卒　業　論　文

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

2018 年 01 月

南山大学 理工学部 ソフトウェア工学科

英語タイトル

2014SE006　AOKI Ryo
2014SE059　MASUDA Hiroki
2014SE089　SHIBATA Shintaro

Supervisor　YOKOYAMA Tetsuo

Month 2018

Department of Software Engineering
Faculty of Science and Engineering
Nanzan University

要約

可逆プログラミング言語 R-WHILE の計算モデルは，万能可逆チューリング機械と同じ計算能力があるとされている．しかし，筆者の知る限りにおいて，その具体的な証明についてはこれまで報告が無かった．本稿では任意の可逆チューリング機械を，意味が同じ R-WHILE プログラムに変換できることを示す．このことにより，可逆万能チューリング機械の R-WHILE プログラムが構成できることを示す．

Abstract

目次

第 1 章

序論

1.1 導入

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，意味論と構文規則がほかの可逆プログラミング言語より少ないため，単純な言語であるといえる．そのため，可逆なプログラムや，可逆なプログラムの振る舞いについての定理を証明する際に重宝される．しかし，R-WHILE の計算能力の高さは文献 [?] によって仮定されているが，筆者の知る範囲では R-WHILE で任意の可逆チューリング機械以下，RTM を構成できるほどの計算能力をもつという報告はない．そのため，R-WHILE は，RTM の知見を得ることが出来ていない．

1.2 背景

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，ほかのプログラミング言語との知見を共有する上で重要である．

1.3 期待される効果

我々は本研究において，R-WHILE で任意の RTM を模倣し，R-WHILE が可逆的計算万能性を持つことを証明する．これにより，R-WHILE は RTM や，可逆的計算万能性をもつ可逆プログラミング言語との知見を共有することができるようになる．

1.4 課題及び目的

増田大輝は第 1 章，第 2 章を，柴田心太郎は第 3 章と R-WHILE のプログラムを，青木峻は第 4 章，第 5 章を担当した．

第 2 章

関連研究

2.1 可逆計算

¡¡¡¡¡¡¡ HEAD 可逆計算とは，計算過程において，初期状態と最終状態を除いたすべての状態が，直前と直後にとり得る状態を高々一つもつものである．Landauer は，計算機において非可逆な演算はエネルギーの放出を伴うことを指摘した．可逆的な演算はこのような不可避なエネルギーの放出を減らす 1 つの解======= 可逆計算とは，計算過程において，どの状態においても直前と直後にとり得る状態を高々一つもつものである．*Landauer* は，計算機において非可逆な演算はエネルギーの放出を伴うことを指摘した．可逆的な演算はこのような不可避なエネルギーの放出を減らす 1 つの解¿¿¿¿¿¿¿ 2ee469a2632cb5f0908265060a46bf2c567eceef 決策として用いられる．可逆計算では，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 [5].

2.2 構造化定理

1 つの入り口と 1 つの出口を持つようなアルゴリズムには 3 つの基本構造がある．1 つは順次と呼ばれるプログラムに記されている順に随時処理を行う構造である．2 つ目は繰り返しと呼ばれ，ある条件が成立するならば処理 A を行い，そうでなければ処理 B を行なう．3 つ目は，一定の条件が満たされている間処理を繰り返す選択である．この 3 つの基本構造の組み合わせを利用してアルゴリズムを記述することができる [**?**].

2.3 可逆プログラミング言語

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の例を挙げると，*Janus*，*R* などが存在する．可逆プログラミング言語は，可逆であることの形式的証明されている．それらの言語は非可逆なプログラムを記述することができないため，可逆プログラミング言語で書かれた任意のプログラムの可逆性が保証される．

2.3.1 Janus

Janus とは，多重集合と配列の書換えに基づく制約処理モデルを持つ可逆プログラミング言語の一種．可逆プログラミング言語である *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．*Janus* の変数の基本型は `R-WHILE` とは異なり，整数型，整数型の配列とスタックである．そのため各データへのアクセスが容易である．また，*Janus* には複合代入演算子 $(+=, -=, !=)$ と交換演算子 $(:)$ が存在するため，変数の変更が可能である．その場合，複合代入演算子は右辺の式を評価し，演算子に応じて左側の変数を変更する演算子であり，交換演算子は左右の変数の値を交換する演算子である．この 2 つの演算子を利用する場合，単射な計算を行う記述であるために，右側の式や変数に，左側の変数が利用されてはならない．これ

は添字演算子式の場合でも同様に，その添字が右側の式や変数に利用されてはならない．

2.4 WHILE 言語

WHILE 言語とは，ローカル変数，整数，真偽式をもつ単純な命令型言語である．WHILE は構文規則や意味論が他のプログラム言語と比べ少なく，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．構造化定理の基本となる順次，分岐，繰り返しを表現することができ，分岐の場合 $if-else$ 文，繰り返しの場合 $while$ 文がある．その他には，変数に整数式を代入する割り当て，ユーザーからの入力を読み取って画面に出力する文の読み書き文，効果のないスキップ文がある．この言語には，0 n の値と，0 または 1 の結果のパラメータをを取ることができる手続きも含まれている [9].

2.5 セルオートマトン

セルオートマトンとは 1950 年代に von bNeumann(ジョン・フォン・ノイマン)　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によって決定される．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．標準的な CA は

$$(\mathbb{Z}^k, Q, N, f, \#)$$

として定める．ただし，Q はセルの状態と呼ばれる空でない有限集合，$N(=\{n_1,\ldots,n_m\})$ は近傍と呼ばれる $\mathbb{Z}^k$ の部分集合，局所関数 $f : Q^m \to Q$ は各セルの状態遷移を定めるものとする．なお，$\# \in Q$ は静止状態と呼ばれ，$f(\#,\ldots,\#) = \#$ を満たす．集合 Q 上の k 次元の状相とは $\alpha : \mathbb{Z}^k \to Q$ であるような写像 α をいう．Q 上の k 次元状相すべての集合を $\mathrm{Conf}_k(Q) = \{\alpha \mid \alpha : \mathbb{Z}^k \to Q\}$ で表す．状相 α は，集合 $\{x \mid x \in \mathbb{Z}^k \wedge \alpha(x) \neq \#\}$ が有限であるとき，有限と呼ばれる．

大域関数 $F : \mathrm{Conf}_k(Q) \to \mathrm{Conf}_k(Q)$ を

$$\begin{aligned} &\forall \alpha \in \mathrm{Conf}_k(Q), \forall x \in \mathbb{Z}^k : \\ &F(\alpha)(x) = f(\alpha(x+n_1),\ldots,\alpha(x+n_m)) \end{aligned} \tag{2.1}$$

と定める [5].

可逆セルオートマトン

可逆セルオートマトン (以下 RCA) とは，可逆性が保証されたセルオートマトン (以下 CA)，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 CA のことである．RCA は，以下の条件を満たしている．

1. 大域関数 F が単射であること．
2. 状相の遷移がちょうど逆であるような CA が存在すること．

1 次元，2 次元 RCA は計算可逆万能性をもつモデルが数多く発見されている．Toffoli は RCA の次元を 1 増やすことで，2 次元以上の RCA において任意の TM が模倣できることを示し，そこから分割 CA の枠組みを利用して，1 次元以上の RCA が計算可逆万能性を持つことを証明した [5].

