置を指している．このように，チューリング機械の様相は計算が進むごとに変化する．通常，ある様相から変化する様相は一意に定まる．つまり，ある様相に対しての計算が異なる計算結果を返す事はない．この性質は決定性と呼ばれる．決定性を持つチューリング機械は，決定的チューリング機械である．

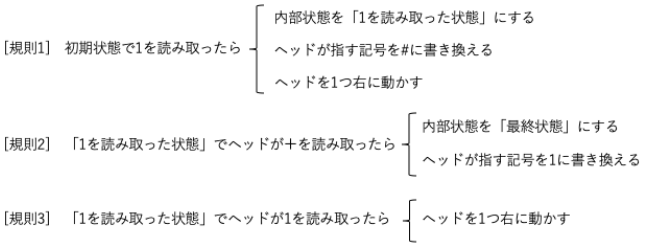

図 5　簡単な足し算を行うチューリング機械の動作規則

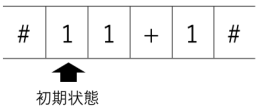

図 6　TM の振る舞い 1

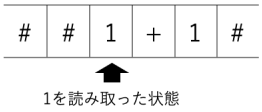

図 7　TM の振る舞い 2

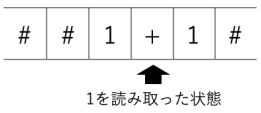

図 8　TM の振る舞い 3

図 9　TM の振る舞い 4

A.2　チューリング機械の形式的な定義

チューリング機械の形式的な表現にはいくつかの種類があるが，ここでは [17] の定義を用いる[*6]．この定義ではチューリング機械 T を $T = (Q, \Sigma, b, \delta, q_s, q_f)$ と表現する．ここで，Q は内部状態の有限集合，Σ はテープ記号の有限集合，$b(\in \Sigma)$ は空白記号，$q_s(\in Q)$ は初期状態，$q_f(\in Q)$ は最終状態である．また，δ は $Q \times [(\Sigma \times \Sigma) \cup \{\leftarrow, \downarrow, \rightarrow\}] \times Q$ の部分集合であり，集合の各要素はチューリング機械の動作規則である．ただし，$\leftarrow$ はヘッドが左へ移動することを，$\rightarrow$ は右へ移動することを表している．また，$\downarrow$ はヘッドが動かないことを表す．

[*6] その他のチューリング機械・可逆チューリング機械の形式的定義には，例えば [18] がある．

δ の要素であるチューリング機械の動作規則は，$(q_1,(s_1,s_2),q_2)$ または (q_1,d,q_2) の 3 項組で定義される．ここで，$q_1,q_2 \in Q$，$s_1,s_2 \in \Sigma$，$d \in \{\leftarrow,\downarrow,\rightarrow\}$ である．つまり，q_1,q_2 は任意の内部状態，s_1,s_2 は任意のテープ記号，d はヘッドの移動方向である．$(q_1,(s_1,s_2),q_2)$ は，T が状態 q_1 でテープ上の記号 s_1 を読み取った場合，テープに記号 s_2 を書き込み，内部状態を q_2 へ遷移させる規則を表している．また，(q_1,d,q_2) は，T の内部状態が q_1 の場合，ヘッドを d の示す方向に移動し，内部状態を q_2 へ遷移させる規則を表している．

チューリング機械の様相は，$(q,(l,s,r))$ で定義される．ここで，q は内部状態，s はヘッド下のテープ記号である．また，l はヘッド左側の，r はヘッド右側のテープの記号列を表す．

A.3 可逆チューリング機械の形式的な定義

可逆チューリング機械を形式的に定義するために，チューリング機械の前方・後方決定性について説明する．チューリング機械のすべての動作規則 $(q_1,(s_1,s_2),q_2)$，$(q_1',(s_1',s_2'),q_2')$ において，$q_1 = q_1'$ であり，$(s_1,s_2) \wedge (s_1',s_2') \wedge s_1 \neq s_1'$ であればチューリング機械は前方決定性を持つ．前方決定性は，遷移前の内部状態が等しい規則において，遷移前のヘッド下の記号が等しいものがないことを表す．また，チューリング機械のすべての動作規則 $(q_1,(s_1,s_2),q_2)$，$(q_1',(s_1',s_2'),q_2')$ において，$q_2 = q_2'$ であり，$(s_1,s_2) \wedge (s_1',s_2') \wedge s_2 \neq s_2'$ であればチューリング機械は後方決定性を持つ．後方決定性は，遷移後の内部状態が等しい規則において，遷移後のヘッド下の記号が等しいものがないことを表す．

