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，計算可能性理論の議論の際に重要である．また，任意の TM を模倣できる計算システムは計算万能性をもつといわれる．そのため，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可逆チューリング機械 (以下，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もまた重要である．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 [2]．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．従って，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（以下，URTM）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 と同様に命令型の可逆プログラミング言語の一種で，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　WHILE 言語

WHILE 言語とは，命令型言語である.WHILE は単純な言語でありながら．TM を模倣できるくらいの計算能力を持っているため，プログラムやプログラムの振る舞いについての定理を証明する場合に重宝される．

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 [3] 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)．テープには予め記号列が格納されており，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 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．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．

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 はヘッドの上にある記号，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 が 2 個以上続くときを λ と表記して

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

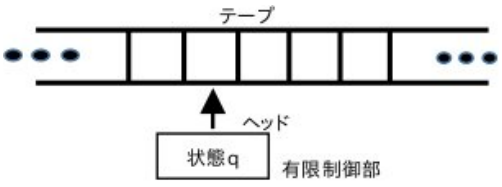

図 1: チューリング機械の模式図

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) の様相であるとき，遷移を繰り返し行った結果が最終状態 q_f でテープ内が (λ, b, r') の様相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 TM に基づいて，簡単な TM を考えてみる．

例　与えられた 2 進数に 1 を加える TM　$T_1 = (Q_1, \{b, 0, 1\}, b, \delta_1, q_s, \{q_f\})$ 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．

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

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 進数表現の左隣のマス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 進数表現の左隣のマス目までヘッドを移動し，最終状態 q_f で停止する TM 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の TM を RTM 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．

3.4 万能可逆チューリング機械

URTM とは，任意の RTM を可逆的に模倣したもので，RTM と入力した情報をエンコードしたものを受け取ってそれを出力とする．

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きない．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．

与えられたリストを反転させて表示するプログラム例 reverse を用いていくつかの言語の機能について説明する．

```
read X;（リストを反転させるプログラム）
  from (=? Y nil)
  loop (Z.X) <= X;
       Y<= (Z.Y)
  until(=? X nil);
write Y
```

プログラムの入力は変数 X を読み込む．そして，出力は変数 Y を書き出す．すべての変数の初期値は nil である．可逆ループを行う命令，loop...until は Y が nil であると主張されたとき，繰り返しが行われ，X が nil のときに繰り返しが終了する．loop 内の命令はテストとアサーションが偽である限り繰り返す．loop 内の最初の命令 ((Z.X) <= X) では X の値を先頭とそれ以降の部分に分解をしている．2 番目の命令（Y <= (Z.X)）では，Z と Y を組にした値を構成している．可逆置換右辺の元の値を左辺の変数に束縛する前に，右辺の値を nil にセットする (たとえば Y <= (Z.Y) が実行された後 Z は nil である)．出力の変数である Y を除くすべての変数は最終的に nil でなくてはならない．

R-WHILE の構文規則は図 2のように定義される．プログラム P は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．$Vars$ は変数名の無限集合である．本稿では $\mathtt{d}, \mathtt{e}, \mathtt{f}, \ldots \in \mathbb{D}$ とする．また，$\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \ldots \in Vars$ 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 またはパターンの組を表す cons Q R からなる．本稿では以後組を表す 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．

```
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

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．
可逆置換 Q <= R はまず，R の値を nil とし，Q の値を R の変数を使って更新する．可逆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．繰り返し文はテスト E をもち，条件アサーション F をもつ．

可逆条件文 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．可逆ループ from E do C loop D until F は繰り返しを行うとき，テスト E は真でなくてはならない．そして，命令 C が実行される．実行後のアサーション 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) 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．この逆変換器によって求められたプログラムは元のプログラムと同じ働きをする．

5 RTM から R-WHILE への変換

図 3a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．またプログラムにマクロを使用した．マクロ展開することで，右辺の変数は実引数に置き換えられる．RTM に対応する R-WHILE の記述を $\bar{\cdot}$ を用いて表す．

main プログラム (図 3a) は入力として RTM 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．また可逆置換 T <= (nil $\bar{b}$ 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．

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 3b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 4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 3c) はヘッドを一つ左に動かすためのマクロである．RTM 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．この操作によって無限のテープの中で有限の記号列を表している状態を保つことができる．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 証明

ここでは，(図 4) で与えた遷移規則から R-WHILE の書き変え規則への変換が正しいことを示す．これを証明する

