ジグソーパズルによる関数型プログラミング

中　野　圭　介†

関数型プログラミングにおいて，リストを消費する関数 *foldr* および生成する関数 *unfoldr* は重要な役割を果たす．特に，入力を *foldr* によって消費し，出力を *unfoldr* によって生成するような関数はメタモルフィズムとよばれ，基数変換などに代表される或るデータ表現から別のデータ表現への変換を自然に定義することが可能である．Bird と Gibbons は，このメタモルフィズムに対し，入力を消費しながら出力を生成するようなストリーム処理として実装できるための条件を提示したが，自然数に対する基数変換には応用することができなかった．これは入力の桁を全て読み込まない限り，出力の桁を決定できないためで，彼らの提示した条件を満たしていない．これを解決するために著者が提案した手法が，ジグソーパズルを利用したメタモルフィズムの計算機構である．本稿では，グレイコードのような特別な記数法に対してこの機構を適用し，対応するジグソーモデルの導出方法を示す．

Functional Programming in Jigsaw

Keisuke Nakano†

The *foldr* and *unfoldr* functions play important roles in functional programming: *foldr* consumes a list and produces a value; *unfoldr* generates a list from a value. In particular, we call a metamorphism for a function specified by an unfold after a fold, consuming an input by the fold then generating an output by the unfold. It naturally specifies a conversion over data representations such as radix conversion of numbers. Bird and Gibbons have shown that metamorphisms can be incrementally processed in streaming style when a certain condition holds because part of the output can be determined before the whole input is given. However, radix conversion of natural numbers cannot satisfy the condition because it is impossible to determine part of the output before the whole input is completed. To solve the problem, the author has introduced a jigsaw model in which metamorphisms can be partially processed for outputs even when the streaming condition does not hold. In this paper, we illustrate that the jigsaw method can be applied to a special numeral system called Gray codes. It is also shown how we obtain the jigsaw model in a systematic way.

1. はじめに

関数型プログラミングにおいて，リストを消費する関数 *foldr* とリストを生成する関数 *unfoldr* は重要な役割を果たしている．特に，*foldr* によって入力リストを消費し，その値を利用して *unfoldr* によって出力リストを生成する関数は**メタモルフィズム**とよばれ，リストを用いたデータ表現間の変換を自然に定義することができる．たとえば，m 進表現から n 進表現への基数変換は典型的なメタモルフィズムの例であり，m 進表現 (リスト) から対応する値を計算する関数を *foldr* で定義し，その値から n 進表現 (リスト) を導く関数を *unfoldr* で定義することで実現できる．

Bird と Gibbons[1),2)] はメタモルフィズムに対する効率的な実装方法として，ストリーム化とよばれる手法を提案している．メタモルフィズムを安直に実装する場合，*foldr* に対する入力を最後まで読み込まない限り，その結果の値を *unfoldr* に渡すことができないため，一般的には実行効率が悪くなってしまう．ストリーム化は，この問題を解決するために提案されたもので，メタモルフィズムで与えられた関数を入力を消費しながら出力を生成するような**ストリーム関数**に変換する手法である．Gibbons は，メタモルフィズムで表現された連分数変換や無限小数間の基数変換などに対し，ストリーム化が有効であることを示している[2)]．

しかしながら，メタモルフィズムで与えられた全ての関数がストリーム化できるわけではない．たとえば，自然数に対する基数変換はメタモルフィズムで表現できるが，入力を消費しながら出力を生成するストリーム関数で実現することはできない．これは，自然数に対する基数変換では，入力に対応する完全な値が分か

† 電気通信大学
The University of Electro-Communications

らない限り，最下位の桁が出力できないためである．Bird と Gibbons[1] は**ストリーム化条件**を提示し，それを満たすメタモルフィズムに限り，ストリーム関数を導くことが可能であることを示している．実際，自然数に対する基数変換はストリーム化条件を満たさないため，彼らの手法の対象外である．

著者は，より広い範囲のメタモルフィズムに対する実装方法として，ジグソーパズルを解く要領で計算を実行するモデルを提案した[5]．このジグソーモデルでは，自然数に対する基数変換やヒープソートなどストリーム化条件を満たさないメタモルフィズムを実装することが可能であり，入力しながら出力を計算する**ジグソー関数**によって実現される．また，このモデルでは，ジグソーパズルのように順序に捉われることなくピースを置くことで計算が進められるため，並列計算などへの応用も期待される．文献 5) では，**ジグソー化条件**とよばれる条件を提示し，これを満たすメタモルフィズムに限りジグソーモデルによって実現できることを示したが，紙面の都合上☆，多くの例やその導出方法について言及することができなかった．

