同期的データフロープログラミングにおける逆計算の構成方式

白井 瑞貴　森口 草介　渡部 卓雄

プログラムの出力から入力を求めることを逆計算という．本論文では同期的データフロープログラミングにおける逆計算を扱う．特に，順計算と逆計算の両方を逐次的に行えるという性質を保証しながらプログラムを構成する手法について議論する．同期的データフロープログラムは，入力をすべて読んで出力を返して停止するのではなく，列として順に与えられる入力に対して出力を返し続ける必要がある．したがってその逆計算の手法は一般的なパラダイムのものと異なる．本論文では同期的データフロープログラミングのパラダイムを列の変換としてモデル化し，それに対する逐次的な逆計算の概念を定義する．さらに，逐次的な逆計算が可能なシステムを可逆関数を用いて構成する方法を定義することで，逐次的な逆計算のための制約を関数の可逆性に帰着する．この議論は逐次的な逆計算を保証する同期的データフロー言語の設計の指針になりうる．

1　はじめに

プログラムの出力から入力が一意に定まるときにそれを求めることを**逆計算**という．逆計算の概念はソフトウェア開発の過程でしばしば現れる．対になる順計算と逆計算を扱う例として，データのエンコードとデコードが挙げられる．順計算と逆計算の両方のプログラムを個別に記述することはバグを引き起こす要因であり，両者が対になることを保証する手法がいくつか研究されている [7] [9] [8] [12] [5] [4] [1].

本論文では**逐次的な列の変換**を行うシステムにおける逆計算を扱う．ここで逐次的な列の変換とは，入力列をすべて読んでから出力列を計算するのではなく，入力列の要素が与えられるたびに対応する出力列の要素を計算するような変換のこととする．このような変換を記述するプログラミングパラダイムはリアクティブシステムを記述する方法の一つであり，同期的データフロー言語 Lustre [2]，Signal [6] や関数リアクティブ言語 Emfrp [10] の実行モデルは逐次的な列の変換と見なせる．

逐次的な列の変換を行うシステムの開発においても，変換とその逆変換が対になることを保証することは有用である．例えば，列として与えられるデータのエンコードやデコードを行うプログラムには，エンコードとデコードの両者が対になることの保証が求められる．また，出力の列からテストケースを生成する手法への応用が考えられる．

逆計算に関連する研究の多くは，入力をすべて読んで出力を返して停止するプログラムを対象とするものであり，同期的データフロープログラムやリアクティブプログラムを対象とした研究は著者が調査した限りではない．例えば Janus [7] をはじめとする可逆プログラミング言語は記述されたプログラムの可逆性を保証するが，保証される可逆性は逐次的な列の変換の逆計算に求められる性質とは異なる．

逐次的な列の変換では入力列の要素を読むたびに出力列の新しい要素が得られる．これと同様の性質が逆計算でも成り立つ場合がある．つまり，逐次的な列の変換の逆変換もまた逐次的な列の変換となる場合がある．例えば，整数の入力列に対してその累積和を出力列とする変換の逆変換は逐次的な列の変換である．この変換は入力列 $1, 2, 3, 4, 5$ に対して出力列

Construction of Inverse Computation in Synchronous Dataflow Programming

Mizuki Shirai, Sosuke Moriguchi, Takuo Watanabe, 東京工業大学情報理工学院, Dept. of Computer Science, Tokyo Institute of Technology.

$1,3,6,10,15$ を返すが，その逆変換においては，出力列の要素をすべて読まなくても $1,3,6$ まで読めば入力列の $1,2,3$ を復元できる．つまり逆計算も逐次的な列の変換である．

本論文では同期的データフロープログラミングのパラダイムを列の変換にモデル化し，それに対する逐次的な逆計算の概念を定義する．さらに，逐次的な逆計算が可能な変換を可逆関数を用いて構成する方法を定義することで，逐次的な逆計算のための制約を関数の可逆性に帰着する．

本論文の構成を以下に示す．2 節では逐次的な列の変換とその逆変換について定義を与える．3 節では構成子という関数を導入し，順方向と逆方向の逐次的な列の変換を同時に構成する手法を示す．4 節では逐次的な列の変換の構成の具体例を与える．5 節では，構成された順方向と逆方向の変換が，それぞれ互いの逆変換であることを証明する．6 節では任意の逐次的に可逆な列の変換が構成子項で表現できることを示す．7 節では構成子項を反転する方法を示し，反転された構成子項と元の構成子項に対応する変換が対になることを示す．8 節では関連研究を紹介する．9 節では本論文のまとめと今後の課題を述べる．

2 逐次的な列の変換と逆変換

定義 1 (列)**.** 添字の集合 I から要素の集合 X への写像を X の**列**という．添字の集合 I は有限集合 $\{1,\ldots,n\}$ または正整数の集合 $\mathbb{Z}_{\geq 1}$ である．列は次のように表記される．

- $[a_1,\ldots,a_n]$
- $[a_i \mid i \in I]$

また，X の列すべてからなる集合を $[X]$ と表記する．

列の先頭から有限個または無限個の要素を欠かさずに取り出して得られる列を**プレフィクス**と呼ぶ．

定義 2 (列のプレフィクス)**.** 列 $s = [x_i \mid i \in I]$ と $s' = [x'_i \mid i \in I']$ について，$I \subseteq I'$ かつ任意の $i \in I$ で $x_i = x'_i$ が成り立つならば，s は s' の**プレフィクス**であるといい，$s \prec s'$ と表記する．

定義 3 (列変換)**.** 部分写像 $[X] \rightharpoonup [Y]$ を X から Y への**列変換**という．

とくに，入力列の要素を先頭から i 個読むことで，出力列の要素が先頭から $(i-d)$ 個だけ得られるような列変換を d-**逐次列変換**という．Emfrp [10] などの言語で記述したリアクティブシステムは 0-逐次列変換としてモデル化できる．また，一般の d-逐次列変換は 0-逐次列変換の逆変換として表れることがある．

定義 4 (d-逐次列変換)**.** 列変換 $st\colon [X] \rightharpoonup [Y]$ について，X の有限列から Y への部分関数 w と非負整数 d が存在して，列 $[x_i \mid i \in I] \in [X]$, $[y_i \mid i \in I'] \in [Y]$ に対して，$st([x_i \mid i \in I]) = [y_i \mid i \in I']$ のとき，かつそのときに限り $I' = \{i-d \mid i \in I, i-d \geq 1\}$ かつ $i \in I'$ で $y_i = w([x_1,\ldots,x_{i+d}])$ が成り立つならば，st を d-**逐次列変換**という．また，このとき列変換 st と関数 w が対応するという．