```
read R;
  Q ^= q_s;
  T <= (nil b̄ R);
  from (=? Q q_s) loop
    STEP(Q,T)
  until (=? Q q_f);
  (nil b̄ R') <= T;
  Q ^= q_f;
write R'
```

(a) main プログラム

```
macro STEP(Q,T) ≡
rewrite [Q,T] by
 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
 [b̄,nil] => [nil,nil]
 [S,STK] => [nil,(S.STK)]
```

(d) マクロ PUSH

図 3: RTM を模倣する R-WHILE プログラム

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

図 4: 遷移規則から R-WHILE の書き換え規則への変換

ために，**補題 1** と**補題 2** を証明する必要がある．それぞれの証明の詳細は本稿にて記述する．
RTM の様相を表す $(q,(l,s,r))$ は $(q,(l,s,r)) \in Q \times ((\Sigma\backslash\{b\})^* \times \Sigma \times (\Sigma\backslash\{b\})^*)$ である．ここでは，$\mathcal{T}[\![(q,(l,s,r))]\!] = \{\mathtt{Q} \mapsto \overline{q},\ \mathtt{T} \mapsto (\bar{l}\ \bar{s}\ \bar{r})\}$ とする．また，$\mathcal{T}[\![x]\!]$ を $\bar{x}$ の様に記述している．可逆チューリング機械における任意の様相を c または c の様に表し，$\underset{T}{\Rightarrow}$ は可逆チューリング機械の計算ステップにおいて，様相 c を様相 c' に遷移していることを表す．

補題 1　任意の様相 c，c' に対して，

$$c \underset{T}{\Rightarrow} c'$$

のとき，

$$\mathcal{C}[\![\mathtt{STEP(Q,T)}]\!]\ (\bar{c}) = \overline{c'}$$

が成り立つ．

補題 1 を証明することで，可逆チューリング機械の様相の遷移に命令マクロ `STEP(Q,T)` を実行することによる状態の変化が対応していることが示される．証明の手順としては RTM の様相の遷移が R-WHILE における状態の変化に対応しているかを示した．

補題 2　任意の様相 c，c' と自然数 n に対して，

$$c \underset{T}{\Rightarrow}^n c'$$

であるならば，

$$(\mathcal{C}[\![\mathtt{STEP(Q,T)}]\!](\bar{c}))^n = \overline{c'}$$

が成り立つ．

補題 2 を証明することで，様相の遷移が複数回行われた場合でも**補題 1** が成り立つことを示される．証明には数学的帰納法を用いた．二つの補題によって**定理 1** が成り立つ．

定理 1　任意の可逆チューリング機械 T，任意の r に対して

$$[\![T]\!]^{TM} r = R2T[\![[\![\mathcal{T}[\![T]\!]]\!]^{\mathtt{R\text{-}WHILE}}(T2R[\![r]\!])]\!]$$

である．

変換器 $R2T$ は R-WHILE の $\mathbb{D}$ を可逆チューリング機械の Σ^+（1 つ以上の集合）へと変換するものであり，定義は以下のようである．
$R2T : \mathbb{D} \rightarrow \Sigma^+$
$R2T[\![\mathtt{nil}]\!] = 0$
$R2T[\![(\mathtt{d_1}.\mathtt{d_2})]\!] = 1 \cdot R2T[\![\mathtt{d_1}]\!] \cdot R2T[\![\mathtt{d_2}]\!]$
このとき，$\cdot$ は連接を表す．また，$R2T$ の逆である Σ^+ を $\mathbb{D}$ へ変換する変換器を $T2R$ としている．

したがって，R-WHILE は URTM を模倣できる．

7　おわりに

本稿の 5，6 章において任意の RTM から R-WHILE プログラムへの変換器 $\mathcal{T}$ を R-WHILE の書き換え規則によって定義し，それを証明した．従って可逆プログラミング言語 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 は URTM を模倣できる．そのため，R-WHILE の計算モデルは可逆計算万能性をもつ．すなわち，可逆プログラミング言語 R-WHILE は可逆計算万能性をもつ．

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] 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).

[3] 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, Springer-Verlag, pp. 14–29(2012).

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