本稿では，文献 5) で紹介できなかったメタモルフィズムの例について，対応するジグソーモデルを示す．具体的には n 進表現から 2 進グレイコードへの変換やその逆変換についてジグソーモデルの導出を行う．この例はメタモルフィズムで自然に定義ができるが，ジグソー化条件は満たすかは自明ではない．本研究では，与えられたメタモルフィズムを少し書き換えることでジグソー化条件を成り立たせることができることを示し，対応するジグソーモデルの導出に成功した．

本稿におけるプログラムの記述には Haskell 風の構文を利用する．ただし，記述を簡潔にするため，商と剰余を求める演算子として / および % を用いるものとする．これらは Haskell における `` `div` `` および `` `mod` `` に対応する．すなわち，$p/q = \lfloor \frac{p}{q} \rfloor$ および $p\%q = p - q\lfloor \frac{p}{q} \rfloor$ と定義する．

2. ジグソーモデルによる基数変換

本節では，ジグソーモデルにおける計算の仕組みを理解するために，3 進表現から 2 進表現への基数変換の具体例を見ながら説明する．まず，ジグソーモデルによる基数変換の手続きを示した後，その仕組みと特徴について解説する．

☆ 文献 5) は Functional Pearl として採録されているためページ数に制限がある．

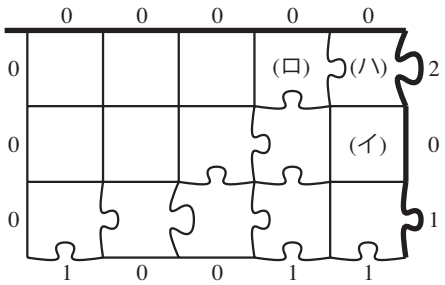
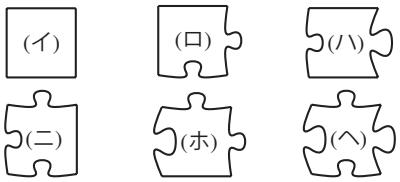


図 1 3 進表現 (201) から 2 進表現 (10011) への変換

2.1 ジグソーモデルによる 3 進 2 進変換

通常の 3 進 2 進変換では，入力の 3 進表現に対応する値を和と積により計算し，その値の 2 進表現を商と剰余により計算を進めるが，ジグソーモデルによる基数変換では四則演算を一切用いない．3 進 2 進変換を実現するジグソーモデルでは以下の 6 つのピースを用いる:



ここで，各ピースの辺には ——, ∿∿, ∿∿ の 3 種類しか現れないことに注意されたい．これらの線はそれぞれ数値の 0, 1, 2 に対応する．たとえば，ピース (ホ) は，右と上の辺が 1，左の辺が 2，下の辺が 0 を表している．3 進表現から 2 進表現への変換はこれらのピースを以下のルールに従ってボードに配置することで実現される:

- 各ピースとも何度配置してもよい．
- どのピースも回転・反転してはならない．
- ボードの上端は直線 (0) である．
- ボードの右端は 3 進表現に対応する曲線である．
- 左端に直線が現れるまでピースを配置する．

この手続きにより，目的の 2 進表現がボードの下端に曲線として現れる．

図 1 は，ジグソーモデルを用いて 3 進表現の 201 を 2 進表現へ変換した手続きを表している．太線で描かれたボードの部分から始めて，まず右上の角に置けるピース (ハ) を配置する．続いてそのピースの下や左に置けるピース (イ) と (ロ) をそれぞれ配置する．このように置けるピースを配置していき，左端に直線が現れた時点で下端に現れた曲線 10011 が対応する 2 進表現となる．

