可逆プログラミング言語 R-WHILE による 万能可逆チューリング機械の構成

2014SE006 青木　崚　2014SE059 増田　大輝　2014SE089 柴田　心太郎

指導教員：横山　哲郎

1　はじめに

計算できるという概念をチューリング機械 (以下，TM) で計算できるということにしようという主張は広く認められている [1]．TM は計算の効率を問題としなければ現在のコンピュータをも模倣できるとされている強力な計算モデルであり，計算可能性理論を議論する際に重要である．任意の TM を模倣できる計算システムは計算万能性 (チューリング完全) をもつといわれる．プログラミング言語の計算モデルが計算万能性をもつことを示すことは重要である．

可逆計算とは，計算過程のどの状態においても直前と直後にとり得る状態を高々 1 つもつものである．*R.Landauer* の研究により，計算機において非可逆計算で消散されるエネルギー量 (熱) は，kTln2(k はボルツマン定数，T は絶対温度) で与えられることがわかっている [2]．一方，可逆計算ではこのような不可避なエネルギーの発生を減らす 1 つの解決策として用いられる．すなわち，可逆計算では，理想的な状況下ではエネルギー消費を 0 にすることが可能であり，入力から出力までの過程を出力から逆算して入力を求めることが可能であるため，情報を消失することなく出力結果を導くことが出来る．可逆チューリング機械 (以下，RTM) が計算できるのは単射な計算可能関数であることが知られている [3]．可逆プログラミング言語とは，そのプログラムの実行過程が必ず可逆になるような言語設計がなされているプログラミング言語である．可逆プログラミング言語が RTM を模倣できること，すなわちその計算モデルが可逆的計算万能性をもつことを示すことは重要である．

可逆プログラミング言語 `R-WHILE` は，命令型の可逆プログラミング言語であり，我々の知る範囲では，`R-WHILE` が可逆的計算万能性をもつという報告はない．したがって，本稿では可逆プログラミング言語 `R-WHILE` によって万能可チューリング機械を構成することにより，`R-WHILE` が可逆的計算万能性をもつことの証明を目的とする．

2　関連研究

本章では，１章で述べた可逆計算に関連するものをはじめとした本稿に関連する研究について説明する．

2.1　Janus

`Janus` は `R-WHILE` と同様に可逆プログラミング言語の一種で，多重集合と配列の書木換えに基づく制約処理モデルを持つ．`Janus` は，C 言語に似た構文に加えて可逆性を保証するための構文規則を持つ．

2.2　セルオートマトン

セルオートマトンとは 1950 年代に *John von Neumann* が自己増殖オートマトンを設計するための理論的枠組みとして提案されたモデルである．いくつかの状態を持つセルという単位によって構成され，事前に設定された規則に従い，そのセル自身や近傍の状態によって，時間発展と共に，その時間における各セルの状態が決定される．現在では，計算システムの数理モデルの一種として扱われ，計算の基礎理論をはじめとして，交通流や生物などのシステムのシミュレーションに用いられている．

2.3　可逆セルオートマトン

可逆セルオートマトンとは，可逆性が保証されたセルオートマトン，つまり，現在の状態から 1 つ前の状態を一意に決めることができるセルオートマトンのことである．

3　可逆チューリング機械

本章では，TM と TM に制限を加えた RTM の定義を述べる．本稿では，簡単のため 1 テープの TM のみを考える．また，文献 [4] の表記を用いる．

3.1　チューリング機械

TM はマス目に分割された左右に無限長のテープをもち，有限制御部，及びテープ上の記号を読み書きするためのヘッドから構成されている (図 1)．テープには予め入力情報 (2 進数の 0 と 1 や多数のアルファベット) が格納されており，ヘッドが位置するマス目の記号を読み取る．そして, この記号と現在の有限制御部の状態 (内部状態，図 1においては状態 q を指す) に依存して，マス目の記号を書き換え，ヘッドを左か右に 1 コマ移動もしくは不動にし，内部状態を遷移させる．この一連の振る舞い (動作) を繰り返すことで計算を行う．

