可逆プログラミング言語 R-WHILE の可逆チューリング完全性

青木　峻[†]　　横山　哲郎[†]（正員）

r-Turing Completeness of Reversible Programming Language R-WHILE

Ryo AOKI[†], *Nonmember* and Tetsuo YOKOYAMA[†], *Member*

[†] 南山大学理工学部ソフトウェア工学科
Department of Software Engineering, Faculty of Science and Engineering, Nanzan University

あらまし　任意の可逆プログラミング言語のプログラムで計算できるものが，可逆な R-WHILE プログラムによって計算できることを示す．

キーワード　可逆プログラミング言語，可逆チューリング機械，可逆チューリング完全性，R-WHILE

1. まえがき

チューリング機械（以下，TM）で計算できるものを計算の定義とするチャーチ・チューリングの提唱が知られている．任意の TM を模倣できる TM を万能とよび，TM と同じ計算能力をもつ計算モデルをチューリング完全とよぶ．プログラミング言語の計算モデルがチューリング完全であること，すなわち計算能力が高いことは重要である．汎用プログラミング言語の多くはチューリング完全であると示されている．

可逆計算とは，計算過程の任意の状態に対して直前と直後にとり得る状態が高々 1 つである計算である．可逆チューリング機械（以下，RTM）には，任意の RTM と入力を受け取り可逆的にその RTM を模倣してその RTM と入力に対応する出力を返す可逆的に万能なものが存在することが知られている [1]．RTM を可逆的に模倣できるプログラミング言語は可逆的にチューリング完全という．可逆プログラミング言語とは，その言語で書かれたプログラムの全ての原子的な計算ステップの意味が必ず単射になることが保証されたプログラミング言語である．汎用可逆プログラミング言語が可逆的にチューリング完全であること，すなわち高い計算能力をもつことを保証されていることは重要である．

可逆プログラミング言語 R-WHILE は，Jones の言語 WHILE [2] を可逆化したものであり，任意の命令 C に対して逆命令 $\mathcal{I}[\![\mathtt{C}]\!]$ があるという特徴をもち，木構造のデータをもつ．我々の知る範囲では，R-WHILE が可逆的にチューリング完全であることの具体的な証明の報告はない．

本稿では，任意の RTM を整形な R-WHILE プログラムにその意味を変えずに変換できることを示す．すなわち，可逆プログラミング言語 R-WHILE が可逆的にチューリング完全であることを示す．

2. 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．

2.1 チューリング機械

TM を $T = (Q, \Sigma, b, \delta, q_s, q_f)$ として定める．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，b $(\in \Sigma)$ は空白記号，δ は $((Q/\{q_f\}) \times [\Sigma \times \Sigma] \times (Q/\{q_s\})) \cup ((Q/\{q_f\}) \times \{\leftarrow, \downarrow, \rightarrow\} \times (Q/\{q_s\}))$ の部分集合，q_s $(\in Q)$ は初期状態，q_f $(\in Q)$ は最終状態とする．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 $(q, (s, s'), q')$ または (q, a, q') である $(q, q' \in Q, s, s' \in \Sigma, a \in \{\leftarrow, \downarrow, \rightarrow\})$．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．また，後者の 3 項組は T が状態 q の場合ヘッドを a の方向に動かし，状態を q' にすることを意味する．

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．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の記号列，r はヘッドの右側のテープの記号列を表す．λ は空語を表す．

TM $T = (Q, \Sigma, b, \delta, q_s, q_f)$ の計算ステップは，図 1 のように定義され，$c \underset{T}{\Rightarrow} c'$ を満たすように様相 c を様相 c' に更新する．ただし，ヘッドを 1 つ右へ動かす単射関数は

$$
\begin{aligned}
movel(\lambda, s, r) &= (\lambda, b, sr) \\
movel(ls', s, r) &= (l, s', sr) \\
movel(ls, b, \lambda) &= (l, s, \lambda)
\end{aligned}
$$

とする．$\underset{T}{\Rightarrow}$ の反射推移閉包を $\underset{T}{\Rightarrow}^*$ と記す．特に T の遷移規則 d $(\in \delta)$ で遷移する場合，$\underset{T}{\Rightarrow}$ を $\underset{d}{\Rightarrow}$ と記す．