2.2 ジグソーモデルの仕組み

ジグソーモデルによる 3 進 2 進変換の絡繰りは，次のようにして理解できる．まず，整数上の 2 項演算子 ▼ と ► (いずれも左結合) を導入し，それぞれ $x \blacktriangledown y = 3x + y$，$x \blacktriangleright y = 2x + y$ を満たすものとする．この演算子を用いれば，3 進表現の 201 や 2 進表現の 10011 が 19 という値を表すことは $2 \blacktriangledown 0 \blacktriangledown 1 = 1 \blacktriangleright 0 \blacktriangleright 0 \blacktriangleright 1 \blacktriangleright 1 = 19$ によって確認できる．ジグソーモデルによる基数変換は，$2 \blacktriangledown 0 \blacktriangledown 1$ から $1 \blacktriangleright 0 \blacktriangleright 0 \blacktriangleright 1 \blacktriangleright 1$ への変換を記号的に行う手続きを表している．ジグソーモデルによる計算において，ピースが置かれている部分と置かれていない部分の境界の推移に着目すると，「ピースを置く」操作が「境界の一部を書き換える」操作に対応することが分かる．たとえば，何もピースを置いてない状態における境界は図 1 の太線に相当し，左から水平方向に 0，0，0，0，0，そこから下に 2，0，1 と列んでいる．この境界を $0 \blacktriangleright 0 \blacktriangleright 0 \blacktriangleright 0 \blacktriangleright 0 \blacktriangledown 2 \blacktriangledown 0 \blacktriangledown 1$ と表すことにすると，ピース (ハ) を置くという操作により，$0 \blacktriangleright 0 \blacktriangleright 0 \blacktriangleright 0 \blacktriangledown 1 \blacktriangleright 0 \blacktriangledown 0 \blacktriangledown 1$ という境界へ推移する．一般に，ピースの上下左右の辺がそれぞれ T，B，L，R で表されるとき，このピースを置くという操作は，境界 $\ldots \blacktriangleright T \blacktriangledown R \ldots$ が $\ldots \blacktriangledown L \blacktriangleright B \ldots$ に推移することに対応する．ジグソーモデルによる 3 進 2 進変換の肝は，6 種類全てのピースが任意の整数 x に対して $x \blacktriangleright T \blacktriangledown R = x \blacktriangledown L \blacktriangleright B$ を満たすことである．たとえば，ピース (ホ) は $x \blacktriangleright 1 \blacktriangledown 1 = x \blacktriangledown 2 \blacktriangleright 0 = 6x+4$ となっている．この等式は境界の推移により境界の「値」が変化しないことを表しており，ピースを置く前の境界と置き終わった後の境界とで値が等しいことが保証される．すなわち，

$$\begin{aligned} &0 \blacktriangleright 0 \blacktriangleright 0 \blacktriangleright 0 \blacktriangleright 0 \blacktriangledown 2 \blacktriangledown 0 \blacktriangledown 1 \\ &\quad = 0 \blacktriangledown 0 \blacktriangledown 0 \blacktriangledown 1 \blacktriangleright 0 \blacktriangleright 0 \blacktriangleright 1 \blacktriangleright 1 \end{aligned}$$

が成り立つ．定義から得られる自明な等式 $0 \blacktriangleright x = x$ および $0 \blacktriangledown x = x$ を用いれば，$2 \blacktriangledown 0 \blacktriangledown 1 = 1 \blacktriangleright 0 \blacktriangleright 0 \blacktriangleright 1 \blacktriangleright 1$ が成り立つことが容易に確認できる．

上の例では，上端に 5 つの 0 が列んだボードから始めたが，一般にはいくつの 0 が必要かは予測できないため，実際の手続きでは必要に応じて境界の先頭に $(0 \blacktriangleright)$ を補うことが求められる．境界の推移させる手続きは，(a) ► が ▼ より左に現れず，(b) ▼ の左に現れる数値は全て 0 であるときに終了する．条件 (a) はピースを置くところが無くなったことに相当し，(b) は左端に直線が現れていることに相当する．基数変換に関していえば，この手続きは必ず停止する．これは，境界の推移にするたびに各 ▼ の左の非演算子の値の合計が必ず減少することによる．また，境界の先頭に現れた $(0 \blacktriangledown)$ は恒等関数であり，手続きの途中でこれを取り除いても結果に影響を及ぼすことはない．

なお，ここで紹介した基数変換の手続きは m 進表現から n 進表現への変換に容易に一般化可能である．m 進 n 進変換では，mn 種類のピースを使って同様の手続きを行う．

2.3 ジグソーモデルの特徴

ここでジグソーモデルにおける計算の特徴についていくつか言及しておく．ここまで紹介した 3 進 2 進変換は飽くまでジグソーモデルによる計算の一例であり，通常の計算よりも効率よく基数変換が行えるということを主張するものではない．たとえば，計算複雑性に着目すると，入出力の桁数を m, n とするとき，単純な計算では $O(m)$ 回の和と積および $O(n)$ 回の商と剰余だけで計算できるのに対し，ジグソーモデルにおいては $O(mn)$ 回のピースを置く手続き (パターンマッチ) が必要であると言えるが，それぞれ計算の単位が異なるためこの比較は意味をなさない．本研究の主張は，ジグソー化条件を満たすメタモルフィズムはジグソーモデルで計算を表現することができ，それによりジグソーモデルの特徴に関連する恩恵が得られるということである．

ジグソーモデルで計算を表現することの大きな利点は，順序に依存せずに計算を進めることができるという点である．3 進 2 進変換のジグソーモデルで用いた 6 種類のピースは上と右の辺の組が互いに異なるため，ピースを置く順序によらず結果は一意に定まる．したがって，右の列から順に配置したり，上の行から順に配置したり，不規則な順序で配置したりしても同じ計算結果が得られる．また，同時に複数箇所にピースを置くことも許されるため並列に実行することが可能であり，これはウェーブフロント計算の一種と見なすこともできる．

3. ジグソー関数とジグソー化

本節では，一般的なジグソーモデルによる計算を表現するジグソー関数を紹介する．また，メタモルフィズムとして与えられた計算がジグソーモデルにより表現できるため条件を提示し，メタモルフィズムをジグソー化する手法を示す．