本稿では, 1 テープ TM を $T = (Q, \Sigma, b, \delta, q_s, Q_f)$ として定める．(以下, 1 テープ TM のことを TM と呼ぶことにする．) ただし Q は内部状態の空でない有限集合，Σ はテープ記号の空でない有限集合であり，$b(\in \Sigma)$ は空白記号でテープの有限個のマス目を除く残り全てのマス目に b が格納されていると仮定する．δ は $(Q\times[\Sigma\times\Sigma]\times Q)\cup(Q\times\{\leftarrow, \downarrow, \rightarrow\}\times Q)$ の部分集合，$q_s(\in Q)$ は初期状態，$Q_f(\subset Q)$ は最終状態の集合とする．

δ は遷移規則の集合である．矢印 $(\leftarrow, \downarrow, \rightarrow)$ はそれぞれヘッドの移動 (左，不動，右) を表す．遷移規則は 3 項組であり，$(q, \langle s, s'\rangle, q')$ または (q, d, q') である．

$(q, q' \in Q, s, s' \in \Sigma, d \in \{\leftarrow, \downarrow, \rightarrow\})$．前者の 3 項組は

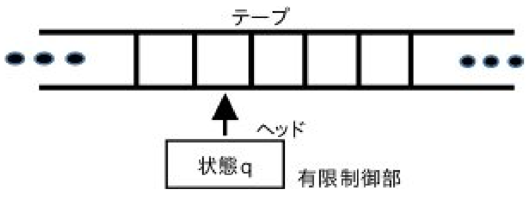

図 1: チューリング機械の模式図

T が状態 q で記号 s を読んだ場合，記号 s' に書き換え，状態を q' にすることを意味する．また，後者の 3 項組は T が状態 q の場合ヘッドを d の方向に動かし, 状態を q' にすることを意味する．遷移規則に 5 項組を用いることもあるが，5 項組で設計を行った場合，ヘッドの移動とテープの記号の書き換えを 1 つの遷移規則で行うことができる為,少ない遷移規則で記述することが出来る．一方 3 項組は,遷移規則の数は 5 項組と比べてヘッドの移動とテープの記号書き換えが別々なため多くなってしまう．しかし，遷移規則に対称性があるため，以降の章で説明する RTM を設計する際に前方決定的または後方決定的であるということの判断がしやすいため今回は 3 項組を用いる．

ここで，TM が計算を行っているある時点での様子を様相とする．このとき，TM $T = (Q, \Sigma, b, \delta, q_s, Q_f)$ の様相とは組 $(q, (l, s, r)) \in Q \times ((\Sigma \backslash \{b\})^* \times \Sigma \times (\Sigma \backslash \{b\})^*)$ である．ここで V^* は V 中の記号を 0 個以上並べたものの集合を表す．ただし q は内部状態,s はヘッドの下にある記号，l はヘッドの左側のテープを表す記号列，r はヘッドの右側のテープを表す記号列を表す．

TM $T = (Q, \Sigma, b, \delta, q_s, Q_f)$ の計算ステップは，$c \underset{T}{\Rightarrow} c'$ を満たすように様相 c を様相 c' に移すものとする．ただし，ここで, b つまり空白記号が無限に続くときを λ として