例 1. 式 $y_i = x_{i+1} + x_i$ で表される列変換は，1-逐次列変換である．

例 2. 整数列からその累積和の列を得る列変換は式 $y_i = \sum_{k=1}^{i} x_k$ で表される．この列変換は 0-逐次列変換である．

例 3. 列変換 st_3, st'_3 をそれぞれ次に定義する．

$$
\begin{aligned}
&st_3([x_i \mid i \in I]) = \left[y_i \mid i \in I'\right] \\
&\quad \text{where } y_1 = 0, \\
&\qquad\qquad y_{i+1} = x_i \quad (i \in I), \\
&\qquad\qquad I' = \{1\} \cup \{i+1 \mid i \in I\} \\
&st'_3([x_i \mid i \in I]) = [y_i \mid i \in I] \\
&\quad \text{where } y_1 = 0, \\
&\qquad\qquad y_i = x_{i-1} \quad (i \in I \setminus \{1\}),
\end{aligned}
$$

st_3, st'_3 の変換例を以下に示す．

- $st_3([1,2,3]) = [0,1,2,3]$
- $st'_3([1,2,3]) = [0,1,2]$
- $st_3([]) = [0]$
- $st'_3([]) = []$

st_3 は, 入力列に対して出力列の長さが 1 だけ長いため，d-逐次列変換ではない．一方で st'_3 は 0-逐次列変換である．

ある列 s_x が d-逐次列変換 st の定義域に含まれるとき，その列のプレフィクス s'_x もまた st の定義域に含まれる．また，s'_x に対応する出力列 $st(x'_x)$ は，元の出力列 $st(s_x)$ のプレフィクスになる．

補題 1. 任意の d-逐次列変換 st について，$st(s_x) = s_y$ かつ $s'_x \prec s_x$ ならば $st(s'_x) \prec s_y$ が成り立つ．

証明. $st([x_i \mid i \in I_x]) = [y_i \mid i \in I_y]$ とする．定義 4 から関数 w が存在して $i \in I_y$ で (1) が成り立つ．

$$y_i = w([x_1, \ldots, x_{i+d}]) \tag{1}$$

$[x_i \mid i \in I_x]$ の任意のプレフィクス $[x_i \mid i \in I'_x]$ について，$I'_y = \{i - d \mid i \in I'_x,\ i - d \geq 1\}$ とおけば，$I'_y \subseteq I_y$ より $i \in I'_y$ で (1) が成り立つ．よって定義 4 より $st([x_i \mid i \in I'_x]) = [y_i \mid i \in I'_y] \prec [y_i \mid i \in I_y]$ が成り立つ． □

列変換の逆変換について考察する．すなわち，変換後の列から変換前の列を復元するような変換について考える．有限列の列変換では，出力列から入力列のプレフィクスを復元できるが，末尾からいくつかの要素を復元できないという場合がある．

例 4. $X = \{0, \ldots, 99\}$ とする．列変換 $st_4 \colon [X] \to [X]$ を式 $y_i = \lfloor x_{i-1}/10 \rfloor \times 10 + (x_i \bmod 10)$ で表す．ただし $x_0 = 0$ とする．st_4 は，入力列の $(i-1)$ 番目の要素の十の位と i 番目の要素の一の位の合成を，出力列の i 番目の要素とする 0-逐次列変換である．変換例を次に示す．

$$st_4([12, 34, 56]) = [2, 14, 36]$$

st_4 の逆変換を考えると，出力列 $[2, 14, 36]$ からは入力列のプレフィクス $[12, 34]$ を復元できるが，3 番目の要素の十の位である 5 を復元できない．

ある列変換に対して，出力列から入力列のプレフィクスを得る列変換が存在すれば，その列変換が元の列変換の**逆列変換**であると定義する．

定義 5 (逆列変換)**.** 列変換 st, st' について $st(s_x) = s_y$ ならば $st'(s_y) \prec s_x$ が成り立つとき，st' は st の**逆列変換**であるという．

st' が列変換 st の逆列変換であっても，st が st' の逆列変換であるとは限らない．また逆列変換は一意には定まらず，そのなかには有用でないものが含まれている．例えば，常に長さ 0 の列を返す列変換は任意の d-逐次列変換の逆列変換である．定義 5 に加えて制約をかけることで，逆列変換を有用なものに絞り込む．

定義 6 (d-逐次逆列変換)**.** 列変換 st, st' について，st' が d-逐次列変換でありかつ st の逆列変換であるとき，st' は st の d-**逐次逆列変換**であるという．

定義 7 ((d, d')-逐次可逆列変換)**.** 列変換 st について，st が d-逐次列変換であり，かつ st の d'-逐次逆列変換が存在するとき，st を (d, d')-**逐次可逆列変換**という．

例 5. 式 $y_i = x_i - x_{i-1}$（ただし $x_0 = 0$ とする）で表される 0-逐次列変換は，例 2 に示した累積和を計算する 0-逐次列変換の逆列変換である．したがって例 2 の列変換は $(0, 0)$-逐次可逆列変換である．

例 6. $X = \{0, \ldots, 99\}$ とする．式 $y_i = \lfloor x_{i+1}/10 \rfloor \times 10 + (x_i \bmod 10)$ で表される 1-逐次列変換 $st_6 \colon [X] \to [X]$ は，例 4 に示した 0-逐次列変換 st_4 の逆列変換である．したがって st_4 は $(0, 1)$-逐次可逆列変換である．

任意の (d, d')-逐次可逆列変換には，互いが逆列変換になるような (d', d)-逐次可逆列変換が存在する．詳細は 6 節に示す．

ただし (d, d')-逐次可逆列変換の d'-逐次逆列変換は一意には定まらず，(d', d)-逐次可逆列変換であるとは限らない．例えば例 6 の列変換 st_6 は $(0, 1)$-逐次可逆列変換 st_4 の 1-逐次逆列変換であるが，st_6 は $(1, 0)$-逐次可逆列変換ではない．変換例

$$st_6([11, 22, 33]) = [21, 32]$$
$$st_6([91, 22, 33]) = [21, 32]$$

が示すように，入力列の 1 番目の要素の十の位を，出力列から復元できないためである．もし st_6 が部分写像で $0 \leq x_1 \leq 9$ を満たさない入力列に対して未定義であれば，st_6 は $(1, 0)$-逐次可逆列変換になる．

3 逐次可逆列変換の構成手法

互いが逆列変換になるような (d, d')-逐次可逆列変換と (d', d)-逐次可逆列変換を構成する手法を提案する．

定義 8 (構成子項)**.** 図 1 に示す関数を**構成子**という．構成子の適用によって得られる値 $e \in E\,(X, Y)$ を**構成子項**という．構成子項からは，自身を得るために適用された構成子の種類とその引数を取得できる．

$X \rightleftharpoons Y$ という表記は部分関数 $X \rightharpoonup Y$ で可逆なものすべてからなる集合を表すものであり，$X \rightleftharpoons Y \stackrel{\text{def}}{=} \{f \colon X \rightharpoonup Y \mid \exists g \colon Y \rightharpoonup X \text{ s.t. } f(x) = y \iff g(y) = x\}$ で定義される．したがって $\mathit{map\text{-}fold}(a_0, f, g)$ の引数 $f \colon A \to (X \rightleftharpoons Y)$ には，任意の $a \in A$ への部分適用で得られる部分関数

$$
\begin{aligned}
\mathit{map\text{-}fold}&\colon A \times (A \to (X \rightleftharpoons Y)) \times (A \to X \to A) \to E\,(X, Y)\\
\mathit{delay}&\colon X \to E\,(X, X)\\
\mathit{hasten}&\colon X \to E\,(X, X)\\
\ggg&\colon E\,(X, Y) \times E\,(Y, Z) \to E\,(X, Z)\\
\otimes&\colon E\,(X_1, Y_1) \times E\,(X_2, Y_2) \to E\,(X_1 \times X_2, Y_1 \times Y_2)
\end{aligned}
$$