2.5.1 構造的帰納法

構造化帰納法とは関数の中からその関数自身を呼び出すような処理を行う再帰関数を数学的帰納法によって正当性を証明する手法である．リスト構造や木など，再帰的なデータ構造に関する命題を証明する際に用いら

れる．すべての構造 S について命題 $P(S)$ が成り立つことを証明する場合方法を説明する．

1. 極小元 μ について命題 $P(\mu)$ が成り立つことを示す．
2. S を任意の構造とし，S より小さなすべての構造 S ’を仮定して，$P(S)$ を示す．

1,2 を示すことによって，すべての構造 S について命題 $P(S)$ が成り立つことが証明される．
今回私たちが証明する計算万能性は URTM 変換前の様相を a，変換後の様相を b とすると，任意の a を R-WHILE で実行できるようにした関数 a ’が R-WHILE での変換後の関数が b と意味的に等しい b ’であることを証明することにより，R-WHILE の遷移規則が URTM の遷移規則と意味的に等しいことを証明する．

2.5.2 万能性

第 3 章

可逆チューリング機械

この章ではチューリング機械とチューリング機械に制約を加えた可逆チューリング機械の定義，及び具体例を記述する．

3.1 チューリング機械

あるシステムにおいて，実行すべきことに曖昧さがなく明確に記述されていて，記述されたことを機械的に実行すれば出力が得られるものとする．このときそのように記述されたものを機械的な手順，あるいはアルゴリズムと呼ぶ [3]．また，アルゴリズムの特徴として実効性，有限性がある．実効性は，その手順を実際に行うことのできる性質のことであり，有限性は，実行する時間が有限であることを示す．この機械的な手順は直感的な概念であるが，Turing は，この概念を厳密に定義することを試み，1936 年にチューリング機械と呼ばれる理論的計算モデルを導入した．チューリング機械は現在のコンピュータをもシミュレートできるとされている．そのため，任意のチューリング機械をシミュレートできる計算システムは計算万能性をもつといわれる．このとき，計算万能性とは計算可能な問題であれば全て計算可能であることを指す. あるシステムから特徴的な要素を抽出してモデル化し，それとほぼ同じ条件や規則に従った別のシステムで模擬することをシミュレーションという．

また，1943 年に Kleen によって，機械的な手順で計算できるということはチューリング機械の枠組みの中で定式化できるという提唱 (チャーチ・チューリングのテーゼ) がなされた [1].

3.1.1 チューリング機械の振る舞い (動作)

チューリング機械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 3.1]．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 3.1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．

3.1.2 定義

本稿では，1 テープチューリング機械を $T = (Q, \Sigma, b, \delta, q_s, Q_f)$ として定める．(以下，1 テープチューリング機械のことをチューリング機械と呼ぶ．) 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

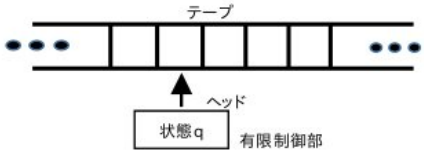

図 3.1: チューリング機械の概要

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 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]，5 項組で設計を行った場合，ヘッダ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 項組と比べてヘッダ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が，遷移規則に対称性があるため，以降の小節で説明する可逆チューリング機械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と判断したため今回は 3 項組を用いる．

ここで，チューリング機械が計算を行っているある時点での様子を様相と呼ぶ．このとき，チューリング機械 $T=(Q,\Sigma,b,\delta,q_s,q_f)$ 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．

チューリング機械 $T=(Q,\Sigma,b,\delta,q_s,q_f)$ 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

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

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

チューリング機械 $T=(Q,\Sigma,b,\delta,q_s,q_f)$ 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．

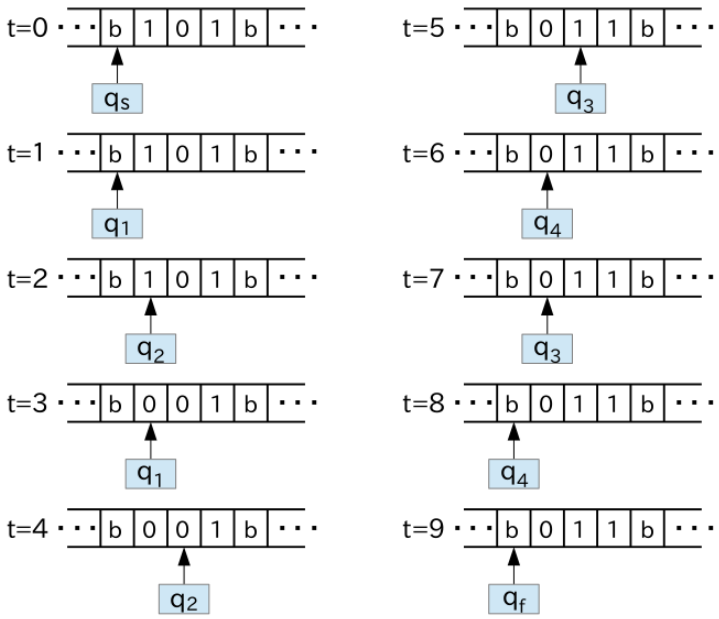

図 3.2: チューリング機械の動作例

3.1.3 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．各状態における T_1 の動作を以下に説明する．また実際の動作の例は [図 3.2] である．このとき，あらかじめ格納されている 2 進数を「101」とする．t は計算の進行 (遷移) を表す．

q_s：2 進数表現 n の左隣の b を読み，q_1 へ．

q_1：ヘッドを右に 1 コマ移動し，q_2 へ.
q_2：1 を読んだとき，0 に書き換え q_1 へ．b または 0 を読んだとき 1 に書き換え，q_3 へ.
q_3：ヘッドを左に 1 コマ移動し，q_4 へ.
q_4：0 または 1 を読んだ場合，q_3 へ，b を読んだ場合 q_f へ.