TM $T = (Q, \Sigma, b, \delta, q_s, q_f)$ の意味を以下とする：

$$
[\![T]\!]^{\mathtt{TM}} r = r' \Longleftrightarrow (q_s, (\lambda, b, r)) \underset{T}{\Rightarrow}^* (q_f, (\lambda, b, r'))
$$

$$
\begin{aligned}
(q,(l,s,r)) &\underset{T}{\Rightarrow} (q',(l,s',r)) \quad \text{if} \quad (q,(s,s'),q') \in \delta \\
(q,(l,s,r)) &\underset{T}{\Rightarrow} (q',(l',s',r')) \quad \text{if} \quad (q,\leftarrow,q') \quad\ \in \delta \\
&\qquad \text{where } \mathit{movel}(l,s,r) = (l',s',r') \\
(q,(l,s,r)) &\underset{T}{\Rightarrow} (q',(l,s,r)) \quad \text{if} \quad (q,\downarrow,q') \quad\ \in \delta \\
(q,(l,s,r)) &\underset{T}{\Rightarrow} (q',(l',s',r')) \quad \text{if} \quad (q,\rightarrow,q') \quad\ \in \delta \\
&\qquad \text{where } (l,s,r) = \mathit{movel}(l',s',r')
\end{aligned}
$$

図 1: 計算ステップ $c \underset{T}{\Rightarrow} c'$

Fig. 1 Computation step $c \underset{T}{\Rightarrow} c'$.

2.2 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 は，局所的に前方決定的かつ局所的に後方決定的であり，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．

3. 可逆プログラミング言語 R-WHILE

本章では，可逆プログラミング言語 R-WHILE のデータ，構文，意味論について，本論文で必要な範囲を簡潔に述べる．詳しい内容は文献 [3] を参照されたい．

可逆プログラミング言語 R-WHILE のデータ領域 $\mathbb{D}$ は，nil およびデータ領域 $\mathbb{D}$ の任意の 2 つの値 d_1, d_2 の組 $(d_1 . d_2)$ の全てからなる最小の集合である．$\mathbb{D}$ の要素は，真偽値として評価されるところでは nil で偽，それ以外で真と呼ぶことにする．

R-WHILE プログラムは，単射な意味をもつ命令列からなる．可逆代入命令 x `^=` e は，変数 x の値が nil ならば e の値を x に格納し，変数 x の値が v ならば e の値が v であることを確認して x の値を nil にする．可逆置換命令 q `<=` r は，命令を実行する直前の状態におけるパターン r の値が命令を実行する直後の状態におけるパターン q の値に一致するように，r に含まれる全ての変数を nil にしたのちに p に含まれる全ての変数に値を束縛する．可逆条件命令 `if` e `then` c `fi` f は，テスト e の値が真ならば命令 c を実行してアサーション f の値が真であることを確認し，テスト e の値が偽ならば命令 c を実行してアサーション f の値が偽であることを確認する．可逆繰返し命令 `from` e `do` c `loop` d `until` f は，アサーション e の値が真であることを確認し命令 c を実行し，以降はテスト f が偽でなくなるまで，命令 d を実行しアサーション e が偽であることを確認して命令 c を実行することを繰り返し続ける．

命令 c の意味を $\mathcal{C}[\![c]\!]$, R-WHILE プログラム p の意味を $[\![p]\!]^{\text{R-WHILE}}$ で表す．R-WHILE の任意の命令 c に対して意味が逆関数になる逆命令 $\mathcal{I}[\![c]\!]$ が必ず存在することが知られている．

4. RTM から R-WHILE への変換

TM の任意のデータや状態 x を単射 $\overline{x}$ で R-WHILE のデータに，単射 $\underline{x}$ で R-WHILE のプログラム片に符号化することにする．TM $T = (Q,\Sigma,b,\delta,q_s,q_f)$ を模倣する R-WHILE プログラム $\underline{T}$ を図 2a に示す．ここでマクロ `STEP(Q,T)` の定義は，図 2b の通りである．$\underline{\delta}$ は，様相を表す符号を格納する組 `[Q,T]` を書き換えるマクロ `rewrite [Q,T] by` $\underline{d}^*$ である．ここで各遷移規則 d は δ の要素であり，書き換え規則 $\underline{d}$ は，図 3 の通りである．`rewrite` マクロの書き換え規則の任意の 2 つの左辺では，$\overline{s_1}$ が互いに現れた場合は $\overline{q_1}$ と $\overline{s_1}$ の組，現れない場合は互いの $\overline{q_1}$ が異なることが前提で条件分岐する．`rewrite` マクロの書き換え規則の任意の 2 つの右辺では，$\overline{s_2}$ が互いに現れた場合は $\overline{q_2}$ と $\overline{s_2}$ の組，現れない場合は互いの $\overline{q_2}$ が異なることが前提で制御が合流される．変換前の TM が可逆の場合はこの 2 つの前提が必ず満たされる．任意の様相を表す符号に対して制御の分岐と合流がこのように保証されることを全ての書き換え規則が直行するという．