$$
\begin{array}{llll}
(q,(l,s,r)) & \underset{T}{\Rightarrow} (q',(l,s',r)) & \text{if} & (q,(s,s'),q') \in \delta \\
(q,(\lambda,s,r)) & \underset{T}{\Rightarrow} (q',(\lambda,b,sr)) & \text{if} & (q,\leftarrow,q') \quad \in \delta \\
(q,(ls',s,r)) & \underset{T}{\Rightarrow} (q',(l,s',sr)) & \text{if} & (q,\leftarrow,q') \quad \in \delta \\
(q,(ls,b,\lambda)) & \underset{T}{\Rightarrow} (q',(l,s,\lambda)) & \text{if} & (q,\leftarrow,q') \quad \in \delta \\
(q,(l,s,r)) & \underset{T}{\Rightarrow} (q',(l,s,r)) & \text{if} & (q,\downarrow,q') \quad \in \delta \\
(q,(l,s,\lambda)) & \underset{T}{\Rightarrow} (q',(ls,b,\lambda)) & \text{if} & (q,\rightarrow,q') \quad \in \delta \\
(q,(l,s,s'r)) & \underset{T}{\Rightarrow} (q',(ls,s',r)) & \text{if} & (q,\rightarrow,q') \quad \in \delta \\
(q,(\lambda,b,sr)) & \underset{T}{\Rightarrow} (q',(\lambda,s,r)) & \text{if} & (q,\rightarrow,q') \quad \in \delta
\end{array}
$$

である．$\underset{T}{\Rightarrow}$ の反射推移閉包を $\underset{T}{\Rightarrow}^*$ と記す．

TM $T = (Q, \Sigma, b, \delta, q_s, \{q_f\})$ の意味をとして，

$[\![T]\!] = \{(r, r') | (q_s, (\lambda, b, r)) \underset{T}{\Rightarrow}^* (q_f, (\lambda, b, r'))\}$ とする．これは初期状態 q_s でテープ内が (λ, b, r) の状態のとき，遷移を繰り返し最終状態になったときのテープ内が (λ, b, r') になるということを表している．

3.2 チューリング機械の例

これまでに定義したチューリング機械に基づいて，簡単なチューリング機械を考えてみる．

例　与えられた 2 進数に 1 を加えるチューリング機械 $T_1 = (Q_1, b, 0, 1, b, \delta_1, q_s, q_f)$ を考える．ただし，$Q_1 = \{q_s, q_1, q_2, q_3, q_4, q_f\}$ であり，δ_1 は以下の 3 項組の集合である．

$$
\begin{aligned}
\delta_1 = \{&[q_s, \langle b, b\rangle, q_1], [q_1, \rightarrow, q_2], \\
&[q_2, \langle 1, 0\rangle, q_1], \\
&[q_2, \langle 0, 1\rangle, q_3], [q_3, \leftarrow, q_4], \\
&[q_2, \langle b, 1\rangle, q_3], \\
&[q_4, \langle 0, 0\rangle, q_3], \\
&[q_4, \langle 1, 1\rangle, q_3], \\
&[q_4, \langle b, b\rangle, q_f]\}
\end{aligned}
$$

T_1 は，非負整数 n の 2 進数表現が書かれたテープが与えられ，ヘッドを 2 進数表現の左隣のます目に置いて初期状態 q_s から動作を開始したとき, n を n に 1 を加えたものに書き換え，2 進数表現の左隣のます目までヘッドを移動し，最終状態 q_f で停止するチューリング機械である (テープに書かれる 2 進数表現は反転されたものとする)．このとき，T_1 は 2 進数表現の一番下位の桁から読んでいく．1 を読んだとき，1 を 0 に書き換える．また，0 または b を読み込んだときにそれを 1 に書き換える．

3.3 可逆チューリング機械

TM T は任意の異なる遷移規則 $(q_1, a_1, q_1'), (q_2, a_2, q_2') \in \delta$ に対して，$q_1 = q_2$ ならば $a_1 = (s_1, s_1')$, $a_2 = (s_2, s_2')$ およびに $s_1 \neq s_2$ であるならば局所的に前方決定的であるという．また TM T は任意の異なる遷移規則 $(q_1, a_1, q_1'), (q_2, a_2, q_2') \in \delta$ に対して $q_1' = q_2'$ ならば $a_1 = (s_1, s_1')$, $a_2 = (s_2, s_2')$ および $s_1' \neq s_2'$ であるならば局所的に後方決定的であるという．

TM $T = (Q, \Sigma, b, \delta, q_s, q_f)$ は，局所的に前方決定的かつ後方決定的であり，最終状態からの遷移および初期状態への遷移がないとき，可逆と呼ぶ．このとき，RTM は最終状態を一つしかもたないため q_f とする．