3.1 ジグソー関数

ジグソーモデルによる計算は，リストからリストへの変換であるため，$[\tau_{\mathrm{V}}] \to [\tau_{\mathrm{H}}]$ 型の関数で表現できる．ここで，型 $[\tau]$ は 型 τ の要素から成るリストの型を表し，τ_{V} や τ_{H} はそれぞれ垂直方向，水平方向に

現れる値の型である．3 進 2 進変換の例では，型 τ_{V} および τ_{H} はそれぞれ次で定義される型 Trit および Bit である:

$\mathbf{data}\ \mathsf{Trit} = \mathsf{T}_0 \mid \mathsf{T}_1 \mid \mathsf{T}_2$ -- 3 進表現の 0, 1, 2
$\mathbf{data}\ \mathsf{Bit} = \mathsf{B}_0 \mid \mathsf{B}_1$ -- 2 進表現の 0, 1

なお，メタモルフィズムとの自然な関係の記述のため，入力となる垂直方向の値のリストは 下から，出力となる水平方向の値のリストは 右から 列べたものであるとする．すなわち，図 1 において，関数の入力は $[\mathsf{T}_1, \mathsf{T}_0, \mathsf{T}_2]$ であり，出力は $[\mathsf{B}_1, \mathsf{B}_1, \mathsf{B}_0, \mathsf{B}_0, \mathsf{B}_1]$ である．以下では，ジグソーモデルによる計算を特徴づける 3 つの要素，ピースの集合，ボードの上端の値，終了条件となる左端の値を順に紹介する．

(1) ピースの集合 (関数 *pieces*)

ジグソーモデルによる計算において最も重要な役割を果たすのがピースの種類である．前節で見たように，各ピースの右と上の辺の組は互いに異なるときに計算結果が一意になり，実行結果がピースを置く順序に依存しないというジグソーモデルの恩恵が得られる．そこで，ピースの集合を右と上の辺から下と左の辺への関数 *pieces* として表現し，型 $(\tau_{\mathrm{V}}, \tau_{\mathrm{H}}) \to (\tau_{\mathrm{H}}, \tau_{\mathrm{V}})$ をもつものとする．たとえば，3 進 2 進変換を行うジグソーモデルの例に対しては，関数 *pieces* は以下で定義される $pieces_{3,2}$ である:

$$
\begin{aligned}
&pieces_{3,2} :: (\mathsf{Trit}, \mathsf{Bit}) \to (\mathsf{Bit}, \mathsf{Trit})\\
&pieces_{3,2}\ (\mathsf{T}_0, \mathsf{B}_0) = (\mathsf{B}_0, \mathsf{T}_0)\\
&pieces_{3,2}\ (\mathsf{T}_1, \mathsf{B}_0) = (\mathsf{B}_1, \mathsf{T}_0)\\
&pieces_{3,2}\ (\mathsf{T}_2, \mathsf{B}_0) = (\mathsf{B}_0, \mathsf{T}_1)\\
&pieces_{3,2}\ (\mathsf{T}_0, \mathsf{B}_1) = (\mathsf{B}_1, \mathsf{T}_1)\\
&pieces_{3,2}\ (\mathsf{T}_1, \mathsf{B}_1) = (\mathsf{B}_0, \mathsf{T}_2)\\
&pieces_{3,2}\ (\mathsf{T}_2, \mathsf{B}_1) = (\mathsf{B}_1, \mathsf{T}_2)
\end{aligned}
$$

(2) ボードの上端に相当する値 h_0

ジグソーモデルにおける計算の振舞いを定める 2 つめの要素は，ボードの上端に現れる値の列である．ジグソー関数では，簡単のため一種類の値が列んでいるものとする．すなわち，型 τ_{H} の値で表現できる．3 進 2 進変換の例では $\mathsf{B}_0 :: \mathsf{Bit}$ がこれに相当する．

(3) 手続きの終了条件となる左端の値 v_0

ジグソーモデルにおける計算は左端に現れる線によって，手続きの終了を判定している．ジグソー関数では，簡単のため左端の全ての値が特定の値になったら終了という条件で計算を定義する．すなわち，型 τ_{V} の値で表現できる．3 進 2 進変換の例では，$\mathsf{T}_0 :: \mathsf{Trit}$ がこれに相当する．

ジグソー関数 *jigsaw* はこれら 3 つの要素によって特徴づけられることから

$$((\tau_{\mathrm{V}}, \tau_{\mathrm{H}}) \to (\tau_{\mathrm{H}}, \tau_{\mathrm{V}})) \to \tau_{\mathrm{H}} \to \tau_{\mathrm{V}} \to [\tau_{\mathrm{V}}] \to [\tau_{\mathrm{H}}]$$

