```
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   ::= X; C; write Y                               R-WHILE のプログラム
```

図 1: 言語 R-WHILE の構文規則

1 R-WHILE 言語での実装

この章では命令型可逆プログラミング言語 R-WHILE（以下，単に R-WHILE と呼ぶことにする) を万能可逆チューリング機械に変換する規則を与えることで，R-WHILE が万能可逆チューリング機械を実装可能であることを示す.

1.1 R-WHILE について

ここでは R-WHILE[文献] 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もっていることである．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．木構造のデータは同じ可逆プログラミング言語である Janus ももっているが，R-WHILE は他の木構造のデータを持っている言語に比べてとても単純な構造である．R-WHILE の構文規則は図 1である．プログラム P はただ一つの入り口と出口の点 (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 D はアトム nil とすべての組 (d_1, d_2) を含む（$d_1, d_2 \in D$）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 $d, e, f, ... \in D$ とする．また，X,Y,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 (先頭と素の残りを表す 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.

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．R-WHILE では可逆的代入 X ^= E を用いる．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 では左辺の変数 X に右辺の式 E があらわれてはならない．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．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

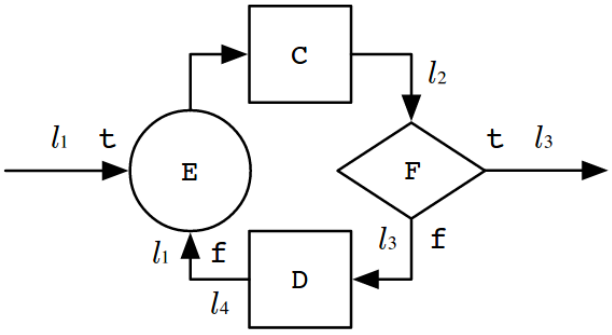

図 2: while loop のフローチャート

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 は図 2の様に描くことができる．ループを行うとき，アサーション E は真でなくてはならない（図 2の t）．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（図 2の f）．

R-WHILE にはプログラム逆変換機 $\mathcal{I}$ が定義されている (図 4)．それによって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．例えば，図 5はリストを反転させる R-WHILE のプログラム reverse と逆変換機 $\mathcal{I}$ によって求められた reverse のプログラム $\mathcal{I}\,[\![\text{reverse}]\!]$ である．$\mathcal{I}\,[\![\text{reverse}]\!]$ は X と Y の位置が入れ替わっていることを除き，reverse と同じプログラムの構造をしていることがわかる．

1.2 可逆チューリング機械から R-WHILE への変換

この章では実際に可逆チューリング機械を R-WHILE のプログラムで模倣する．
図 6a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．図 6aの main プログラム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 プログラム内の Q はチューリング機械の内部状態を表している．また，T はチューリング機械のテープの状態を表している．そして，組 (Q,T) はチューリング機械の様相を表している．プログラム内のループでは，チューリング機械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 を繰り返し実行する．

図 6b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．$\mathcal{T}[\![t]\!]^*$ は生成された書き換え規則の列である．

それぞれの書き換え規則は図 7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．可逆チューリング機械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．

図 6cMOVEL はヘッドを一つ左に動かすためのものである．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．