前方・後方決定性の両方の性質を持つチューリング機械では，すべての動作規則において，内部状態とテープの記号列の組み合わせ (つまり様相) が同じになるものは一度として現れない．つまり，前方・後方決定性の両方の性質を持つチューリング機械の様相の遷移先はただ 1 つに決まり，様相遷移グラフは単射になる．以上より，可逆チューリング機械は前方・後方決定性の両方の性質を持つチューリング機械であると定義できる[*7]．

付録 B 今後のメモ

- 可逆な計算について，現在よりも詳しい定義，詳しいまとめを行う
- チューリング機械の振る舞い例について，[17] の定義で形式的表現可能な例を提示し，振る舞い例に対して形式的表現を行う

参考文献

[1] Frank, M.P. and Knight Jr, T.F.: Reversibility for efficient computing, PhD Thesis, MIT (1999).

[2] Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem, *J. of Math*, Vol.58, No.345-363, p.5 (1936).

[3] Hu, Z., Schürr, A., Stevens, P. and Terwilliger, J.: Bidirectional Transformation "bx" (Dagstuhl Seminar 11031), *Dagstuhl Reports*, Vol.1, No.1, pp.42–67 (2011).

[4] Stevens, P.: Bidirectional Transformations in the Large, *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp.1–11 (2017).

[5] Aman, B., Ciobanu, G., Glück, R., Kaarsgaard, R., Kari, J., Kutrib, M., Lanese, I., Mezzina, C.A., Mikulski, Ł., Nagarajan, R. et al.: Foundations of Reversible Computation, *International Conference on Reversible Computation*, Springer, pp.1–40 (2020).

[6] Tyagi, N., Lynch, J. and Demaine, E.D.: Toward an Energy Efficient Language and Compiler for (Partially) Reversible Algorithms, *Reversible Computation* (Devitt, S. and Lanese, I., Eds.), Cham, Springer International Publishing, pp.121–136 (2016).

[7] Landauer, R.: Irreversibility and heat generation in the computing process, *IBM journal of research and development*, Vol.5, No.3, pp.183–191 (1961).

[8] Bennett, C.H.: Logical reversibility of computation, *IBM journal of Research and Development*, Vol.17, No.6, pp.525–532 (1973).

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

[10] Lange, K.J., McKenzie, P. and Tapp, A.: Reversible space equals deterministic space, *Journal of Computer and System Sciences*, Vol.60, No.2, pp.354–367 (2000).

[11] Axelsen, H.B. and Yokoyama, T.: Programming techniques for reversible comparison sorts, *Asian Symposium on Programming Languages and Systems*, Springer, pp.407–426 (2015).

[12] Hall, J.S.: A Reversible Instruction Set Architecture and Algorithms, *Physics and Computation. Proceedings*, IEEE Press, pp.128–134 (1994).

[13] 増田大輝：効率的な可逆線形探索と木構造の可逆深さ優先探索の設計と解析，南山大学 2019 年度修士論文 (2020).

[14] Yokoyama, T.: Reversible computation and re-

[*7] 後方決定性のみを持つチューリング機械は，決定的チューリング機械である

versible programming languages, *Electronic Notes in Theoretical Computer Science*, Vol.253, No.6, pp.71–81 (2010).

[15] 戸川淳平，鳥居大樹，吉田翔亮：可逆線形探索の解析と可逆深さ優先探索の設計，南山大学 2019 年度卒業論文 (2020).

[16] 大久保雄飛，横山哲郎，金山知俊：二分木のランク計算のクリーン可逆シミュレーション，日本ソフトウェア科学会第 33 回大会講演論文集，pp.1–12 (2016).

[17] Yokoyama, T., Axelsen, H.B. and Glück, R.: Towards a reversible functional language, *International Workshop on Reversible Computation*, Springer, pp.14–29 (2011).

[18] 森田憲一：計算における可逆性 - 可逆チューリング機械と可逆論理回路 -，情報処理，Vol.35, No.4, pp.306–314 (1994).