図 1　構成子

$f(a)\colon X \rightharpoonup Y$ が逆関数を持つことが要求される．

$\mathit{map\text{-}fold}(a_0, f, g)$ の引数に対する可逆性の制約によって，列変換全体の逐次的な逆計算が可能になる．言い換えれば，列変換の逐次的な逆計算のための制約は関数の可逆性に帰着できる．

部分適用の結果が可逆になる関数は可逆計算やプログラム逆化の分野で研究されている．例えば Sparcl [8] は部分適用の結果が可逆になる関数を記述するプログラミング言語である．また西田らの手法 [9] は，与えられた項書き換え系に対して，入力のいくつかと出力から残りの入力を計算するような系を生成する．本研究は関数の可逆性に関してこれらの研究と同様の立場をとる．

構成子項の意味論を示す．構成子項から列変換を得る関数 $\mathcal{F}, \mathcal{B}$ をそれぞれ図 2 と図 3 に定義する．また，構成子項から自然数のペアを得る関数 $\mathcal{D}$ を図 4 に定義する．

構成子項 $e \in E\,(X, Y)$ は，互いが逆列変換になるような，X から Y への (d, d')-逐次可逆列変換および Y から X への (d', d)-逐次可逆列変換に対応している．$e \in E\,(X, Y)$ に対して，関数 $\mathcal{F}$ を適用すると X から Y への (d, d')-逐次可逆列変換が得られ，関数 $\mathcal{B}$ を適用すると Y から X への (d', d)-逐次可逆列変換が得られる．また，関数 $\mathcal{D}$ を適用すると d と d' の値が得られる．

構成子項 $\mathit{map\text{-}fold}(a_0, f, g)$ で定義される列変換の出力列の要素は，関数 $f\colon A \to (X \rightleftharpoons Y)$ によって得られる．f に渡される引数 $a_i \in A$ をアキュムレータといい，アキュムレータは初期値 $a_0 \in A$ を関数 $g\colon A \to X \to A$ で更新することで得られる．

構成子 *map-fold* が作る列変換は $(0, 0)$-逐次可逆列変換である．また，任意の $(0, 0)$-逐次可逆列変換は *map-fold* によって表現できる．

構成子項 $\mathit{delay}(x_0)$ は，入力列の要素をひとつ遅らせて出力列の要素とする $(0, 1)$-逐次可逆列変換を定義する．構成子項 $\mathit{hasten}(x_0)$ は $\mathit{delay}(x_0)$ の逆である．

構成子 $\ggg$ と構成子 $\otimes$ は列変換の合成に対応している．$\ggg$ は直列の結合で，$\otimes$ は並列の結合である．

4　逐次可逆列変換の構成例

定義 9 (map)**.** 列の各要素に可逆関数 $r\colon X \rightleftharpoons Y$ を適用するような列変換に対応する関数 $\mathit{map}\colon (X \rightleftharpoons Y) \to E\,(X, Y)$ を次に定義する．

$$
\begin{aligned}
&\mathit{map}(r) = \mathit{map\text{-}fold}(\top, f, g)\\
&\quad \text{where } f(a)(x) = r(x),\ g(a)(x) = \top
\end{aligned}
$$

map は $\mathit{map\text{-}fold}\colon A \times (A \to (X \rightleftharpoons Y)) \times (A \to X \to A) \to E\,(X, Y)$ の特別な場合といえる．*map* の計算にはアキュムレータが必要ないため $A = \{\top\}$ とした．

定義 10 (id)**.** 恒等関数に *map* を適用して得られる構成子項 $id \in E(X, X)$ を次に定義する．

$$
id = \mathit{map}(r) \text{ where } r(x) = x
$$

id は入力列をそのまま出力列とする列変換に対応している．

例 7. 例 2 に示した累積和を計算する列変換は，$(0, 0)$-逐次可逆列変換であるから *map-fold* だけで構成できる．対応する構成子項を e とする．

$$
\begin{aligned}
f(a)(x) &= a + x\\
e &= \mathit{map\text{-}fold}(0, f, f)
\end{aligned}
$$

$f(a)$ には逆関数 $f(a)^{-1}(y) = y - a$ が存在するから，f は可逆性の制約を満たしている．

例 8. 例 4 に示した，十の位をひとつ遅らせて出力す

$$
\begin{aligned}
&\mathcal{F}\left[\!\left[\textit{map-fold}(a_0, f, g)\right]\!\right]\left([x_i \mid i \in I]\right) = [y_i \mid i \in I] \\
&\quad \text{where } y_i = f(a_{i-1})(x_i), \\
&\qquad\quad a_i = g(a_{i-1})(x_i) \\
&\mathcal{F}\left[\!\left[\textit{delay}(x_0)\right]\!\right]\left([x_i \mid i \in I]\right) = [x_{i-1} \mid i \in I] \\
&\mathcal{F}\left[\!\left[\textit{hasten}(x_0)\right]\!\right]\left([x_i \mid i \in I]\right) = \\
&\quad \begin{cases} [x_{i+1} \mid i \in I'] \text{ where } I' = \{i-1 \mid i \in I,\ i \geq 2\} & (I = \emptyset \vee x_1 = x_0) \\ \bot & (\text{otherwise}) \end{cases} \\
&\mathcal{F}\left[\!\left[e_1 \ggg e_2\right]\!\right] = \mathcal{F}\left[\!\left[e_2\right]\!\right] \circ \mathcal{F}\left[\!\left[e_1\right]\!\right] \\
&\mathcal{F}\left[\!\left[e_1 \otimes e_2\right]\!\right]\left(\left[\left\langle x_i^1, x_i^2 \right\rangle \mid i \in I\right]\right) = \left[\left\langle y_i^1, y_i^2 \right\rangle \mid i \in I_1 \cap I_2\right] \\
&\quad \text{where } \left[y_i^1 \mid i \in I_1\right] = \mathcal{F}\left[\!\left[e_1\right]\!\right]\left(\left[x_i^1 \mid i \in I\right]\right), \\
&\qquad\quad \left[y_i^2 \mid i \in I_2\right] = \mathcal{F}\left[\!\left[e_2\right]\!\right]\left(\left[x_i^2 \mid i \in I\right]\right)
\end{aligned}
$$