また，この遷移規則による様相を 3.1.2 節の記法を用いて表わす.

$$
\begin{aligned}
t=0 \qquad & (q_s,(\lambda,b,1)) && \underset{T_1}{\Rightarrow} && (q_1,(\lambda,b,1))\\
t=1 \qquad & && \underset{T_1}{\Rightarrow} && (q_2,(b,1,0))\\
t=2 \qquad & && \underset{T_1}{\Rightarrow} && (q_1,(b,0,0))\\
t=3 \qquad & && \underset{T_1}{\Rightarrow} && (q_2,(0,0,1))\\
t=4 \qquad & && \underset{T_1}{\Rightarrow} && (q_3,(0,1,1))\\
t=5 \qquad & && \underset{T_1}{\Rightarrow} && (q_4,(b,0,1))\\
t=6 \qquad & && \underset{T_1}{\Rightarrow} && (q_3,(b,0,1))\\
t=7 \qquad & && \underset{T_1}{\Rightarrow} && (q_4,(\lambda,b,0))\\
t=8 \qquad & && \underset{T_1}{\Rightarrow} && (q_3,(\lambda,b,0))\\
t=9 \qquad & && \underset{T_1}{\Rightarrow} && (q_f,(\lambda,b,0))
\end{aligned}
$$

3.1.4 n テープチューリング機械

先程までは, 1 テープのみを考えたが, テープとヘッドがそれぞれ n 個ある n テープチューリング機械も存在する．n テープチューリング機械 T' は $T'=(Q,(\Sigma_1,\Sigma_2,...,\Sigma_n),b,\delta,q_s,Q_f)$ として定める．Q,b,q_s,Q_f は 1 テープチューリング機械と同様であり，Σ_1,Σ_2,...,Σ_n はそれぞれ n 本目のテープで利用されるテープ記号の空でない有限集合, δ は $((Q\backslash q_f)\times[[\Sigma_1\times\Sigma_2\times...\times\Sigma_n]\times[\Sigma_1\times\Sigma_2\times...\times\Sigma_n]]\times Q)\cup((Q\backslash q_f)\times[d_1\times d_2\times...\times d_n]\times Q)$ の部分集合である．ただし，d はヘッド移動の表す矢印 $[\leftarrow,\downarrow,\rightarrow]$ である．

また，任意の n テープチューリング機械に対して，それと等価な 1 テープチューリング機械が存在する [3].

3.2 可逆チューリング機械

可逆チューリング機械とは，内部状態とヘッドの先にある記号に対して，直後の動作が唯一である場合，遷移直後の内部状態とヘッドの記号が決まれば，どのような動作をしたかが決定できる性質をもつチューリング機械である．そのため，時間軸の逆方向に決定的なチューリング機械であると言える.

3.2.1 定義

チューリング機械 $T=(Q,\Sigma,b,\delta,q_s,Q_f)$ において，任意の異なる遷移規則 $m_1=[p,\langle s,t\rangle,q]$，$m_2=[p',\langle s',t'\rangle,q']\in\delta$ に対して， $p=p'$ ならば,$s\neq s'$ を満たすとき T を局所的に前方決定的または，決定性チューリング機械と呼ぶ．また，T の任意の異なる遷移規則 $m_1=[p,\langle s,t\rangle,q]$，$m_2=[p',\langle s',t'\rangle,q']\in\delta$ に対して $q=q'$ ならば,$t\neq t'$ を満たすとき T を局所的に後方決定的または，可逆チューリング機械と呼ぶ.

以下，可逆チューリング機械とは決定性チューリング機械の条件と可逆チューリング機械の条件をもち最終

状態からの遷移及び，初期状態への遷移がないものとする．

3.2.2 可逆チューリング機械の例

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

例　与えられた 2 進数の 0 と 1 を反転させる可逆チューリング機械　$T_2 = (Q_2, 0, 1, b, \delta_2, q_s, q_f)$ を考える．ただし，$Q_2 = \{q_s, q_1, q_2, q_3, q_4, q_f\}$ であり，δ_2 は以下の 3 項組の集合である．

$\delta_2 = \{[q_s, \langle b, b\rangle, q_1], [q_1, \rightarrow, q_2],$
$[q_2, \langle 0, 0\rangle, q_1],$
$[q_2, \langle 1, 1\rangle, q_1],$
$[q_2, \langle b, b\rangle, q_3], [q_3, \leftarrow, q_4],$
$[q_4, \langle 0, 1\rangle, q_3],$
$[q_4, \langle 1, 0\rangle, q_3],$
$[q_4, \langle b, b\rangle, q_f]\}$

T_2 は，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，書かれている 2 進数表現を反転したものに書き換え，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．このとき，T_2 は 2 進数表現の 1 つ右隣のます目までヘッド移動する．その後，ヘッドを初期の位置に移動させながら，1 を読み込んだとき 0 に書き換える．また，0 を読み込んだとき 1 に書き換える．各状態における T_2 の動作を以下で説明する．また実際の動作の例は…である．

q_s：2 進数表現 n の左隣の b を読み，q_1 へ．
q_1：ヘッドを右に 1 コマ移動し，q_2 へ．
q_2：b 以外を読んだとき，q_1 へ．また，b を読んだとき，q_3 へ．
q_3：ヘッドを左に 1 コマ移動し，q_4 へ．
q_4：0 を読んだとき，1 に書き換える．また，1 を読み込んだ場合 0 に書き換え，q_3 へ．そして，b を読んだとき
　q_f へ．

T_2 において，q_1 が 1 番目の項として現れている 3 項組は (i) $[q_2, \langle 0, 0\rangle, q_1]$,(ii) $[q_2, \langle 1, 1\rangle, q_1]$ または (iii) $[q_2, \langle b, b\rangle, q_3]$ の 3 つである．ある時刻において T_1 が状態 q_1 であり，現在読んでいるます目が 0 ならば (i)，1 ならば (ii) または b ならば (iii) が直後に実行されるということが一意に決まる．これと同様のことは状態 q_4 でも言えるため，T_2 は局所的に前方決定的である．また，q_1 が 3 番目の項として現れている 3 項組は (iv) $[q_s, \langle b, b\rangle, q_1]$，(v) $[q_2, \langle 0, 0\rangle, q_1]$ または (vi) $[q_2, \langle 0, 0\rangle, q_1]$ の 3 つである. ある時刻において T_1 が状態 q_1 であり，現在読んでいるます目が b ならば (iv)，0 ならば (v) または 1 ならば (vi) が直後に実行されるということが一意に決まる．これと同様のことは状態 q_3 でも言えるため，T_2 は局所的に後方決定的である．そして T_1 は最終状態からの遷移および初期状態への遷移はない．以上より T_2 は可逆チューリング機械である．

3.2.3 可逆シミュレーション

様々な理由により実物による可逆化が困難または不可能な場合に，実物から特徴的な要素を抽出してモデル化し，それとほぼ同じ条件や規則に従った別のシステムで模擬することを可逆シミュレーションという．

3.2.4 非可逆のチューリング機械の可逆にする例

ここでは非可逆なチューリング機械の遷移規則を工夫することで，可逆チューリング機械に作り変えること (可逆シミュレート) が出来るということを実際に例を用いて説明する．例には 3.3 節で説明した T_1 を用いる．T_1 の遷移規則 δ_1 は以下の 3 項組の集合であった．

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

T_1 は次の理由により非可逆なチューリング機械である．q_3 が 3 番目の項として現れている 3 項組は (i) $[q_2, \langle 0, 1\rangle, q_3]$，(ii) $[q_2, \langle b, 1\rangle, q_3]$，(iii) $[q_4, \langle 0, 0\rangle, q_3]$ または (iv) $[q_4, \langle 1, 1\rangle, q_3]$ の 4 つである．そのうち (i),(ii) または (iv) の 3 つの遷移規則は書き換えた後の記号がどれも 1 である．つまり，ある時刻において T_1 が状態 q_3 にあり，現在読んでいるます目が 1 ならば，(i),(ii) または (iv) のどれが直前に実行されたかが一意に決まらない．すなわち，可逆チューリング機械の条件である局所的に後方決定的であるという条件を満たさないため，T_1は 非可逆なチューリング機械である．