$$
\begin{array}{lcl}
\mathcal{E}[\![d]\!]\sigma & = & d \\
\mathcal{E}[\![\mathtt{X}]\!]\sigma & = & \sigma(\mathtt{X}) \\
\mathcal{E}[\![\mathtt{hd\ E}]\!]\sigma & = & \mathtt{e} \text{ if } \mathcal{E}[\![\mathtt{E}]\!]\sigma = \mathtt{(e.f)} \\
\mathcal{E}[\![\mathtt{tl\ E}]\!]\sigma & = & \mathtt{f} \text{ if } \mathcal{E}[\![\mathtt{E}]\!]\sigma = \mathtt{(e.f)}
\end{array}
\qquad
\begin{array}{lcl}
\mathcal{E}[\![\mathtt{cons\ E\ F}]\!]\sigma & = & (\mathcal{E}[\![\mathtt{E}]\!]\sigma, \mathcal{E}[\![\mathtt{F}]\!]\sigma) \\
\mathcal{E}[\![\mathtt{=?\ \ E\ F}]\!]\sigma & = & \begin{cases} \mathtt{true} & \text{if } \mathcal{E}[\![\mathtt{E}]\!]\sigma = \mathcal{E}[\![\mathtt{F}]\!]\sigma \\ \mathtt{false} & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{Q}[\![d]\!]\sigma & = & (d, \sigma) \\
\mathcal{Q}[\![\mathtt{X}]\!](\sigma \uplus \{\mathtt{X} \mapsto d\}) & = & (d, \sigma \uplus \{\mathtt{X} \mapsto \mathtt{nil}\}) \\
\mathcal{Q}[\![\mathtt{cons\ Q\ R}]\!]\sigma & = & ((d_1 \,.\, d_2)), \sigma_2 \text{ where } (d_1, \sigma_1) = \mathcal{Q}[\![\mathtt{Q}]\!]\sigma \ \wedge\ (d_2, \sigma_2) = \mathcal{Q}[\![\mathtt{R}]\!]\sigma_1
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{C}[\![\mathtt{X\ \hat{}= E}]\!](\sigma \uplus \{\mathtt{X} \mapsto d\}) & = & \sigma \uplus \{\mathtt{X} \mapsto d \odot \mathcal{E}[\![\mathtt{E}]\!]\sigma\} \\
\mathcal{C}[\![\mathtt{Q <= R}]\!] & = & \mathcal{Q}[\![\mathtt{Q}]\!]^{-1}(\mathcal{Q}[\![\mathtt{R}]\!]\sigma) \\
\mathcal{C}[\![\mathtt{C;\ D}]\!]\sigma & = & \mathcal{C}[\![\mathtt{D}]\!](\mathcal{C}[\![\mathtt{C}]\!]\sigma) \\
\mathcal{C}[\![\mathtt{if\ E\ then\ C\ else\ D\ fi\ F}]\!] & = & \begin{cases} \sigma' & \text{if } \mathcal{E}[\![\mathtt{E}]\!]\sigma = \mathtt{true} \ \wedge\ \sigma' = \mathcal{C}[\![\mathtt{C}]\!]\sigma \ \wedge\ \mathcal{E}[\![\mathtt{F}]\!]\sigma' = \mathtt{true} \\ \sigma' & \text{if } \mathcal{E}[\![\mathtt{E}]\!]\sigma = \mathtt{false} \ \wedge\ \sigma' = \mathcal{C}[\![\mathtt{D}]\!]\sigma \ \wedge\ \mathcal{E}[\![\mathtt{F}]\!]\sigma' = \mathtt{false} \end{cases} \\
\mathcal{C}[\![\mathtt{from\ E\ do\ C\ loop\ D\ until\ F}]\!] & = & \sigma' \quad \text{if} \mathcal{E}[\![\mathtt{E}]\!] = \mathtt{true} \wedge \sigma' = fix(F)(\sigma) \\
 & & \quad \text{where } F(\varphi) \ = \ \{(\sigma, \sigma_1) \mid \sigma_1 = \mathcal{C}[\![\mathtt{C}]\!]\sigma \ \wedge\ \mathcal{E}[\![\mathtt{F}]\!]\sigma_1 = \mathtt{true}\} \cup \\
 & & \qquad \{(\sigma, \sigma_3) \mid \sigma_1 = \mathcal{C}[\![\mathtt{C}]\!]\sigma \ \wedge\ \mathcal{E}[\![\mathtt{F}]\!]\sigma_1 = \mathtt{false} \wedge \\
 & & \qquad\quad \sigma_2 = \mathcal{C}[\![\mathtt{D}]\!]\sigma_1 \ \wedge\ \mathcal{E}[\![\mathtt{E}]\!]\sigma_2 = \mathtt{false} \wedge \\
 & & \qquad\quad \sigma_3 = \varphi(\sigma_2)\}
\end{array}
$$

$$
\mathcal{P}[\![\mathtt{P}]\!]D = D' \quad \text{if P is } \mathtt{read\ X;\ C;\ write\ Y} \ \wedge\ \mathcal{C}[\![\mathtt{C}]\!](\sigma^{\mathtt{P}}_{\mathtt{X}}(D)) = \sigma^{\mathtt{P}}_{\mathtt{Y}}(D')
$$

図 3: R-WHILE の表示的意味

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

図 4: R-WHILE の逆変換機 $\mathcal{I}$

```
read X;
 from (=? Y nil)
 loop (Z.X) <= X;
      Y<= (Z.Y)
 until(=? X nil);
write Y
```

(a) プログラム reverse

```
read Y;
 from (=? X nil)
 loop (Z.Y) <= Y;
      X <= (Z.X)
 until(=? Y nil);
write X
```

(b) プログラム $\mathcal{I}$[reverse]

図 5: 逆変換器 $\mathcal{I}$ の例

```
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 6: RTM を模倣する R-WHILE プログラム

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

図 7: 遷移規則から R-WHILE の書き換え規則への変換器 $\mathcal{T}$