図 2　関数 $\mathcal{F}\colon E\,(X, Y) \to [X] \rightharpoonup [Y]$

$$
\begin{aligned}
&\mathcal{B}\left[\!\left[\textit{map-fold}(a_0, f, g)\right]\!\right]\left([y_i \mid i \in I]\right) = [x_i \mid i \in I] \\
&\quad \text{where } x_i = f(a_{i-1})^{-1}(y_i), \\
&\qquad\quad a_i = g(a_{i-1})(x_i) \\
&\mathcal{B}\left[\!\left[\textit{delay}(x_0)\right]\!\right] = \mathcal{F}\left[\!\left[\textit{hasten}(x_0)\right]\!\right] \\
&\mathcal{B}\left[\!\left[\textit{hasten}(x_0)\right]\!\right] = \mathcal{F}\left[\!\left[\textit{delay}(x_0)\right]\!\right] \\
&\mathcal{B}\left[\!\left[e_1 \ggg e_2\right]\!\right] = \mathcal{B}\left[\!\left[e_1\right]\!\right] \circ \mathcal{B}\left[\!\left[e_2\right]\!\right] \\
&\mathcal{B}\left[\!\left[e_1 \otimes e_2\right]\!\right]\left(\left[\left\langle y_i^1, y_i^2 \right\rangle \mid i \in I\right]\right) = \left[\left\langle x_i^1, x_i^2 \right\rangle \mid i \in I_1 \cap I_2\right] \\
&\quad \text{where } \left[x_i^1 \mid i \in I_1\right] = \mathcal{B}\left[\!\left[e_1\right]\!\right]\left(\left[y_i^1 \mid i \in I\right]\right), \\
&\qquad\quad \left[x_i^2 \mid i \in I_2\right] = \mathcal{B}\left[\!\left[e_2\right]\!\right]\left(\left[y_i^2 \mid i \in I\right]\right)
\end{aligned}
$$

図 3　関数 $\mathcal{B}\colon E\,(X, Y) \to [Y] \rightharpoonup [X]$

$$
\begin{aligned}
&\mathcal{D}\left[\!\left[\textit{map-fold}(a_0, f, g)\right]\!\right] = \langle 0, 0 \rangle \\
&\mathcal{D}\left[\!\left[\textit{delay}(x_0)\right]\!\right] = \langle 0, 1 \rangle \\
&\mathcal{D}\left[\!\left[\textit{hasten}(x_0)\right]\!\right] = \langle 1, 0 \rangle \\
&\mathcal{D}\left[\!\left[e_1 \ggg e_2\right]\!\right] = \left\langle d_1 + d_2, d_1' + d_2' \right\rangle \\
&\quad \text{where } \left\langle d_1, d_1' \right\rangle = \mathcal{D}\left[\!\left[e_1\right]\!\right],\ \left\langle d_2, d_2' \right\rangle = \mathcal{D}\left[\!\left[e_2\right]\!\right] \\
&\mathcal{D}\left[\!\left[e_1 \otimes e_2\right]\!\right] = \left\langle \max(d_1, d_2), \max(d_1', d_2') \right\rangle \\
&\quad \text{where } \left\langle d_1, d_1' \right\rangle = \mathcal{D}\left[\!\left[e_1\right]\!\right],\ \left\langle d_2, d_2' \right\rangle = \mathcal{D}\left[\!\left[e_2\right]\!\right]
\end{aligned}
$$

図 4　関数 $\mathcal{D}\colon E\,(X, Y) \to \mathbb{N} \times \mathbb{N}$

る $(0, 1)$-逐次可逆列変換 st_4 を構成する．

$$
\begin{aligned}
f(x) &= \langle x \bmod 10, \lfloor x/10 \rfloor \rangle \\
e &= \textit{map}(f) \ggg (\textit{id} \otimes \textit{delay}(0)) \ggg \textit{map}(f^{-1})
\end{aligned}
$$

$\mathcal{F}\left[\!\left[e\right]\!\right] = st_4$ が成り立つ．また $\mathcal{B}\left[\!\left[e\right]\!\right]$ は $(1, 0)$-逐次可逆列変換であり，st_4 は $\mathcal{B}\left[\!\left[e\right]\!\right]$ の逆列変換である．このことは，例 6 の列変換 st_6 が st_4 の逆列変換であるが st_4 は st_6 の逆列変換ではないことと対照的である．両者の違いは，$\mathcal{B}\left[\!\left[e\right]\!\right]$ が $\mathcal{B}\left[\!\left[\textit{delay}(0)\right]\!\right]$ を含むために初項の十の位が 0 でない列に対して未定義であることによって生じる．

例 9. 入力列の $(2i-1)$ 番目の要素と $2i$ 番目の要素を入れ替える $(1, 1)$-逐次可逆列変換 st_9 を構成する．st_9 の変換例を次に示す．

- $st_9([1, 2, 3, 4, 5]) = [2, 1, 4, 3]$
- $st_9([1, 2, 3, 4, 5, 6]) = [2, 1, 4, 3, 6]$

2 番目の例では，st_9 が 1-逐次列変換であるために，出力列の 6 番目の要素を 5 とせずに打ち切ることに留意が必要である．st_9 に対応する構成子項 e は次の

ように構成できる．

$$f_1(i)(x) = \begin{cases} \langle x, \mathrm{None}\rangle & (i \bmod 2 = 1) \\ \langle \mathrm{None}, x\rangle & (i \bmod 2 = 0) \end{cases}$$

$$f_2(i)(\langle x, x'\rangle) = \begin{cases} x' & (i \bmod 2 = 1 \wedge x = \mathrm{None}) \\ x & (i \bmod 2 = 0 \wedge x' = \mathrm{None}) \\ \bot & (\text{otherwise}) \end{cases}$$

$g(i)(x) = i + 1$

$e_1 = \mathit{map\text{-}fold}(1, f_1, g)$

$e_2 = \mathit{delay}(\mathrm{None}) \otimes \mathit{hasten}(\mathrm{None})$

$e_3 = \mathit{map\text{-}fold}(1, f_2, g)$

$e = e_1 \ggg e_2 \ggg e_3$

$\mathcal{F}[\![e_1]\!]$ は入力列の i 番目の要素を i の偶奇でペアの左右に分配する．ペアの反対側の要素には None が入る．$\mathcal{F}[\![e_2]\!]$ は，左（奇数番目）の要素を遅らせて，右（偶数番目）の要素を早めることで，両者の順番を逆転させる．$\mathcal{F}[\![e_3]\!]$ はペアの左右から None でない要素を取り出す．

$\mathcal{F}[\![e]\!]$ は $\mathcal{F}[\![\mathit{hasten}(\mathrm{None})]\!]$ を含むが全域写像である．$\mathcal{F}[\![e_1]\!]$ の出力列の初項は必ず $\langle x_1, \mathrm{None}\rangle$ という形になるため，$\mathcal{F}[\![\mathit{hasten}(\mathrm{None})]\!]$ の入力列の初項は None であり，出力列が未定義になることはない．

5 構成子項の逆計算可能性の証明

補題 2. 構成子項 e について，$\mathcal{D}[\![e]\!] = \langle d, d'\rangle$ ならば $\mathcal{F}[\![e]\!]$ は d-逐次列変換であり $\mathcal{B}[\![e]\!]$ は d'-逐次列変換である．