図 2c の MOVEL はヘッドを 1 つ左に動かすものである．テープ (l,s,r) は，スタック L と R を用いて (L S R) で表す．ヘッドを 1 つ左に動かすのは，スタック R からヘッダの下にある記号 S を PUSH(S,R) によりプッシュし，スタック L から 1 つ要素を POP(S,L)

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

(a) main プログラム

```
macro STEP(Q,T) ≡ δ̲
```

(b) マクロ STEP

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

(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 2: RTM を可逆的に模倣する R-WHILE プログラム
Fig. 2 An R-WHILE program for performing reversible simulations of RTMs.

$$
\begin{aligned}
\underline{(q_1,(s_1,s_2),q_2)} &= [\overline{q_1},(\mathtt{L}\ \overline{s_1}\ \mathtt{R})]\ \mathtt{=>}\ [\overline{q_2},(\mathtt{L}\ \overline{s_2}\ \mathtt{R})]\\
\underline{(q_1,\leftarrow,q_2)} &= [\overline{q_1},\mathtt{T}]\ \mathtt{=>}\ \{\mathtt{MOVEL(T);\ Q\ \hat{}=}\overline{q_1}\mathtt{;\ Q\ \hat{}=}\overline{q_2}\}\\
\underline{(q_1,\downarrow,q_2)} &= [\overline{q_1},\mathtt{T}]\ \mathtt{=>}\ [\overline{q_2},\mathtt{T}]\\
\underline{(q_1,\rightarrow,q_2)} &= [\overline{q_1},\mathtt{T}]\ \mathtt{=>}\ \{\mathtt{MOVER(T);\ Q\ \hat{}=}\overline{q_1}\mathtt{;\ Q\ \hat{}=}\overline{q_2}\}
\end{aligned}
$$

図 3: 書き換え規則 $\underline{d}$
Fig. 3 Translation rules $\underline{d}$.

によってポップしてヘッダの下を表す記号 S とすることで模倣する．ヘッドを右に 1 つ動かす MOVER は，MOVEL の逆命令 $\mathcal{I}[\![\mathtt{MOVEL(X)}]\!]$ である．

要素 S をスタック STK にプッシュするマクロ PUSH(S,STK) を図 2d のように定義する．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 $\mathcal{I}[\![\mathtt{PUSH(S,STK)}]\!]$ とする．

5. R-WHILE の可逆的なチューリング完全性

本章では，任意の RTM T に対して，R-WHILE プログラム $\underline{T}$ の意味が T の意味と一致することを示す．まずは，各遷移規則による様相の更新を R-WHILE プログラムで模倣できることを次の補題で示す．

補題 1. 任意の RTM T, 様相 c, c' に対して，δ を T の遷移規則の集合とすると，$d \in \delta$ が存在して，

$$c \underset{d}{\Rightarrow} c' \implies \overline{c'} = \mathcal{C}[\![\underline{\delta}]\!]\overline{c} \qquad (1)$$

が成り立つ．

証明. RTM T, T の遷移規則の集合 δ, 様相 $c = (q,(l,s,r))$, $c' = (q',(l',s',r'))$ を仮定する．T は RTM であるので遷移規則は局所的に前方および後方決定的であるから，$c \underset{d}{\Rightarrow} c'$ となる d は高々 1 つである．d が存在しない場合は前提が偽であるので，式 1 は成り立つ．以下，$d = (q_1, a, q_2)$ が存在し，また $\overline{c} = \{\mathtt{Q} \mapsto \overline{q}, \mathtt{T} \mapsto (\overline{q}\ \overline{s}\ \overline{r})\} \cup \sigma$ かつ $\overline{c'} = \{\mathtt{Q} \mapsto \overline{q'}, \mathtt{T} \mapsto (\overline{q'}\ \overline{s'}\ \overline{r'})\} \cup \sigma'$ となる σ と σ' が存在する場合を考える．$\overline{\delta}$ が Q と T の値しか更新しないことから $\sigma = \sigma'$ である．

a について場合分けを行う．

$a = (s_1, s_2)$ の場合．$\underset{d}{\Rightarrow}$ の定義より，$d = (q,(s,s'),q')$ である．書き換え規則が直行することから，$\underline{d}$ によって $\overline{c}$ は $\overline{c'}$ に更新される．