この非可逆なチューリング機械 T_1 の遷移規則を可逆になるように設計した可逆チューリング機械 $T_{1-2} = (Q_3, (b, 0, 1), b, \delta_1, q_s, q_f)$ とする. ただし，$Q_3 = \{q_s, q_1, q_2, q_3, q_4, q_5, q_6 . q_7, q_8, q_9, q_{10}, q_f\}$ であこのとき，遷移規則 δ_{1-2} 以下のように表す.

$$\begin{aligned}\delta_{1-2} = \{&[q_s, \langle b, b\rangle, q_1], [q_1, \rightarrow, q_2],\\ &[q_2, \langle 1, 0\rangle, q_1],\\ &[q_2, \langle 0, 1\rangle, q_3], [q_3, \rightarrow, q_5],\\ &[q_5, \langle 0, 0\rangle, q_7],\\ &[q_5, \langle 1, 1\rangle, q_7],\\ &[q_2, \langle b, 1\rangle, q_4], [q_4, \rightarrow, q_6],\\ &[q_6, \langle b, b\rangle, q_7], [q_7, \leftarrow, q_8],\\ &[q_8, \langle 1, 1\rangle, q_9], [q_9, \leftarrow, q_{10}],\\ &[q_{10}, \langle 0, 0\rangle, q_9],\\ &[q_{10}, \langle b, b\rangle, q_f]\}\end{aligned}$$

T_1 は 1 を加えた直後の状態から直前に 1 を加えることで 2 進数表現に桁上げが行われたのかを判断することが出来なかった．そこで T_{1-2} では 1 を加える動作の後にヘッドを 2 進数表現の 1 つ左まで移動するのではなく，さらに右に 1 つヘッドを移動し、そのます目に書かれる記号が空白記号であるかそうでないかを確認する動作を追加した．そうすることで 2 進数表現に 1 が加えられたとき，桁上げが行われていたかを判断すること

が出きるようになる.
章○では単純な可逆チューリング機械について説明したが，この例の様に非可逆なチューリング機械は単純に設計されていても，可逆なものに再度設計 (可逆シミュレーション) することで単純であった遷移規則が複雑になってしまうことがある.

また，この遷移規則による様相を 3.1 の記法を用いて表わす.

$$
\begin{array}{lll}
t=0 & (q_s,(\lambda,b,1)) & \underset{T_{1-2}}{\Longrightarrow} \quad (q_1,(\lambda,b,1)) \\
t=1 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_2,(b,1,0)) \\
t=2 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_1,(b,0,0)) \\
t=3 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_2,(0,0,1)) \\
t=4 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_3,(0,1,1)) \\
t=5 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_5,(1,1,b)) \\
t=6 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_7,(1,1,b)) \\
t=7 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_8,(0,1,1)) \\
t=8 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_9,(0,1,1)) \\
t=9 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_{10},(b,0,1)) \\
t=10 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_9,(b,0,1)) \\
t=11 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_{10},(\lambda,b,0)) \\
t=12 & & \underset{T_{1-2}}{\Longrightarrow} \quad (q_f,(\lambda,b,0))
\end{array}
$$

3.2.5 3 テープ可逆チューリング機械

2.4 節では，1 テープの非可逆チューリング機械を 1 テープの可逆チューリング機械にする可逆シミュレーションを行ったが，本節では非可逆なチューリング機械を可逆なチューリング機械を構成するために，3 本のテープを持つチューリング機械を用いる．3 テープ可逆チューリング機械　T'' を 1.4 節の定義より $T''=(Q,(\Sigma_1,\Sigma_2,\Sigma_3),b,\delta,q_s,q_f)$ と定める．T'' の 1 本目のテープは T のテープをそのまま表現し，ヘッドの位置も同じである．　2 本目のテープは各ステップで T のどの 3 項組が実行されたのかという記述を記録する．3 本目のテープは 1 本目のテープの記述をコピーし，可逆化によって初期状態に戻る 1 本目のテープの代わりに，結果を記録する．Q,b,q_s,q_f は 1 テープチューリング機械と同様であり，$\Sigma_1,\Sigma_2,\Sigma_3$ は 3 本のテープそれぞれで使用される記号の有限集合である.

δ は 3 項組集合であるが，各項は,$[p,\langle s,t\rangle,q]$ または $[p,d,q]$ の形であった 1 テープのチューリング機械の遷移規則である s,t,d がそれぞれ 3 テープ分必要になるため，3 テープチューリング機械では $[p,\langle[s_1,s_2,s_3],[t_1,t_2,t_3]\rangle,q]$ または $[p,[d_1,d_2,d_3],q]$ の形をしている．そのため, 前者の 3 項組は T'' が状態 p で，3 つのヘッドの記号 s_1,s_2,s_3 を読んだ場合，記号 t_1,t_2,t_3 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_1,d_2,d_3 の方向に動かし，状態 q にすることを意味する.

可逆性の定義も 1 テープの場合と同様である．すなわち任意の異なる 2 つの 3 項組 $n_1=[p,\langle[s_1,s_2,s_3],[t_1,t_2,t_3]\rangle,q]$,$n_2=[p',\langle[s'_1,s'_2,s'_3],[t'_1,t'_2,t'_3]\rangle,q']$ に対して $p'=q'$ ならば $[t_1,t_2,t_3]\neq[t'_1,t'_2,t'_3]$ という関係が成り立つ.

可逆化 (Bennett)

まず計算における可逆であることの利点を述べる．可逆計算において単純でわかりやすいことから，計算に要するエネルギーを減らすことが可能である点である．出力結果から入力の値を求めることができる性質を利用し，不必要な情報 (以下ゴミ情報と呼ぶ) を消去することで，計算に必要なエネルギーを最小にすることが出来る．3 テープ可逆チューリング機械の動作の過程は，大きく以下の 3 つに分割される．

1.「2 本目のテープに動作履歴を残しながら計算を実行する過程」
2.「計算結果を 3 本目のテープにコピーする過程」
3.「動作履歴を可逆的に消去しながら逆の動作を実行する過程」

このようにして動作を実行することで，最終的に 1 本目のテープに入力が残り，3 本目のテープ に答えが出力され，ゴミ情報は綺麗に消去された状態で停止することが出来る．

Bennett の可逆化の例

3-2-2 の記法を用いて，3-1 章の具体例で扱った 1 テープ非可逆チューリング機械を 3 テープ可逆チューリング機械でシミュレートする．

チューリング機械 T_1 をシミュレートする可逆チューリング機械 T_3 は，

T_3=($\{q_s, q_1, q_2, q_3, q_4, q_f, c_b, c_1, c_2, c_3, c_4, c_5, p_s, p_1, p_2, p_3, p_4, p_f\}, \{b, 0, 1\}, \{0, 1, 2, 3, 4, 5, 6, 7\}, \{b, 0, 1\}, 0, \delta_3, q_s, \{p_s\}$)