証明. e の構造についての帰納法による．

1. ($e = \mathit{map\text{-}fold}(a_0, f, g)$ の場合) 関数
$$w([x_1, \ldots, x_i]) = f(g(\cdots g(g(a_0)(x_1))(x_2)\cdots)(x_{i-1}))(x_i)$$
の存在から，定義 4 より $\mathcal{F}[\![e]\!]$ は 0-逐次列変換である．また，関数
$$w'([y_1, \ldots, y_i]) = x_i \quad \text{where } x_k = f(a_{k-1})^{-1}(y_k) \quad (1 \le k \le i),\ a_k = g(a_{k-1})(x_k) \quad (1 \le k \le i-1)$$
の存在から，$\mathcal{B}[\![e]\!]$ は 0-逐次列変換である．
2. ($e = \mathit{delay}(x_0)$ の場合) 関数 $w([x_1, \ldots, x_i]) = x_{i-1}$ の存在から，定義 4 より $\mathcal{F}[\![e]\!]$ は 0-逐次列変換である．また，関数
$$w'([x_1, \ldots, x_{i+1}]) = \begin{cases} x_{i+1} & (x_1 = x_0) \\ \bot & (\text{otherwise}) \end{cases}$$
の存在から，$\mathcal{B}[\![e]\!]$ は 1-逐次列変換である．
3. ($e = \mathit{hasten}(x_0)$ の場合) delay の場合と同様にして成り立つ．
4. ($e = e_1 \ggg e_2$ の場合) $j \in \{1, 2\}$ で，$\mathcal{D}[\![e_j]\!] = \langle d_j, d'_j\rangle$ とする．$\mathcal{F}[\![e_1 \ggg e_2]\!]$ が $(d_1 + d_2)$-逐次列変換であることを示す．帰納法の仮定から $\mathcal{F}[\![e_j]\!]$ は d_j-逐次列変換であり，対応する関数 w_j が存在する．関数
$$w([x_1, \ldots, x_{i+d_1+d_2}]) = w_2([y_1, \ldots, y_{i+d_2}]) \quad \text{where } y_k = w_1([x_1, \ldots, x_{k+d_1}])$$
の存在から，定義 4 より $\mathcal{F}[\![e_1 \ggg e_2]\!]$ は $(d_1 + d_2)$-逐次列変換である．$\mathcal{B}[\![e_1 \ggg e_2]\!]$ が $(d'_1 + d'_2)$-逐次列変換であることも同様にして成り立つ．
5. ($e = e_1 \otimes e_2$ の場合) $j \in \{1, 2\}$ で，$\mathcal{D}[\![e_j]\!] = \langle d_j, d'_j\rangle$ とする．$m = \max(d_1, d_2)$ とおく．$\mathcal{F}[\![e_1 \otimes e_2]\!]$ が m-逐次列変換であることを示す．帰納法の仮定から $\mathcal{F}[\![e_j]\!]$ は d_j-逐次列変換であり，対応する関数 w_j が存在する．関数 $w\left(\left[\langle x_1^1, x_1^2\rangle, \ldots, \langle x_{i+m}^1, x_{i+m}^2\rangle\right]\right) = \left\langle w_1\left(\left[x_1^1, \ldots, x_{i+d_1}^1\right]\right), w_2\left(\left[x_1^2, \ldots, x_{i+d_2}^2\right]\right)\right\rangle$ の存在から，定義 4 より $\mathcal{F}[\![e_1 \otimes e_2]\!]$ は m-逐次列変換である．$\mathcal{B}[\![e_1 \otimes e_2]\!]$ が $\max(d'_1, d'_2)$-逐次列変換であることも同様にして成り立つ．

□

補題 3. 構成子項 e について，列変換 $\mathcal{F}[\![e]\!], \mathcal{B}[\![e]\!]$ は互いの逆列変換である．

証明. e の構造についての帰納法による．

1. ($e = \mathit{map\text{-}fold}(a_0, f, g)$ の場合) $\mathcal{F}[\![e]\!](s_x) = s_y \implies \mathcal{B}[\![e]\!](s_y) = s_x$ を示す．$s_x = [x_i \mid i \in I]$ とする．前件から，$i \in I$ で $a_i = g(a_{i-1})(x_i)$，$y_i = f(a_{i-1})(x_i)$ とおけば，$s_y = [y_i \mid i \in I]$ が成り立つ．$x_i = f(a_{i-1})^{-1}(y_i)$ であるから，後件が成り立つ．よって $\mathcal{B}[\![e]\!]$ は $\mathcal{F}[\![e]\!]$ の逆列変換である．逆も同様にして成り立つ．
2. ($e = \mathit{delay}(x_0)$ の場合) まず $\mathcal{F}[\![e]\!](s_x) = s_y \implies \mathcal{B}[\![e]\!](s_y) \prec s_x$ を示す．$s_x = [x_i \mid i \in I]$

とする．前件から $s_y = [x_{i-1} \mid i \in I]$ が成り立つ．$I = \emptyset$ のとき $\mathcal{B}[\![e]\!](s_y) = \mathcal{B}[\![e]\!]([]) = [] \prec s_x$ が成り立つ．$I \neq \emptyset$ のとき，$s_y(1) = x_0$ であるから，$I' = \{i-1 \mid i \in I,\ i \geq 2\}$ として $\mathcal{B}[\![e]\!](s_y) = \{s_y(i+1) \mid i \in I'\} = \{x_i \mid i \in I'\} \prec s_x$ が成り立つ．よって後件が示された．

つぎに $\mathcal{B}[\![e]\!](s_y) = s_x \implies \mathcal{F}[\![e]\!](s_x) \prec s_y$ を示す．$s_y = [y_i \mid i \in I]$ とする．$I = \emptyset$ のとき後件は成り立つ．$I \neq \emptyset$ のとき，前件から，$I' = \{i-1 \mid i \in I,\ i \geq 2\}$ として $s_x = [y_{i+1} \mid i \in I']$ と $y_1 = x_0$ が成り立つ．したがって $\mathcal{F}[\![e]\!](s_x) = [y_i \mid i \in I'] \prec s_y$ が成り立つから，後件が示された．

3. ($e = hasten(x_0)$ の場合) *delay* の場合と同様にして成り立つ．

4. ($e = e_1 \ggg e_2$ の場合) $\mathcal{F}[\![e_1 \ggg e_2]\!](s_x) = s_z \implies \mathcal{B}[\![e_1 \ggg e_2]\!](s_z) \prec s_x$ を示す．前件から $\mathcal{F}[\![e_1]\!](s_x) = s_y$, $\mathcal{F}[\![e_2]\!](s_y) = s_z$ を満たす列 s_y が存在する．したがって帰納法の仮定から (2)(3) が成り立つ．

$$\mathcal{B}[\![e_1]\!](s_y) \prec s_x \tag{2}$$
$$\mathcal{B}[\![e_2]\!](s_z) \prec s_y \tag{3}$$

補題 2 と $\mathcal{D}$ の全域性から，ある d が存在して列変換 $\mathcal{B}[\![e_1]\!]$ は d-逐次列変換である．したがって補題 1 と (2)(3) から $\mathcal{B}[\![e_1]\!] \circ \mathcal{B}[\![e_2]\!](s_z) \prec s_x$ が成り立ち，後件が示された．$\mathcal{B}[\![e_1 \ggg e_2]\!](s_z) = s_x \implies \mathcal{F}[\![e_1 \ggg e_2]\!](s_x) \prec s_z$ も同様にして成り立つ．