4 可逆プログラミング言語 R-WHILE

ここでは `R-WHILE` について説明する．`R-WHILE` は，Jones の言語 `WHILE` を可逆化したものである．`R-WHILE` は非可逆なプログラムを記述することがでない．そのため単射関数しか表すことはできない．この言語の特徴は木構造のデータを表現できることである．それにより単純な方法で身近なデータ構造をモデリングが可能である, これは,既存の可逆命令言語 `Janus` では難しいことである．
`R-WHILE` の構文規則は単純である (図 2)．プログラム `P` は

```
E,F ::= X | d | cons E F | hd E | tl E | =? E F     式
Q,R ::= X | d | cons Q R                              パターン
C,D ::= X ^= E                                        命令
      | Q <= R
      | C; D
      | if E then C else D fi F
      | from E do C loop D until F
P   ::= read X; C; write Y                            R-WHILE のプログラム
```

図 2: 言語 R-WHILE の構文規則

$$
\begin{array}{lcl}
\mathcal{I}[\![\texttt{X ^= E}]\!] & = & \texttt{X ^= E} \\
\mathcal{I}[\![\texttt{Q <= R}]\!] & = & \texttt{Q <= R} \\
\mathcal{I}[\![\texttt{C; D}]\!] & = & \mathcal{I}[\![\texttt{D}]\!]\texttt{; }\mathcal{I}[\![\texttt{C}]\!] \\
\mathcal{I}[\![\texttt{if E then C else D fi F}]\!] & = & \texttt{if F then }\mathcal{I}[\![\texttt{C}]\!]\texttt{ else }\mathcal{I}[\![\texttt{D}]\!]\texttt{ fi E} \\
\mathcal{I}[\![\texttt{from E do C loop D until F}]\!] & = & \texttt{from F do }\mathcal{I}[\![\texttt{C}]\!]\texttt{ loop }\mathcal{I}[\![\texttt{D}]\!]\texttt{ until E} \\
\mathcal{I}[\![\texttt{read X; C; write Y}]\!] & = & \texttt{read Y;}\mathcal{I}[\![\texttt{C}]\!]\texttt{; write X}
\end{array}
$$

図 3: R-WHILE の逆変換器 $\mathcal{I}$

ただ一つの入口と出口の点 (read,write) をもち，命令 C がプログラムの本体である．

プログラムのデータ領域 $\mathbb{D}$ はアトム nil とすべての組 $(\mathtt{d}_1, \mathtt{d}_2)$ を含む（$\mathtt{d}_1, \mathtt{d}_2 \in \mathbb{D}$）最小の集合である．$Var$ は変数名の無限集合である．本稿では $\mathtt{d}, \mathtt{e}, \mathtt{f}, ... \in \mathbb{D}$ とする．また，$\mathtt{X}, \mathtt{Y}, \mathtt{Z}, ... \in Vars$ とする．

式は変数 X，定数 d，または複数の演算子からなる (先頭とそれ以降を表す hd と tail，組を表す cons また等号を表す=?）からなる．パターンは式の部分集合であり，変数 X，定数 d またはパターンのペアを表す cons Q R からなる．本稿では以後ペアを表す cons E F または cons Q R をそれぞれ (E.F) または (Q.R) と表記する．パターンは線形的でなくてはならない．すなわち，反復変数は含まれない．また，この言語は局所変数をもたない．

次に命令 C について説明する．非可逆なプログラミング言語における代入は左辺の変数の値を上書きする．そして，代入後に再び値を取り戻すことはできない．そのため，可逆プログラミング言語に用いることは出来ない．可逆的代入 X ^= E では，X の値が nil のとき X の値を E の値にする．また，X の値が E の値と等しいとき，X の値を nil にする．X ^= E でなぜなら，X ^= X のとき，X がどのような値であっても X が nil の値になるため，単射ではなくなってしまうからである．可逆的置換 Q <= R は Q の変数を R の変数を使って更新する．例えば