という型をもつ関数で，ピースの集合を表す関数，上端の値，左端の値を受け取り，垂直方向の値のリスト (下が先頭) を水平方向の値のリスト (右が先頭) へ変換する関数を返す．関数 *jigsaw* は，前節で示した 3 進 2 進変換を表すジグソーモデルによる手続きと同様に，境界の推移を繰り返し実行する形で実装される．

関数 *jigsaw* の定義を図 2 に示した．推移させる境界は型 τ_{V} と τ_{H} の値が混合したリストで表現されるため，それらに Vert，Horiz というタグをつけた直和型 Edge を定義しており，境界の型は [Edge] である．また，メタモルフィズムとの自然な関係のため，境界は辺の値を 右下から左上へ 列べたリストとして与えられる．関数 *jigsaw* は，上述の 3 つの要素に相当する *pieces*，h_0，v_0 を引数に取り，垂直方向のリストから水平方向のリストへ変換する関数を返しており，その変換は 3 つの関数，(a) *map unHoriz*，(b) *until* (*all isHoriz*) *step*，(c) *map* Vert の結合によって表現されている．まず，入力のリストは (c) の関数により境界の型 [Edge] に変換される．(b) は，終了条件 (*all isHoriz*) を満たすまで境界を繰り返し推移させる関数で，1 回分の境界の推移は関数 *step* によって表されている．関数 *step* は特殊なパターンマッチ $xs +\!\!+ [\mathsf{Vert}\ v, \mathsf{Horiz}\ h] +\!\!+ ys$ を用いて定義されているが，これはジグソーモデルの特徴であるピースを置く順序の非決定性に相当する．また，関数 *step* では，ピースを置くことによる境界の推移だけでなく，必要に応じて上端の Horiz h_0 を末尾に追加したり，終了状態となる左端の Vert v_0 を末尾から削除したりする役割も果たす．2 節の説明では終了条件を「左端が全て 0 になった時点」としていたが，ここでは関数 *step* が左端の 0 を除去するように定義されているため，「全ての辺が水平になった時点 (*all isHoriz*)」としている．最後の (a) により，境界の型を出力のリストに変換する．

3 進 2 進変換は，でジグソー関数 *jigsaw* を用いて

$$
\begin{aligned}
&radixConv_{3,2} :: [\mathsf{Trit}] \to [\mathsf{Bit}]\\
&radixConv_{3,2} = jigsaw\ pieces_{3,2}\ \mathsf{B}_0\ \mathsf{T}_0
\end{aligned}
$$

で定義することができる．なお，入出力のリストは最下位桁を先頭とする 3 進または 2 進表現である．この定義の正当性については 3.3 節で示す．

3.2 メタモルフィズム

メタモルフィズムは，リストを消費する *foldr* によって定義された関数とリストを生成する *unfoldr* によって定義された関数を結合することで与えられる．関数

$$
\begin{array}{l}
\textbf{data}\ \mathsf{Edge}\ =\ \mathsf{Vert}\ \tau_{\mathrm{V}}\ \mid\ \mathsf{Horiz}\ \tau_{\mathrm{H}} \\
\mathit{jigsaw} :: ((\tau_{\mathrm{V}}, \tau_{\mathrm{H}}) \to (\tau_{\mathrm{H}}, \tau_{\mathrm{V}})) \to \tau_{\mathrm{H}} \to \tau_{\mathrm{V}} \to [\tau_{\mathrm{V}}] \to [\tau_{\mathrm{H}}] \\
\mathit{jigsaw}\ \mathit{pieces}\ h_0\ v_0\ =\ \mathit{map}\ \mathit{unHoriz}\ \cdot\ \mathit{until}\ (\mathit{all}\ \mathit{isHoriz})\ \mathit{step}\ \cdot\ \mathit{map}\ \mathsf{Vert} \\
\quad \textbf{where}\quad \mathit{unHoriz}\ (\mathsf{Horiz}\ x) \quad = x \\
\qquad\qquad \mathit{isHoriz}\ (\mathsf{Horiz}\ _) \quad = \mathsf{True} \\
\qquad\qquad \mathit{isHoriz}\ (\mathsf{Vert}\ _) \quad = \mathsf{False} \\
\qquad\qquad \mathit{step}\ (xs +\!\!+ [\mathsf{Vert}\ v, \mathsf{Horiz}\ h] +\!\!+ ys)\ =\ xs +\!\!+ [\mathsf{Horiz}\ h', \mathsf{Vert}\ v'] +\!\!+ ys \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{where}\ (h', v') = \mathit{pieces}\ (v, h) \\
\qquad\qquad \mathit{step}\ (xs +\!\!+ [\mathsf{Vert}\ v]) \mid v = v_0 \quad = xs \\
\qquad\qquad\qquad\qquad\qquad\quad \mid v \neq v_0 \quad = xs +\!\!+ [\mathsf{Vert}\ v, \mathsf{Horiz}\ h_0].
\end{array}
$$