5. ($e = e_1 \otimes e_2$ の場合) $\mathcal{F}[\![e_1 \otimes e_2]\!](s_x) = s_y \implies \mathcal{B}[\![e_1 \otimes e_2]\!](s_y) \prec s_x$ を示す．$s_x = \left[\left\langle x_i^1, x_i^2 \right\rangle \mid i \in I\right]$, $s_y = \left[\left\langle y_i^1, y_i^2 \right\rangle \mid i \in I'\right]$ とする．前件から $k \in \{1,2\}$ で

$$\mathcal{F}[\![e_k]\!]\left(\left[x_i^k \mid i \in I\right]\right) = \left[y_i^k \mid i \in I_k\right]$$

を満たす I_1, I_2 が存在して $I' = I_1 \cap I_2$ である．帰納法の仮定から

$$\mathcal{B}[\![e_k]\!]\left(\left[y_i^k \mid i \in I_k\right]\right) \prec \left[x_i^k \mid i \in I\right]$$

が成り立つ．したがって補題 1 と $I' \subseteq I_k$ から，ある $I'_k \subseteq I$ が存在して

$$\mathcal{B}[\![e_k]\!]\left(\left[y_i^k \mid i \in I'\right]\right) = \left[x_i^k \mid i \in I'_k\right]$$

を満たす．よって $\mathcal{B}[\![e_1 \otimes e_2]\!](s_y) = \left[\left\langle x_i^1, x_i^2 \right\rangle \mid i \in I'_1 \cap I'_2\right] \prec s_x$ が成り立ち，後件が示された．$\mathcal{B}[\![e_1 \otimes e_2]\!](s_y) = s_x \implies \mathcal{F}[\![e_1 \otimes e_2]\!](s_x) \prec s_y$ も同様にして成り立つ．

□

定理 1. 構成子項 e について $\mathcal{D}[\![e]\!] = \langle d, d' \rangle$ ならば，$\mathcal{F}[\![e]\!]$ は (d, d')-逐次可逆列変換であり $\mathcal{B}[\![e]\!]$ は (d', d)-逐次可逆列変換である．また，$\mathcal{F}[\![e]\!]$ と $\mathcal{B}[\![e]\!]$ はそれぞれ互いの逆列変換である．

証明. 補題 2 と補題 3 から従う． □

6 表現力

構成子には任意の (d, d')-逐次可逆列変換を構成できるだけの表現力がある．

定理 2. 任意の (d, d')-逐次可逆列変換 st について，$st = \mathcal{F}[\![e]\!]$ を満たす構成子項 e が存在する．

証明. 構成子項 e と非負整数 n について表記 e^n を (4) で定義する．e^n は n 個の e を $\ggg$ で繋いだ構成子項を表す．

$$e^n = \begin{cases} id & (n = 0) \\ e \ggg e^{n-1} & (n > 0) \end{cases} \tag{4}$$

st は (d, d')-逐次可逆列変換であるから，d'-逐次逆列変換 st' が存在する．定義 4 より，st に対応する関数 w および st' に対応する関数 w' が存在する．図 5 で w と w' を用いて定義される構成子項 e が $st = \mathcal{F}[\![e]\!]$ を満たすことを示す．

まず関数 f_1 と関数 f_2 が可逆性の制約を満たすことを示す．f_1 は引数 x_i を $\langle y, x_i \rangle$ の右の要素としてそのまま返すので可逆である．また f_2 は引数 $\langle y_i, x \rangle$ に対して y_i だけを返すが，$i \leq d'$ では $x \neq \mathrm{None}$ のとき未定義になり，$i > d'$ では x が y_i の履歴から求まる値と異なるとき未定義になることで可逆性が保証される．

以下では $s = [x_i \mid i \in I], s' = [y_i \mid i \in I']$ とおく．

$$
\begin{aligned}
&f_1([x_1,\ldots,x_{i-1}])(x_i) = \langle y, x_i\rangle \\
&\quad \text{where } y = \begin{cases} \text{None} & (i \le d) \\ w([x_1,\ldots,x_{i-1},x_i]) & (\text{otherwise}) \end{cases} \\
&g_1([x_1,\ldots,x_{i-1}])(x_i) = [x_1,\ldots,x_{i-1},x_i] \\
&f_2([y_1,\ldots,y_{i-1}])(\langle y_i, x\rangle) = \begin{cases} y_i & (i \le d' \wedge x = \text{None}) \\ y_i & (i > d' \wedge x = w'([y_1,\ldots,y_{i-1},y_i])) \\ \bot & (\text{otherwise}) \end{cases} \\
&g_2([y_1,\ldots,y_{i-1}])(\langle y_i, x\rangle) = [y_1,\ldots,y_{i-1},y_i] \\
&e_1 = \mathit{map\text{-}fold}([\,], f_1, g_1) \\
&e_2 = \mathit{hasten}(\text{None})^d \otimes \mathit{delay}(\text{None})^{d'} \\
&e_3 = \mathit{map\text{-}fold}([\,], f_2, g_2) \\
&e = e_1 \ggg e_2 \ggg e_3
\end{aligned}
$$

図 5　$st = \mathcal{F}[\![e]\!]$ を満たす構成子項 e の構成例

また $i \in I$ で y'_i と x'_i をそれぞれ (5)(6) で定義する．

$$y'_i = \begin{cases} \text{None} & (i \le d) \\ y_{i-d} & (\text{otherwise}) \end{cases} \tag{5}$$

$$x'_i = \begin{cases} \text{None} & (i \le d') \\ x_{i-d'} & (\text{otherwise}) \end{cases} \tag{6}$$

$st(s) = s' \implies \mathcal{F}[\![e]\!](s) = s'$ を示す．前件から $I' = \{i - d \mid i \in I, i - d \ge 1\}$ および $\mathcal{F}[\![e_1]\!](s) = [\langle y'_i, x_i\rangle \mid i \in I]$ が成り立つ．$\mathcal{F}[\![e_2]\!]([\langle y'_i, x_i\rangle \mid i \in I]) = [\langle y_i, x'_i\rangle \mid i \in I']$ である．$i \le d'$ で x'_i の定義より $x'_i = \text{None}$，$i > d'$ で前件より $x'_i = w'([y_1,\ldots,y_i])$ がそれぞれ成り立つから $\mathcal{F}[\![e_3]\!]([\langle y_i, x'_i\rangle \mid i \in I']) = [y_i \mid i \in I']$ である．よって後件が成り立つ．

$\mathcal{F}[\![e]\!](s) = s' \implies st(s) = s'$ を示す．前件から $I' = \{i - d \mid i \in I, i - d \ge 1\}$ および $i \in I'$ で $y_i = w([x_1,\ldots,x_{i+d}])$ が成り立つ．よって後件が成り立つ．□

系 1. 任意の (d, d')-逐次可逆列変換 st について，st と st' がそれぞれ互いの逆列変換になるような (d', d)-逐次可逆列変換 st' が存在する．