遷移規則 δ_3 は，

δ_3={ $[q_s, \langle[b, b, b], [b, 1, b]\rangle, q_1]$, $[q_1, [\rightarrow, \rightarrow, \downarrow], q_2]$,
$[q_2, \langle[1, b, b], [0, 2, b]\rangle, q_1]$, $[q_3, [\leftarrow, \rightarrow, \downarrow], q_4]$,
$[q_2, \langle[0, b, b], [1, 3, b]\rangle, q_3]$,
$[q_2, \langle[b, b, b], [1, 4, b]\rangle, q_3]$,
$[q_4, \langle[0, b, b], [0, 5, b]\rangle, q_3]$,
$[q_4, \langle[1, b, b], [1, 6, b]\rangle, q_3]$,
$[q_4, \langle[b, b, b], [b, 7, b]\rangle, q_f]$, $[q_f, [\downarrow, \rightarrow, \downarrow], c_0]$,
$[c_0, \langle[b, b, b], [b, b, b]\rangle, c_1]$, $[c_1, [\rightarrow, \downarrow, \rightarrow], c_2]$,
$[c_2, \langle[1, b, b], [1, b, 1]\rangle, c_1]$,
$[c_2, \langle[0, b, b], [0, b, 0]\rangle, c_1]$,
$[c_2, \langle[b, b, b], [b, b, b]\rangle, c_3]$,$[c_3, [\leftarrow, \downarrow, \leftarrow], c_4]$,
$[c_4, \langle[1, b, 1], [1, b, 1]\rangle, c_3]$,
$[c_4, \langle[0, b, 0], [0, b, 0]\rangle, c_3]$,
$[c_4, \langle[b, b, b], [b, b, b]\rangle, c_5]$,$[c_5, [\downarrow, \leftarrow, \downarrow], p_f]$,
$[p_f, \langle[b, 7, b], [b, b, b]\rangle, p_4]$,
$[p_3, \langle[1, 6, b], [1, b, b]\rangle, p_4]$,
$[p_3, \langle[0, 5, b], [0, b, b]\rangle, p_4]$,
$[p_3, \langle[1, 4, b], [b, b, b]\rangle, p_2]$,
$[p_3, \langle[1, 3, b], [0, b, b]\rangle, p_2]$,
$[p_1, \langle[0, 2, b], [1, b, b]\rangle, p_2]$, $[p_4, [\rightarrow, \leftarrow, \downarrow], p_3]$,
$[p_1, \langle[b, 1, b], [b, b, b]\rangle, p_s]$, $[p_2, [\leftarrow, \leftarrow, \downarrow], p_1]$ }

以下は上で述べた 3 テープ可逆チューリング機械の動作に従いながら解説する．まず 1 つ目の過程であるが，δ_3 の最初の 9 個の 3 項組は δ_2 の 9 個の 3 項組に対応している．これらは 1 本目のテープ上で T_2 の動作を以前と同様にシミュレートするだけでなく，2 本目のテープに δ_2 の $1-7$ のどの 3 項組が使われかを示す $1-7$ の数字を書き込むことで，可逆チューリング機械の条件を満たしながら計算を実行することが出来る.2

つ目の過程では，T_3 は，T_2 の最終状態である q_f に遷移したとき，c_s に移し，その後 $c_i(i = 0, 1, ..., 5)$ を使って 1 本目のテープに記された答えを 3 本目のテープにコピーする．コピー後にゴミ情報を消去するため状態を p_f に遷移する．3 つ目の過程では，δ_3 の最後の 9 個の 3 項組は最初の 9 個の 3 項組の逆から，つまり q_f から q_s までの動作を行う．そのため p と q はそれぞれ対応していて，最終的には状態 p_s で停止する．

また，この遷移規則による様相を 3.1 の記法を用いて表わす.

$$
\begin{array}{lll}
t=0 & (q_s,((\lambda,b,1),(\lambda,b,\lambda),(\lambda,b,\lambda))) & \underset{T_3}{\Rightarrow} (q_1,((\lambda,b,1),(\lambda,1,\lambda),(\lambda,b,\lambda)))\\
t=1 & & \underset{T_3}{\Rightarrow} (q_2,((b,1,0),(1,b,\lambda),(\lambda,b,\lambda)))\\
t=2 & & \underset{T_3}{\Rightarrow} (q_1,((b,0,0),(1,2,\lambda),(\lambda,b,\lambda)))\\
t=3 & & \underset{T_3}{\Rightarrow} (q_2,((0,0,1),(2,b,\lambda),(\lambda,b,\lambda)))\\
t=4 & & \underset{T_3}{\Rightarrow} (q_3,((0,1,1),(2,3,\lambda),(\lambda,b,\lambda)))\\
t=5 & & \underset{T_3}{\Rightarrow} (q_4,((b,0,1),(3,b,\lambda),(\lambda,b,\lambda)))\\
t=6 & & \underset{T_3}{\Rightarrow} (q_3,((b,0,1),(3,5,\lambda),(\lambda,b,\lambda)))\\
t=7 & & \underset{T_3}{\Rightarrow} (q_4,((\lambda,b,0),(5,b,\lambda),(\lambda,b,\lambda)))\\
t=8 & & \underset{T_3}{\Rightarrow} (q_f,((\lambda,b,0),(5,7,\lambda),(\lambda,b,\lambda)))\\
t=9 & & \underset{T_3}{\Rightarrow} (c_0,((\lambda,b,0),(7,b,\lambda),(\lambda,b,\lambda)))\\
t=10 & & \underset{T_3}{\Rightarrow} (c_1,((\lambda,b,0),(7,b,\lambda),(\lambda,b,\lambda)))\\
t=11 & & \underset{T_3}{\Rightarrow} (c_2,((b,0,1),(7,b,\lambda),(b,b,\lambda)))\\
t=12 & & \underset{T_3}{\Rightarrow} (c_1,((b,0,1),(7,b,\lambda),(b,0,\lambda)))\\
t=13 & & \underset{T_3}{\Rightarrow} (c_2,((0,0,1),(7,b,\lambda),(0,b,\lambda)))\\
t=14 & & \underset{T_3}{\Rightarrow} (c_1,((0,1,1),(7,b,\lambda),(0,1,\lambda)))\\
t=15 & & \underset{T_3}{\Rightarrow} (c_2,((1,1,b),(7,b,\lambda),(1,b,\lambda)))\\
t=16 & & \underset{T_3}{\Rightarrow} (c_1,((1,1,b),(7,b,\lambda),(1,1,\lambda)))\\
t=17 & & \underset{T_3}{\Rightarrow} (c_2,((1,b,\lambda),(7,b,\lambda),(1,b,\lambda)))\\
t=18 & & \underset{T_3}{\Rightarrow} (c_3,((1,b,\lambda),(7,b,\lambda),(1,b,\lambda)))\\
t=19 & & \underset{T_3}{\Rightarrow} (c_4,((1,1,b),(7,b,\lambda),(1,1,b)))\\
t=20 & & \underset{T_3}{\Rightarrow} (c_3,((1,1,b),(7,b,\lambda),(1,1,b)))\\
t=21 & & \underset{T_3}{\Rightarrow} (c_4,((0,1,1),(7,b,\lambda),(0,1,1)))\\
t=21 & & \underset{T_3}{\Rightarrow} (c_3,((0,1,1),(7,b,\lambda),(0,1,1)))\\
t=22 & & \underset{T_3}{\Rightarrow} (c_4,((b,0,1),(7,b,\lambda),(b,0,1)))\\
t=23 & & \underset{T_3}{\Rightarrow} (c_3,((b,0,1),(7,b,\lambda),(b,0,1)))\\
t=24 & & \underset{T_3}{\Rightarrow} (c_4,((\lambda,b,0),(7,b,\lambda),(\lambda,b,0)))\\
t=25 & & \underset{T_3}{\Rightarrow} (c_5,((\lambda,b,0),(7,b,\lambda),(\lambda,b,0)))\\
t=26 & & \underset{T_3}{\Rightarrow} (p_f,((\lambda,b,0),(5,7,b),(\lambda,b,0)))\\
t=27 & & \underset{T_3}{\Rightarrow} (p_4,((\lambda,b,0),(5,b,\lambda),(\lambda,b,0)))\\
t=28 & & \underset{T_3}{\Rightarrow} (p_3,((b,0,1),(3,5,b),(\lambda,b,0)))\\
t=29 & & \underset{T_3}{\Rightarrow} (p_4,((b,0,1),(3,b,\lambda),(\lambda,b,0)))\\
t=30 & & \underset{T_3}{\Rightarrow} (p_3,((0,1,1),(2,3,b),(\lambda,b,0)))\\
t=31 & & \underset{T_3}{\Rightarrow} (p_2,((0,0,1),(2,b,\lambda),(\lambda,b,0)))\\
t=32 & & \underset{T_3}{\Rightarrow} (p_1,((b,0,0),(1,2,b),(\lambda,b,0)))\\
t=33 & & \underset{T_3}{\Rightarrow} (p_2,((b,1,0),(1,b,\lambda),(\lambda,b,0)))\\
t=34 & & \underset{T_3}{\Rightarrow} (p_1,((\lambda,b,1),(\lambda,1,\lambda),(\lambda,b,0)))\\
t=35 & & \underset{T_3}{\Rightarrow} (p_s,((\lambda,b,1),(\lambda,b,\lambda),(\lambda,b,0)))
\end{array}
$$