{Q = nil,R = a}
Q <= R;
{Q = a,R = nil}

となる．可逆的代入と比べ両辺に表されるのはパターンの Q と R である．

R-WHILE の２つの構造化された制御フロー演算子は条件文の if E then C else D fi と繰り返し文の from E do C loop D until F である．繰り返し文はエントリーアサーション E をもち，条件文は出口アサーション F をもつ．

可逆的条件文 if E then C else D fi F の制御フローの分岐はテスト E に依存する．もし真であれば命令 C が実行され，アサーション E は真でなくてはならない．また，もし偽であった場合命令 D が実行され，アサーション E は偽でなくてはならない．E と F の返す値が対応していない場合，条件文は定義されない．可逆的ループ from E do C loop D until F は繰り返しを行うとき，アサーション E は真でなくてはならない．そして，命令 C が実行される．実行後のテスト F が真であれば繰り返しは続行される．もし F が偽であった場合，命令 D が実行される．また，アサーション E は偽でなくてはならない．

可逆プログラミング言語である R-WHILE にはプログラミング逆変換器 (I)(図 3) が定義されている．それにより反転されたプログラムを再帰的降下によって得ることができる．

5　RTM から R-WHILE への変換

図 4aにチューリング機械プログラムからの変換で得られた R-WHILE プログラムを示す．なお記述には変換規則を用いた．

main プログラム (図 4a) は入力としてチューリング機械のテープ上に書かれている記号列 R を読み込む．その後，計算を実行し，書き換えられた記号列 R' を出力するというものである．main プログラム本体の Q は TM の内部状態を表している．また，T は TM のテープの状態を表している．そのため，可逆的代入 Q ^= $\overline{q_s}$; によって，内部状態を表す Q は初期状態になる．また可逆的置換 T <= (nil

```
read R;
  Q ^= $\overline{q_s}$;
  T <= (nil $\bar{b}$ R);
  from (=? Q $\overline{q_s}$) loop
    STEP(Q,T)
  until (=? Q $\overline{q_f}$);
  (nil $\bar{b}$ R’) <= T;
  Q ^= $\overline{q_f}$;
write R’
```

(a) main プログラム

```
macro STEP(Q,T) ≡
rewrite [Q,T] by
 $\mathcal{T}[\![t]\!]^*$
```

(b) マクロ STEP

```
macro MOVEL(T) ≡
(L S R) <= T;
PUSH(S,R);
POP(S,L);
T <= (L S R)
```

(c) マクロ MOVEL

```
macro PUSH(S,STK) ≡
rewrite [S,STK] by
  [$\bar{b}$,nil] => [nil,nil]
  [S,STK] => [nil,(S.STK)]
```

(d) マクロ PUSH

図 4: RTM を模倣する R-WHILE プログラム

$$
\begin{aligned}
&\mathcal{T}[\![\langle q_1, \langle s_1, s_2\rangle, q_2\rangle]\!] = \\
&\qquad \texttt{[}\overline{q_1}\texttt{,(L }\overline{s_1}\texttt{ R)] => [}\overline{q_2}\texttt{,(L }\overline{s_2}\texttt{ R)]} \\
&\mathcal{T}[\![\langle q_1, \leftarrow, q_2\rangle]\!] = \\
&\qquad \texttt{[}\overline{q_1}\texttt{,T] => \{MOVEL(T); Q \^{}=}\overline{q_1}\texttt{; Q \^{}=}\overline{q_2}\texttt{\}} \\
&\mathcal{T}[\![\langle q_1, \rightarrow, q_2\rangle]\!] = \\
&\qquad \texttt{[}\overline{q_1}\texttt{,T] => \{MOVER(T); Q \^{}=}\overline{q_1}\texttt{; Q \^{}=}\overline{q_2}\texttt{\}} \\
&\mathcal{T}[\![\langle q_1, \downarrow, q_2\rangle]\!] = \texttt{[}\overline{q_1}\texttt{,T] => [}\overline{q_2}\texttt{,T]}
\end{aligned}
$$