証明. 定理 2 から $st = \mathcal{F}[\![e]\!]$ を満たす構成子項 e が存在する．$st' = \mathcal{B}[\![e]\!]$ とおく．定理 1 から st と st' はそれぞれ互いの逆列変換である．□

$$
\begin{aligned}
&\mathcal{I}[\![\mathit{map\text{-}fold}(a_0, f, g)]\!] = \mathit{map\text{-}fold}(a_0, f', g') \\
&\quad \text{where } f'(a)(y) = f(a)^{-1}(y) \\
&\qquad\quad g'(a)(y) = g(a)(f(a)^{-1}(y)) \\
&\mathcal{I}[\![\mathit{delay}(x_0)]\!] = \mathit{hasten}(x_0) \\
&\mathcal{I}[\![\mathit{hasten}(x_0)]\!] = \mathit{delay}(x_0) \\
&\mathcal{I}[\![e_1 \ggg e_2]\!] = \mathcal{I}[\![e_2]\!] \ggg \mathcal{I}[\![e_1]\!] \\
&\mathcal{I}[\![e_1 \otimes e_2]\!] = \mathcal{I}[\![e_1]\!] \otimes \mathcal{I}[\![e_2]\!]
\end{aligned}
$$

図 6　関数 $\mathcal{I}\colon E(X,Y) \to E(Y,X)$

7　構成子項の反転

任意の構成子項 e について，e に対応する列変換の逆列変換に対応する構成子項を構成できる．関数 $\mathcal{I}$ を図 6 に定義する．

定理 3. 任意の構成子項 e について，$e' = \mathcal{I}[\![e]\!]$ ならば $\mathcal{F}[\![e]\!] = \mathcal{B}[\![e']\!]$ および $\mathcal{B}[\![e]\!] = \mathcal{F}[\![e']\!]$ が成り立つ．

証明. e の構造についての帰納法による．

- （$e = \mathit{map\text{-}fold}(a_0, f, g)$ のとき）$e' = \mathit{map\text{-}fold}(e, f', g')$, $s_x = [x_i \in i \in I]$, $s_y = [y_i \in i \in I]$ とする．$\mathcal{F}[\![e]\!](s_x) = s_y \iff$

$\mathcal{B}[\![e']\!](s_x) = s_y$ を示す．

($\Longrightarrow$) 左辺から $a_i = g(a_{i-1})(x_i)$ とおけば $y_i = f(a_{i-1})(x_i)$ が成り立つ．$x_i = f(a_{i-1})^{-1}(y_i)$ より $a_i = g(a_{i-1})(f(a_{i-1})^{-1}(y_i))$ が成り立つ．よって $y_i = f'(a_{i-1})^{-1}(x_i)$ および $a_i = g'(a_{i-1})(y_i)$ が成り立つ．右辺が示された．

($\Longleftarrow$) 右辺から $y_i = f'(a_{i-1})^{-1}(x_i)$, $a_i = g'(a_{i-1})(y_i)$ を満たす a_i が存在する．よって $y_i = f(a_{i-1})(x_i)$ かつ $a_i = g(a_{i-1})(f(a_{i-1})^{-1}(y_i))$ が成り立つ．$f(a_{i-1})^{-1}(y_i) = x_i$ より $a_i = g(a_{i-1})(x_i)$ が成り立つ．左辺が示された．

よって $\mathcal{F}[\![e]\!] = \mathcal{B}[\![e']\!]$ が成り立つ．$\mathcal{B}[\![e]\!] = \mathcal{F}[\![e']\!]$ も同様にして成り立つ．

- ($e = \mathit{delay}(x_0)$, $e = \mathit{hasten}(x_0)$, $e = e_1 \ggg e_2$, $e = e_1 \otimes e_2$ のとき) $\mathcal{F}[\![\cdot]\!]$ と $\mathcal{B}[\![\cdot]\!]$ の定義より成り立つ．

□

例 10. $y_i = \langle x_i, x_{i+1} \rangle$ で表される 1-逐次列変換に対応する構成子項 e は次のように構成できる．

$$e = \mathit{map\text{-}fold}(1, f, g) \ggg (\mathit{id} \otimes \mathit{hasten}(\mathrm{None}))$$

$$\text{where } f(i)(x) = \begin{cases} \langle x, \mathrm{None} \rangle & (i = 1) \\ \langle x, x \rangle & (i > 1) \end{cases},$$

$$g(i)(x) = i + 1$$

e に対して $\mathcal{I}$ を適用すると

$$\mathcal{I}[\![e]\!] = (\mathit{id} \otimes \mathit{delay}(\mathrm{None})) \ggg \mathit{map\text{-}fold}(1, f', g')$$

$$\text{where } f'(i)(\langle x, x' \rangle) = \begin{cases} x & (i = 1 \wedge x' = \mathrm{None}) \\ x & (i > 1 \wedge x' = x) \\ \bot & (\text{otherwise}) \end{cases}$$

$$g'(i)(y) = i + 1$$

が得られる．

8 関連研究

8.1 可逆計算

計算システムが可逆であるとは，システムの直前の状態が一意に定まることをいう．システムの可逆性は，出力から入力を得られるだけでなく，逆計算を決定的に行えることを主張する．本論文で示した列変換の逆変換は可逆計算とは異なる概念である．

Janus [7]，RFun [12]，Theseus [5] などの可逆プログラミング言語は記述されたプログラムの可逆性を保証する．多くの可逆プログラミング言語は，可逆性を満たす要素を組み合わせて全体の可逆性を成り立たせるという方針で設計されている．本論文で示した (d, d')-逐次可逆列変換の構成法は，逐次的な逆変換が可能であるプリミティブな列変換を合成して全体の性質を保証するものであり，可逆プログラミング言語の設計方針と共通している．

可逆プログラミング言語の構成要素には，ある入力に対してエラーになったり計算が発散したりするものがある．このような構成要素は可逆性の保証のために重要な役割を果たす．例えば Janus の条件分岐の末尾にはアサーションが存在して，満たされない場合にはプログラムが異常終了する．アサーションはプログラムの逆計算では分岐の条件になる．RFun ではパターンマッチ時に逆計算が可能であることを動的に検査して，失敗すればエラーになる．Theseus はパターンマッチの網羅性を静的に保証するが，無限ループによって計算が発散することがある．このようにプログラムの全域性とひきかえに可逆性を保証することは一般的に行われており，本論文でも同様のことを許容した．構成子項 $\mathit{hasten}(x_0)$ で表される列変換は $x_1 = x_0$ を仮定する部分写像である．また構成子項 $\mathit{map\text{-}fold}(a_0, f, g)$ は f に部分関数を許容する．

8.2 部分逆計算

プログラムの入力の一部と出力を既知として残りの入力を復元することを部分逆計算という．例えば関数 $\mathit{add}(x, y) = x + y$ の出力から入力 x, y を復元することはできないが，y を既知とすれば出力から x を復元できる．本論文で示した構成子 *map-fold* の引数 $f\colon A \to (X \rightleftharpoons Y)$ には部分逆計算が可能であることが要求される．

一般の可逆プログラミング言語で部分逆計算をそのまま記述することはできない．可逆計算で部分逆計算をエミュレートするために，既知としたい入力を出力に埋め込む方法が用いられる [11]．例えば関数 add の引数 y を出力に埋め込んだ関数 $\mathit{add}'(x, y) = \langle x + y, y \rangle$