この方法を一般化することにより，次のことが言える．任意の 1 テープチューリング機械 T_a に対し，それをシミュレートする 3 テープの可逆チューリング機械 T'_a では，T_a に記号列 x を与えたとき，記号列 y を書き出

して最終状態で停止するならば，T'_a に x を与えたとき，ゴミ情報である動作履歴を残すことなく x と y だけを書き出し，最終状態で停止することができる．

第 4 章

R-WHILE でも実装

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.

4.1 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，既存の可逆命令言語 Janus では難しいことである.

4.1.1 プログラム例：リスト逆転

与えられたリストを逆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 X を逆転させるプログラム reverse*)
  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．可逆的繰り返しを行う命令，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 をペアにした値を構成している．可逆的置き換えは右辺の元の値を左辺の変数にバインドする前に，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．プログラム reverse の適用は $[\![\text{reverse}]\!]^{\text{R-WHILE}}$ ('a.('b.('c.nil)))=('c.('b('a.nil))) の様に示す．ここで，リストの要素'a,'b **または**'c はアトムである．エントリーアサーションと可逆的置き換えにより，プログラムは可逆である.

```
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 4.1: 言語 R-WHILE の構文規則

4.2 構文

R-WHILE の構文規則は単純である（図 4.1）．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 (d_1, d_2) を含む（$d_1, d_2 \in \mathbb{D}$）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 $d, e, f, ... \in \mathbb{D}$ 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．

可逆的代入 X ^= E において，左辺の変数 X に右辺の式 E があらわれてはならない．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．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 は図??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??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??の f）．

私たちは $(d_1.(d_2.(\bullet\bullet\bullet.(d_n.\mathrm{nil})\bullet\bullet\bullet)))$ をリスト $(d_1 d_2 \bullet\bullet\bullet d_n)$ と書き, $(d_1.(d_2.(\bullet\bullet\bullet.(d_{n-1}.d_n)\bullet\bullet\bullet)))$ を不適切なリスト $(d_1 \bullet\bullet\bullet d_{n-1} d_n)$ と書く．

リストと似た表記として，私たちは $(Q_1.(Q_2.(\bullet\bullet\bullet.(Q_n.\mathrm{nil})\bullet\bullet\bullet)))$ をリスト $(Q_1 Q_2 \bullet\bullet\bullet Q_n)$ と書き，$(Q_1.(Q_2.(\bullet\bullet\bullet.(Q_{n-1}.Q_n)\bullet\bullet\bullet)))$ を不適切なリスト $(Q_1 \bullet\bullet\bullet Q_{n-1} Q_n)$ と書く．

式 $(\mathrm{list}\ E_1 \bullet\bullet\bullet E_n)$ は式 $(\mathrm{cons}\ E_1 \bullet\bullet\bullet (\mathrm{cons}\ E_n \mathrm{nil})\bullet\bullet\bullet)$ の省略法として用いられる．また，式 $(\mathrm{cons}{*}E_1 \bullet\bullet\bullet E_{n-1} E_n)$ は式 $(\mathrm{cons}\ E_1 \bullet\bullet\bullet (\mathrm{cons}\ E_{n-1} E_n)\bullet\bullet\bullet)$ として用いられる．

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

$$
\begin{array}{lcl}
\mathcal{Q}[\![\mathtt{d}]\!]\sigma & = & (d, \sigma) \\
\mathcal{Q}[\![\mathtt{X}]\!](\sigma \uplus \{\mathtt{X} \mapsto \mathtt{d}\}) & = & (\mathtt{d}, \sigma \uplus \{\mathtt{X} \mapsto \mathtt{nil}\}) \\
\mathcal{Q}[\![\mathtt{cons\ Q\ R}]\!]\sigma & = & ((\mathtt{d}_1.\mathtt{d}_2)), \sigma_2 \text{ where } (\mathtt{d}_1, \sigma_1) = \mathcal{Q}[\![\mathtt{Q}]\!]\sigma \ \wedge\ (\mathtt{d}_2, \sigma_2) = \mathcal{Q}[\![\mathtt{R}]\!]\sigma_1
\end{array}
$$

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

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

図 4.2: R-WHILE の表示的意味

4.3 意味論

ここでは R-WHILE のための表示的意味論を定義する．まず，いくつかの表記法から始め，図 4.2の４つの意味関数について説明する．

プログラム P の中にある変数の組と式 E はそれぞれ $Var(\mathtt{P})$, $Var(\mathtt{E})$ と示される．ブール値は明確な二つの要素（$\mathtt{false,true} \in \mathbb{D}$) によって示される．未定義の値は $\perp$ によって示される．そして，組 X または $\{\perp\}$ は $X_\perp$ によって示される．私たちは値の範囲として $\mathbb{D}_\perp$ を用いる．

プログラム P のための状態 σ は $Var(\mathtt{P})$ から $\mathbb{D}_\perp$ への関数である．P のためのすべての状態の組は $\mathtt{Stores}^{\mathtt{P}}$ によって示される．状態 $\sigma\backslash\mathtt{X}$ は変数 X を $\perp$ に写すことを除けば σ と同一である．状態 $\sigma \uplus \sigma'$ は状態 σ と σ' の縛るものの非交和であり，それらは共通の変数をもたない．未定義の状態すなわち，どのような変数の対しても $\perp$ をかえすものは $\perp$ とも書かれる．

状態 $\sigma^{\mathtt{P}}_{\mathtt{X}}$ は変数 X を D に縛る．また，$Var(\mathtt{P})$=$\{\mathtt{X},\mathtt{Y}_1,\ldots,\mathtt{Y}_n\}$ 内のの他すべての変数を nil に縛る．すなわち，

$\sigma^{\mathtt{P}}_{\mathtt{X}}(D) = [\mathtt{X} \mapsto D, \mathtt{Y}_1 \mapsto \mathtt{nil}, \ldots, \mathtt{Y}_n \mapsto \mathtt{nil}]$

となる．

意味関数は以下の様に式 (Expressions)，パターン (Patterns)，命令 (Commands) またはプログラム (Programs) の四種類によって定義される (図 4.2)．このとき，A から B への単射の写像を $\mathrm{A} \hookrightarrow \mathrm{B}$ によって示す．

$\mathcal{E}$: Expressions $\to (\mathtt{Stores}^{\mathtt{P}} \hookrightarrow \mathbb{D}_\perp)$

$\mathcal{Q}$: Patterns $\to (\mathtt{Stores}^{\mathtt{P}} \hookrightarrow \mathbb{D}_\perp \times \mathtt{Stores}^{\mathtt{P}}_\perp)$

$\mathcal{C}$: commands $\to (\mathtt{Stores}^{\mathtt{P}} \hookrightarrow \mathtt{Stores}^{\mathtt{P}}_\perp)$

$\mathcal{P}$: Programs $\to (\mathbb{D} \hookrightarrow \mathbb{D}_\perp)$