図 5: 遷移規則から R-WHILE の書き換え規則への変換

$\bar{b}$ R); によって，ヘッドがテープにかかれている記号列の一つ左を指している状態を表している．

そして，組 (Q,T) は TM の様相を表している．プログラム内のループでは，TM の内部状態が初期状態から最終状態に遷移するまでマクロ STEP が繰り返し実行される．繰り返しを終了した後，命令によって T と Q の値は nil となる．

図 4bで定義されるマクロ STEP(Q,T) では，様相 (Q,T) を書き換え規則により書き換える．$\mathcal{T}[\![t]\!]^*$ は遷移規則から R-WHILE の書き換え規則への変換器 $\mathcal{T}$(図 2) によって生成された書き換え規則の列である．変換器 $\mathcal{T}$ によって

それぞれの書き換え規則は図 5の変換器 $\mathcal{T}$ によって遷移規則 $t(\in \delta)$ から生成される．$\bar{q}$ は，状態 q に対応する R-WHILE のアトムである．RTM の遷移規則列を変換した場合，異なる書き換え規則は矢印=>の左側のパターンと右側で返却される値がそれぞれ重なることはない．

マクロ MOVEL(図 4c) はヘッドを一つ左に動かすためのマクロである．チューリング機械のテープ (l, s, r) はスタック L と R を用いて (L S R) として表す．マクロ MOVEL はマクロ PUSH と POP を実行し変化したテープの状態を T に置き換える．ヘッドを一つ右に動かす為のマクロ MOVER はマクロ MOVEL を逆変換することで得ることができる． マクロ PUSH は，アトム S をスタック STK にプッシュするためのマクロである．S が空白記号の場合，S，STK ともに nil をかえす．マクロ POP はマクロ PUSH を逆変換することで得ることができる．ただし，スタック STK が nil だった場合，マクロ POP は空白記号をポップする．POP(S,STK) ではスタックが空の場合，空白記号がポップされる．この操作によってスタックのボトムが非空白記号である状態（無限のテープの中で有限の記号列を表す）を保つことができる為，任意の回数行うことができる．プッシュの逆操作であるポップ POP(S,STK) は $\mathcal{I}[\![$PUSH(S,STK)$]\!]$ とする．

6 おわりに

本稿の 5 章において任意の RTM から R-WHILE プログラムへの変換 $\mathcal{T}$ を R-WHILE の書き換え規則によって定義した．従って言語 R-WHILE によって任意の RTM プログラムを書けるということである．以上より R-WHILE の計算モデルは可逆的にチューリング完全であると言うことを示した．すなわち可逆プログラミング言語 R-WHILE によって万能可逆チューリング機械が作れることが示された．

今後は R-WHILE が RTM を模倣できていることを証明し，RTM から R-WHILE の変換を示したい．

参考文献

[1] Stephen C. Kleene:The Church-Turing Thesis, Stanford Encyclopedia of Philosophy(online), available from <https://plato.stanford.edu/entries/church-turing/> (accessed2017-09-27).

[2] R. Landauer.: Irreversibility and Heat Generation in the Computing Process, *IBM Journal of Research and Development.* Vol. 5, No. 3, pp. 183–191(1961).

[3] Axelsen, H. B. and Glück, R.: What Do Reversible Programs Compute?, *Foundations of Software Science and Computational Structures. Proceedings*(Hofmann, M. , ed.), LNCS, Vol6604, Springer-Verlag, pp. 42–56(2011)

[4] Yokoyama, T. , Axelsen, H. B. and Glück, R.: Towards a Reversible Functional Language, *Reversible Computation.Proceedings*(De Vos, A. and Wille, R . , eds.), LNCS, Vol. 7165, SpringerVerlag, pp. 14–29(2012)

[5] Jones, N. D.: Computability and Complexity: *From a Programming Perspective*, MIT Press (1997). Revised version, available from `http://www.diku.dk/ neil/Comp2book.html`.