$a = \leftarrow$ の場合．$\underset{d}{\Rightarrow}$ の定義より，$d = (q, \leftarrow, q')$ である．書き換え規則が直行することから，$\underline{d}$ によって $\overline{c}$ は更新される．MOVEL の定義より，σ'' が存在して，$\mathcal{C}[\![\mathtt{MOVEL((L\ S\ R))}]\!](\{\mathtt{L} \mapsto \overline{l}, \mathtt{S} \mapsto \overline{s}, \mathtt{R} \mapsto \overline{r}\} \cup \sigma'') = \{\mathtt{L} \mapsto \overline{l'}, \mathtt{S} \mapsto \overline{s'}, \mathtt{R} \mapsto \overline{r'}\} \cup \sigma''$ かつ $movel(\overline{l}, \overline{s}, \overline{r}) = (\overline{l'}, \overline{s'}, \overline{r'})$ である．

$a = \downarrow$ の場合．この場合は自明であり，$a = \rightarrow$ の場合は $a = \leftarrow$ の場合と同様に示される．

以上より全ての a について式 1 が成り立つ． □

次に，R-WHILE プログラムによって任意の RTM が模倣できることを示す．

定理 2. 任意の RTM T, 空白記号を含まない記号列 r, r' に対して，

$$r' = [\![T]\!]^{\text{TM}} r \implies \overline{r'} = [\![\underline{T}]\!]^{\text{R-WHILE}} \overline{r} \qquad (2)$$

が成り立つ．

証明． 補題 1 によりマクロ `STEP(Q,T)` によって各遷移規則の適用を R-WHILE プログラムで模倣できることがいえた．$(q_s, (\lambda, b, r)) \underset{T}{\Rightarrow}^* (q_f, (\lambda, b, r'))$ を図 2a の main プログラム中の可逆ループで模倣できることは $\underset{T}{\Rightarrow}$ の出現回数の上の帰納法で示される．main プログラムの残りのコードは変数の初期化，値の入れ替え，および値を `nil` にクリアすることのみを行っている． □

この定理により R-WHILE は可逆的にチューリング完全である．さらに，RTM の中には可逆的に万能なものが存在するので，可逆的に万能な RTM の R-WHILE プログラムにより任意の RTM を可逆的に模倣することができる．

6. ま と め

R-WHILE 言語は，可逆なので任意の TM を模倣することはできない [1] が，本稿において任意の可逆プログラミング言語で計算できるものは計算できることが証明された．すなわち，R-WHILE 言語は，可逆的にチューリング完全であると示された．万能 RTM U の意味を持つ R-WHILE プログラムは $\underline{U}$ であり，$\underline{U}$ は任意の RTM を可逆的に模倣できる．

今後は，R-WHILE の解釈系を実現することでその解釈系を実現するプログラミング言語が可逆的にチューリング完全であることを示せる．

謝辞 本論文は，情報処理学会第 79 回全国大会講演論文集における原稿および南山大学理工学部 2017 年度卒業研究論文の一部を改訂したものである．本研究は 2018 年度南山大学パッヘ研究奨励金 I-A-2 の支援を受けた．

文 献

[1] H.B. Axelsen and R. Glück, “What do reversible programs compute?,” FoSSaCS, ed. by M. Hofmann, vol.6604, pp.42–56, LNCS, Springer-Verlag, 2011.

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

[3] R. Glück and T. Yokoyama, “A linear-time self-interpreter of a reversible imperative language,” Computer Software, vol.33, no.3, pp.108–128, 2016.

（平成 xx 年 xx 月 xx 日受付）

Abstract We show any programs in any reversible programming languages can be computed by reversible `R-WHILE` programs.

Key words Reversible programming language, reversible Turing machine, r-Turing completeness, R-WHILE