プログラム p は部分関数 $\mathcal{P}[\![\mathtt{p}]\!]$ を示す．本稿では p が R-WHILE のプログラムであることを明示するために $[\![\mathtt{p}]\!]^{\text{R-WHILE}}$

式評価 $\mathcal{E}$ は状態 σ の式 E を $\mathbb{D}$ の値へ評価する．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=?) の評価は予想されるとおりである．また $\mathcal{E}$ は $\perp$ へ評価することができる．もし E の値が nil だった場合 hd E と tl E の値は定義されないからである．

パターン評価 $\mathcal{Q}$ は状態 σ のパターン Q を値と状態の組へ評価する．Q において使われるすべての変数は結果の状態で nil に縛られる．これは，結果の状態を構築するすべての値を状態から移動する．いくつかの σ のために $\mathcal{Q}[\![\mathtt{Q}]\!]\ \sigma = (\mathcal{E}[\![\mathtt{Q}]\!],\sigma')$ とする．任意の Q において，表示 $\mathcal{Q}[\![\mathtt{Q}]\!]$ は単射関数である．そのため，逆関数 $\mathcal{Q}[\![\mathtt{Q}]\!]^{-1}$ はただ一つである．逆パターン評価 $\mathcal{Q}[\![\mathtt{Q}]\!]^{-1}(\mathtt{d},\sigma')$ は値と状態の組から Q のすべての変数の新しい状態をとる．またこれは，定数 d の対応する部分に縛られおり，与えられるそれらの変数は状態 σ において nil に縛られる．もし $\mathcal{Q}$ が Q と σ について定義されている場合，$\mathcal{Q}[\![\mathtt{Q}]\!]^{-1}(\mathcal{Q}[\![\mathtt{Q}]\!]\sigma) = \sigma$ が成り立つ．パターン評価とその逆は可逆的置き換えの意味を定義するために用いられる．

ここからは命令評価 $\mathcal{C}$ について述べる．非可逆な言語における代入は破壊的である．それらは左辺の変数の値を上書きする．そして，代入を行った後，元の値を再び構築することはできない．そのため，可逆言語で基本的に用いることができない．そのかわりに，可逆的代入 X ^= E を用いる．これは，もし X の値が nil であるとき，X に E の値にし，X の値が E の値と等しいとき，X の値を nil にするというものである．前者の場合 E は複製される．また，後者の場合 X と E の等しさが主張される．以下で，更新演算子 $\odot$ を定義することによって意味論を正式なものにする．$\odot$ は

$$\mathtt{d} \odot \mathtt{e} \quad = \quad \begin{cases} \mathtt{e} & \text{if } \mathtt{d} = \mathtt{nil} \\ \mathtt{nil} & \text{if } \mathtt{d} = \mathtt{e} \neq \mathtt{nil} \end{cases}$$

と定義する．こうすることで，可逆的代入を以下の様に定義することができ，

$$\mathcal{C}[\![\mathtt{X\ \hat{}=\ E}]\!](\sigma \uplus [\mathtt{X} \mapsto \mathtt{d'}]) = \sigma \uplus [\mathtt{X} \mapsto \mathtt{d'}]$$

となる．このとき，$\mathtt{d'} = \mathtt{d} \odot \mathcal{E}[\![\mathtt{E}]\!]\sigma$ である．E が評価される σ において，左辺の X に対する束縛を含まない．これは直和 $\sigma \uplus [\mathtt{X} \mapsto \mathtt{d}]$ であることから導かれる．それゆえ，もし E に X が現れた場合，代入は定義されない．そうでない場合，無効な代入 X ^= X は X がどのような値であっても X に nil をセットする．そして，$\mathcal{C}[\![\mathtt{X\ \hat{}=\ X}]\!]$ は単射ではなくなってしまう．

可逆的置き換え Q <= R は Q と R の両辺を置き換えることによって変数を更新する．$\mathcal{C}[\![\mathtt{Q <= R}]\!]\ \sigma$ は最初の評価 $\mathcal{Q}[\![\mathtt{R}]\!]\ \sigma$ によって定義される．このとき，$\mathcal{Q}[\![\mathtt{R}]\!]\ \sigma$ は $(\mathtt{d},\sigma')$ である．また，$Var(\mathtt{R})$ のすべての変数が nil である場合を除き，d は $\mathcal{E}[\![\mathtt{R}]\!]\ \sigma$ であり，状態 σ' は σ である．そしてこれに逆評価の $\mathcal{Q}[\![\mathtt{Q}]\!]^{-1}(\mathtt{d},\sigma')$ が実行され，可逆的置き換えの最終状態である σ'' を得る．このとき $Var(\mathtt{Q})$ のすべての変数は状態 σ において nil にならなくてはならない．逆評価の $\mathcal{Q}[\![\mathtt{Q}]\!]^{-1}(\mathtt{d},\sigma')$ は $\mathtt{d} = \mathcal{E}[\![Q]\!]\sigma''$ のような d の一部に縛られる．

可逆条件分岐 if E then C else D fi F において，制御フローの分岐はテスト E に依存する．テスト E が true であった場合，命令 C が実行される．そして，アサーション F は必ず true でなくてはならない．テスト E が false であった場合，命令 D が実行される．そして，アサーション F は false でなくてはならない．もし出口の F の値と入口の E の値が対応していない場合，条件分岐は定義されない．アサーション F は条件分岐の実行が後方決定的であることを主張する．

可逆的繰り返し from E do C loop D until F はフローチャートの様に図式化することができる（図 4.3）．意味論は以下のとおりである．繰り返しの入口において，アサーション E は true でなくてはならない（図 4.3の t の部分）．その後，命令 F は実行される．そのとき，テスト F が ture である場合，繰り返しは終了する．もし F が false であった場合，命令 D が実行される．そして，アサーション E は false でなくてはならない（図 4.3の f の部分）．もし E の値が要求される値とことなる場合，繰り返しは定義されない．

4.4 プログラム逆変換器

プログラミング逆変換器は従来の言語では一般的に利用することができない．しかし可逆プログラミング言語である R-WHILE にはプログラム逆変換器 $\mathcal{I}$ が定義されている (図 4.4)．それによって反転されたプログラ

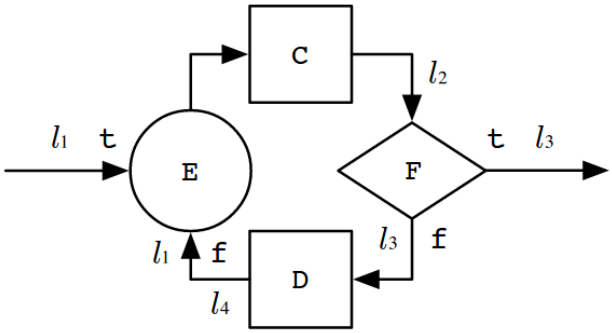

図 4.3: while loop のフローチャート

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

図 4.4: R-WHILE の逆変換器 $\mathcal{I}$

ムを再帰的降下によって得ることができる．可逆的代入は自己逆の為変化はない．可逆的置き換えは両辺を入れ替えるだけである．連続して命令を実行する命令の逆はそれぞれの逆命令の順序を逆にされたものである．テストとアサーションは制御フロー演算子ではそれぞれの役割を交代する．そして，プログラムは入力と出力の変数と入れ替え，プログラムの本体を逆変換することで逆変換されたプログラムを求めることができる．

例えば，図 4.5はリストを反転させる R-WHILE のプログラム reverse と逆変換器 $\mathcal{I}$ によって求められた reverse のプログラム $\mathcal{I}$ [[reverse]] である．逆変換されたプログラムは元のプログラムと同一の動作をする．$\mathcal{I}$ [[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．

プログラム P は二回逆変換を行うことで元のプログラムが求められる（$\texttt{P} = \mathcal{I}[\![\mathcal{I}[\![\texttt{P}]\!]]\!]$）．

4.5 糖類構文としての書き換え規則

プログラムは糖類構文としての可逆版の書き換え規則をもちいることで読みやすくなる．糖類構文によって，複数の変数 $[\texttt{X}_1,\ldots,\texttt{X}_n]$ は複数の規則 $\texttt{Rule}_1 \ldots \texttt{Rule}_n$ によって書き換えられる．規則 $\texttt{Rule}_j$ は

1. $[\texttt{Q}_1,\ldots,\texttt{Q}_n] \Rightarrow [\texttt{E}_1,\ldots,\texttt{E}_n]$ または，
2. $[\texttt{Q}_1,\ldots,\texttt{Q}_n] \Rightarrow \texttt{C;}$

のいずれかである．それぞれの意味は次のようである．もし $\texttt{X}_1$,...,$\texttt{X}_n$ の値が連続して試される規則 $\texttt{Rule}_j$ のパターン $\texttt{Q}_1$,...,$\texttt{Q}_n$ に一致した場合，$\texttt{Q}_i$ は縛られる．次に $\texttt{Rule}_j$ の形式に応じて，$\texttt{X}_1$,...,$\texttt{X}_n$ が $\texttt{E}_1$,...,$\texttt{E}_n$ に縛れるか，命令 C が実行され，$\texttt{X}_i$ も更新される．変数が同じ値と一致する場合，繰り返し変数を含むパターンが一致する．