図 2 ジグソー関数 *jigsaw* の定義

foldr と *unfoldr* は，それぞれ再帰的に

$$
\begin{array}{l}
\mathit{foldr} :: ((\tau_{\mathrm{V}}, a) \to a) \to a \to [\tau_{\mathrm{V}}] \to a \\
\mathit{foldr}\ g\ e\ [\,] \quad\ = e \\
\mathit{foldr}\ g\ e\ (x : xs)\ =\ g\ (x,\ \mathit{foldr}\ g\ e\ xs) \\
\mathit{unfoldr} :: (a \to \mathsf{Maybe}(\tau_{\mathrm{H}}, a)) \to a \to [\tau_{\mathrm{H}}] \\
\mathit{unfoldr}\ f\ s\ =\ \textbf{case}\ f\ s\ \textbf{of} \\
\qquad\qquad\qquad \mathsf{Nothing} \quad \to [\,] \\
\qquad\qquad\qquad \mathsf{Just}(x, s') \ \to\ x : \mathit{unfoldr}\ f\ s'
\end{array}
$$

のように定義される．通常の *foldr* や *unfoldr* は多相型として定義されるが，ジグソーモデルと関連づけるため，型 τ_{V} と 型 τ_{H} を用いて形式化している．また，ジグソー化の関係で *foldr* の第 1 引数はアンカリー化された関数となっており標準の *foldr* とは異なるが，本質的な差はない．メタモルフィズムは，これらの関数に加え，関数 $f :: a \to \mathsf{Maybe}(\tau_{\mathrm{H}}, a)$，関数 $g :: (\tau_{\mathrm{V}}, a) \to a$，値 $e :: a$ を用いて

$$\mathit{unfoldr}\ f \cdot \mathit{foldr}\ g\ e$$

によって表される．本論文では，このメタモルフィズムを $\{\!|f, g, e|\!\}$ の 3 つ組で表現する．

一般に m 進表現から n 進表現への変換は，メタモルフィズムにより表現することが可能である．m 進表現から対応する値を計算する部分を *foldr* で，得られた値から n 進表現を生成する部分を *unfoldr* で表現することで，以下のようにメタモルフィズム $\{\!|\mathit{modDiv}_n, \mathit{sumMul}_m, 0|\!\}$ として定義できる:

$$
\begin{array}{l}
\textbf{data}\ \mathsf{DigitM}\ =\ \mathsf{M}_0\ \mid\ \ldots\ \mid \mathsf{M}_{m-1} \\
\textbf{data}\ \mathsf{DigitN}\ =\ \mathsf{N}_0\ \mid\ \ldots\ \mid \mathsf{N}_{n-1} \\
\mathit{radixConv}_{m,n}\ ::\ [\mathsf{DigitM}] \to [\mathsf{DigitN}] \\
\mathit{radixConv}_{m,n}\ = \\
\qquad \mathit{unfoldr}\ \mathit{modDiv}_n\ \cdot\ \mathit{foldr}\ \mathit{sumMul}_m\ 0 \\
\quad \textbf{where}\ \mathit{sumMul}_m :: (\mathsf{DigitM}, \mathit{Int}) \to \mathit{Int} \\
\qquad\qquad \mathit{sumMul}_m\ (\mathsf{M}_i, s)\ =\ i + s \times m \\
\qquad\qquad \mathit{modDiv}_n :: \mathit{Int} \to \mathsf{Maybe}(\mathsf{DigitN}, \mathit{Int}) \\
\qquad\qquad \mathit{modDiv}_n\ s \mid s = 0\ =\ \mathsf{Nothing} \\
\qquad\qquad\qquad\qquad\quad \mid s \neq 0\ =\ \mathsf{Just}(\mathsf{N}_{s\%n}, s/n)
\end{array}
$$

3.3 ジグソー化条件とジグソー化

メタモルフィズム $\{\!|f, g, e|\!\}$ が或る条件を満たすとき，ジグソー関数で実装することが可能である．この条件を**ジグソー化条件**とよび，以下のように定義される．

定義 3.1 (ジグソー化条件[5]) メタモルフィズム $\{\!|f, g, e|\!\}$ に対し，以下の (1) から (3) を満たすような関数 $\mathit{pieces} :: (\tau_{\mathrm{V}}, \tau_{\mathrm{H}}) \to (\tau_{\mathrm{H}}, \tau_{\mathrm{V}})$，値 $h_0 :: \tau_{\mathrm{H}}$ および $v_0 :: \tau_{\mathrm{V}}$ が存在するとき，そのメタモルフィズムがジグソー化条件を満たすという:

(1) $f\ e = \mathsf{Nothing}$ が成り立つ．

(2) 任意の $(v, s) :: (\tau_{\mathrm{V}}, a)$ について，$f\ (g\ (v, s)) = \mathsf{Nothing}$ であるための必要十分条件は，$v = v_0$ かつ $f\ s = \mathsf{Nothing}$ である．

(3) 任意の $(v, s) :: (\tau_{\mathrm{V}}, a)$ について，$(h, s') = f^{\#}\ s$ かつ $(h', v') = \mathit{pieces}\ (v, h)$ ならば，$f^{\#}(g(v, s)) = (h', g(v', s'))$ が成り立つ．ここで，$f^{\#}$ は以下で定義される関数とする:

$$
\begin{array}{l}
f^{\#} :: a \to (\tau_{\mathrm{H}}, a) \\
f^{\#}\ s\ =\ \textbf{case}\ f\ s\ \textbf{of}\quad \mathsf{Nothing} \quad \to (h_0, s) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{Just}(h, s') \ \to\ (h, s').
\end{array}
$$

□

ジグソー化条件はやや煩雑ではあるが，核となる条件は (3) であり，この条件を視覚化すると 図 3 のようになる．これは「*pieces* を挟むことで f と g の順序を交換できる」ことを表しており，メタモルフィズムにおいて $\mathit{foldr}\ g\ e$ による計算が終わる前に $\mathit{unfoldr}\ f$ による計算が開始できるようにするための仕組みであると理解できる．詳細については文献 5) を参照されたい．

また，以下の定理はジグソー化条件を満たす全てのメタモルフィズムがジグソー関数によって再定義可能であることを表している．

定理 3.2 (ジグソー化定理[5]) メタモルフィズム $\{\!|f, g, e|\!\}$ が *pieces*，h_0 および v_0 によってジグソー化

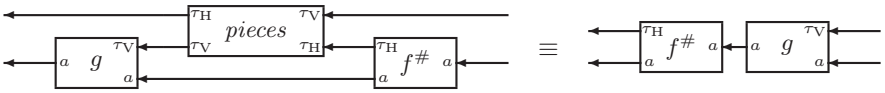


図 3 ジグソー化条件 (3)

条件を満たすとき,

$$jigsaw\ pieces\ h_0\ v_0 = unfoldr\ f\ \cdot foldr\ g\ e.$$

が成り立つ. □

3.2 節で定義した m 進 n 進変換 $radixConv_{m,n}$ を表すメタモルフィズム $\{\!|modDiv_n, sumMul_m, 0|\!\}$ は，以下で定義される関数 $pieces_{m,n}$ と M_0，N_0 によりジグソー化条件を満たすことが容易に確認できる:

$$pieces_{m,n}\ (\mathsf{M}_i, \mathsf{N}_j) = (\mathsf{N}_{(i+j\times m)\%n}, \mathsf{M}_{(i+j\times m)/n})$$

したがって，定理 3.2 より，

$$radixConv_{m,n} = jigsaw\ pieces_{m,n}\ \mathsf{M}_0\ \mathsf{N}_0$$

が成り立つ．$(m,n) = (3,2)$ の場合を考えると，簡単な名前変更 $(\mathsf{M}_i \mapsto \mathsf{T}_i; \mathsf{N}_j \mapsto \mathsf{B}_j)$ により 3.1 節で定義した 3 進 2 進変換の正当性を示すことができる．

4. ジグソー化の応用例

一般に，メタモルフィズム $\{\!|f, g, e|\!\}$ に対し，ジグソー化条件を満たすための $pieces$，v_0，h_0 を見つけることは簡単ではない．文献 5) では，ジグソー化条件を満たすメタモルフィズムとして，m 進 n 進変換，グループ化 (GroupBy) 処理，ヒープソートを例として挙げているが，いずれもジグソー化条件 (3) を満たすような関数 $pieces$ を機械的に特定することは難しい．本稿では，別のメタモルフィズムの例として，グレイコードと他の記数法との相互変換を取り上げ，そのジグソー化について考察し，関数 $pieces$ を特定する具体的な手順についても述べる．

4.1 グレイコード

グレイコードは交番符号ともよばれ，F. Gray が 1947 年に開発した記数法である[3]．通常の記数法は位が上がる際に複数の桁を更新する必要があるが，グレイコードでは全てのインクリメント (デクリメント) 操作において 1 つの桁しか変更されないという特徴をもつため，地上デジタル放送などのデータ通信の誤り訂正にも応用されている．グレイコードの詳細な定義については，本論文の目的から外れるため省略するが，以下に示すプログラムは文献 4) [Chapter 5.4] のアルゴリズムに基づいている．また，簡単のためここでは 2 進グレイコードのみを対象とし，グレイコードの桁の型を

$$\mathbf{data}\ \mathsf{GrayBit} = \mathsf{G}_0 \mid \mathsf{G}_1$$

と定義し，グレイコードはこれらの要素から成るリスト (下位が先頭) により表されるものとする．

整数値から対応するグレイコードを計算する関数 $toGray$ は次のように定義される:

```
toGray :: Int → [GrayBit]
toGray = unfoldr xmodDiv
  where xmodDiv :: Int → Maybe(GrayBit, Int)
        xmodDiv s = if s = 0 then Nothing
                    else Just(G_{(s+s/2)%2}, s/2)
```