は可逆プログラミング言語で記述できる．ただし実装時に入力と出力の y が等しくなることの静的な保証は得られない．

一部の可逆プログラミング言語は部分逆計算をサポートしている．例えば一階の関数型言語である CoreFun [4] は関数の引数が逆実行時に既知であることを型システムで保証する．また高階関数を扱う言語である Sparcl [8] は可逆な関数と非可逆な関数を型システムで区別する．

可逆計算以外にプログラム逆化を扱う分野としてプログラム逆化が挙げられる．プログラム逆化とは，与えられたプログラムに対してその逆計算を行うプログラムを得ることである．例えば項書き換え系に対して部分逆計算を行う系を得る手法が研究されている [9] [1]．

8.3 同期的データフロープログラミング

同期的データフロープログラミングとは，リアクティブシステムを記述するパラダイムである．同期的データフロープログラミングのための言語として Lustre [2] や Signal [6] が挙げられる．これらの言語では，入出力を時間とともに変化する値の列として表現して，それらの関係を宣言的に記述することによってシステムを記述する．入力から出力を得るプロセスは決定的である．つまり，同期的データフロープログラミングで記述されたシステムは列から列への変換とみなせる．本論文で定義した d-逐次列変換（とくに $d = 0$ の場合）は制限されたリアクティブシステムのモデル化であり，その逆変換を扱う際に $d > 0$ の d-逐次列変換が現れることがある．

また，関数リアクティブプログラミングは同期的データフロープログラミングに近いパラダイムである．言語やフレームワークにもよるが，例えばプログラミング言語 Emfrp [10] で記述されたリアクティブシステムは列から列への変換としてモデル化できる．

リアクティブシステムにはタイムトラベルデバッグ (TTD) というデバッグ手法が存在するが，これは本論文で示した列変換の逆変換とは異なる概念である．TTD ではシステムの実行を記録して任意の時点での再生や逆再生を行う．TTD は出力から入力を求めるものではないため逆計算には含まれないが，システムの過去の状態を遡るという点では可逆計算と共通している．本論文で示した列変換の逆計算は出力から入力を求めるが，時間軸を遡ることはせず，むしろ順方向にシステムの状態を再現する．

8.4 双方向変換

双方向変換とは，データ（ソース）の一部または全部を抽出する順方向の変換と，変換後のデータ（ビュー）上の更新をソースに書き戻す逆方向の変換からなる計算の枠組みである．双方向変換の応用例として，扱うデータ形式が異なるソフトウェア同士でのカレンダーやブックマークの同期が挙げられる．

逆計算ではプログラムの出力のみから入力を得るのに対して，双方向変換ではソース（順方向の変換から見た入力）とビュー（順方向の変換から見た出力を更新したもの）から新しいソースを得る．ビューからソースを完全に復元する必要がない点やビューの更新をソースに書き戻すことを重視している点で，双方向計算は逆計算と異なる．例えば，Web ページのブックマークのうち，その一部分を異なるブラウザ間で共有することが双方向変換の枠組みでは可能である．本論文で扱ったのは「列の変換」に対する逆計算であり，双方向変換ではない．

Lens [3] という双方向変換を記述する手法がある．Lens では基本的な双方向変換をレンズコンビネータで組み合わせることで，双方向変換であることを維持しながらより大きな変換を構成していく．本論文の手法と Lens は，基本的な変換を組み合わせて性質を保ちながらより大きな変換を構成するという点で共通している．

9 まとめと今後の課題

本論文ではまず同期的データフロープログラムの逐次的な逆計算について議論して，(d, d')-逐次可逆列変換の定義を与えた．つぎに，部分可逆関数を用いた基本的な列変換を組み合わせることで，逐次的な逆計算の可能性を維持したまま任意の (d, d')-逐次可逆列変換を表現できることを示した．

これらの議論は，同期的プログラムの逐次的な逆計

算を既存研究の逆計算に帰着すると同時に，逐次的な逆計算を保証するプログラミング言語の設計の指針になる．そのような言語の設計は今後の課題である．

また，逐次的な逆計算の定義をより一般的なものに拡張することも今後の課題である．d-逐次列変換には入力列と出力列の長さの差が定数 d になるという制約が課せられている．制約の緩和された列変換に対して逆変換を保証する構成法が存在すれば，より多くの種類のプログラムに対する手法の適用が可能になる．

謝辞 本研究の一部は JSPS 科研費 21K11822 および 22K11967 の助成を受けている．

参 考 文 献

[1] Almendros-Jiménez, J. M. and Vidal, G.: Automatic Partial Inversion of Inductively Sequential Functions, *Implementation and Application of Functional Languages*, Horváth, Z., Zsók, V., and Butterfield, A.(eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, 2007, pp. 253–270.

[2] Caspi, P., Pilaud, D., Halbwachs, N., and Plaice, J. A.: LUSTRE: A Declarative Language for Real-Time Programming, *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, New York, NY, USA, Association for Computing Machinery, 1987, pp. 178–188.

[3] Foster, J. N., Greenwald, M. B., Moore, J. T., Pierce, B. C., and Schmitt, A.: Combinators for Bidirectional Tree Transformations: A Linguistic Approach to the View-Update Problem, *ACM Trans. Program. Lang. Syst.*, Vol. 29, No. 3(2007), pp. 17–es.

[4] Jacobsen, P. A. H., Kaarsgaard, R., and Thomsen, M. K.: CoreFun: A Typed Functional Reversible Core Language, *Reversible Computation*, Kari, J. and Ulidowski, I.(eds.), Cham, Springer International Publishing, 2018, pp. 304–321.

[5] James, R. P.: Theseus : A High Level Language for Reversible Computing, 2014.

[6] LeGuernic, P., Gautier, T., Le Borgne, M., and Le Maire, C.: Programming real-time applications with SIGNAL, *Proceedings of the IEEE*, Vol. 79, No. 9(1991), pp. 1321–1336.

[7] Lutz, C.: Janus: a time-reversible language, 1986. *Letter to R. Landauer.*

[8] Matsuda, K. and Wang, M.: Sparcl: A Language for Partially-Invertible Computation, *Proc. ACM Program. Lang.*, Vol. 4, No. ICFP(2020).

[9] Nishida, N., Sakai, M., and Sakabe, T.: Partial Inversion of Constructor Term Rewriting Systems, *Term Rewriting and Applications*, Giesl, J.(ed.), Berlin, Heidelberg, Springer Berlin Heidelberg, 2005, pp. 264–278.

[10] Sawada, K. and Watanabe, T.: Emfrp: A Functional Reactive Programming Language for Small-Scale Embedded Systems, *Companion Proceedings of the 15th International Conference on Modularity*, MODULARITY Companion 2016, New York, NY, USA, Association for Computing Machinery, 2016, pp. 36–44.

[11] Thomsen, M. K. and Axelsen, H. B.: Interpretation and Programming of the Reversible Functional Language RFUN, *Proceedings of the 27th Symposium on the Implementation and Application of Functional Programming Languages*, IFL '15, New York, NY, USA, Association for Computing Machinery, 2015.

[12] Yokoyama, T., Axelsen, H. B., and Glück, R.: Towards a Reversible Functional Language, *Reversible Computation*, De Vos, A. and Wille, R.(eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, 2012, pp. 14–29.