```
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 $\mathcal{I}[\![\texttt{reverse}]\!]$

図 4.5: 逆変換器 $\mathcal{I}$ の例

4.6 可逆チューリング機械から R-WHILE への変換

ここ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．

4.6.1 可逆チューリング機械を模倣する R-WHILE プログラム

図 4.6a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．なお記述には変換規則を用いた．

main プログラム（図 4.6a）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 $\bar{b}$ 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 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 4.6b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．$\mathcal{T}[\![t]\!]^*$ は遷移規則から R-WHILE の書き換え規則への変換器 $\mathcal{T}$（図 4.7）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 4.7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．

マクロ MOVEL（図 4.6c）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．

4.7 模倣できていることの証明

```
read R;
  Q ^= q_s;
  T <= (nil b R);
  from (=? Q q_s) loop
    STEP(Q,T)
  until (=? Q q_f);
  (nil b 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
 [b,nil] => [nil,nil]
 [S,STK] => [nil,(S.STK)]
```

(d) マクロ PUSH

図 4.6: RTM を模倣する R-WHILE プログラム

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

図 4.7: 遷移規則から R-WHILE の書き換え規則への変換器 $\mathcal{T}$

第 5 章

結論

5.1 **結果**

5.2 **考察**

謝辞

付録

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] Ryo Aoki,Shintaro Shibata,Tetsuo Yokoyama.: A universal reversible Turing machine program inreversible programming language R-WHILE(2016).

[3] 丸岡 章: “計算理論とオートマトン言語理論 コンピュータの原理を明かす “, pp.4–5,142–169, サイエンス社 (2005).

[4] 米田 政明, 広瀬 貞樹, 大里 延康, 大川 知: “オートマトン・言語理論の基礎 “, pp.60,85–103, 近代科学社 (2003).

[5] 森田 憲一: “可逆計算 ナチュラルコンピューティングシリーズ Vo5“, pp.13–22, 近代科学社 (2012).

[6] 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)

[7] Gluck, R. and Yokoyama, T.: A Linear-Time Self-Interpreter of a Reversible Imperative Lan guage.

[8] 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)

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