文献 4) ではグレイコードの各桁を計算するために排他的論理和を用いていたが，ここでは和と 2 による剰余算で代用している．

逆に，グレイコードから対応する値を計算する関数 $fromGray$ は，$foldr$ を用いて以下のように定義される:

```
fromGray :: [GrayBit] → Int
fromGray = foldr xsumMul 0
  where xsumMul (G_i, s) = (i + s)%2 + s × 2
```

4.2 n 進表現からグレイコードへの変換

n 進表現から値への変換は $foldr$ によって与えられ，値からグレイコードへの変換 $toGray$ は $unfoldr$ によって与えられるため，n 進表現からグレイコードへの変換は以下のような関数 $nsToGray_n$ で定義される:

$$nsToGray_n = unfoldr\ xmodDiv \cdot foldr\ sumMul_n\ 0$$

ここで，$xmodDiv$ は 4.1 節で，$sumMul_n$ は 3.2 節でそれぞれ定義した関数である．すなわち，n 進表現からグレイコードへの変換はメタモルフィズム $\{\!|xmodDiv, sumMul_n, 0|\!\}$ により与えられる．

これをジグソー化するためには，適当な $pieces$，h_0，v_0 を見つける必要がある．一般にメタモルフィズム $\{\!|f, g, e|\!\}$ に対し，特にジグソー化条件 (3) を満たす関数 $pieces$ を特定することは難しいが，f と g が単射であるときについては比較的容易である場合が多い．今回の例では $xmodDiv$ が単射ではないため，次のように関数 $nsToGray_n$ を定義し直す:

```
nsToGray_n =
    map fst · unfoldr xmodDiv_inj · foldr sumMul_n 0
  where xmodDiv_inj ::
          Int → Maybe((GrayBit, GrayBit), Int)
        xmodDiv_inj s =
          if s = 0 then Nothing
          else Just((G_{(s+s/2)%2}, G_{s%2}), s/2)
```

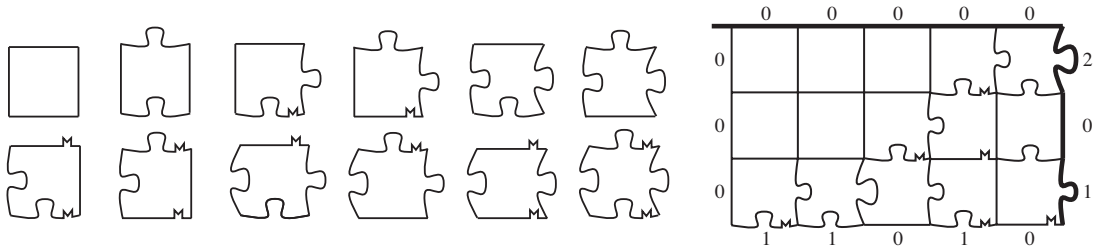


図 4 3 進表現からグレイコードへ変換するジグソーピースの集合と変換例 (201_3 から 11010_{gray} への変換)

関数 $xmodDiv_{\mathrm{inj}}$ は，関数 $xmodDiv$ で出力される情報の他に冗長な情報が追加されているが，$unfoldr$ により生成されたリストからそれを最後に削除する $(map\ fst)$ ことで元の定義と同じように振る舞う関数として定義されている．また，$xmodDiv_{\mathrm{inj}}$ はその冗長な情報により単射である．ここでは，メタモルフィズム $\{\!|xmodDiv_{\mathrm{inj}}, sumMul_n, 0|\!\}$ をジグソー化することで，関数 $nsToGray_n$ を計算するジグソーモデルを導出する．このアプローチにより得られる関数 $pieces$ の型は，$(\mathsf{DigitM}, (\mathsf{GrayBit}, \mathsf{GrayBit})) \to ((\mathsf{GrayBit}, \mathsf{GrayBit}), \mathsf{DigitM})$ となる．

まず，ジグソー化条件 (1) は自明であり，条件 (2) は簡単な計算により $\underline{v_0 = \mathsf{M}_0 \text{ とする}}$ ことで成り立つことが分かる．条件 (3) を満たす $pieces$ や h_0 は次のようにして求められる．

(i) $xmodDiv^{\#}_{\mathrm{inj}}$ の簡略化による h_0 の特定

関数 $xmodDiv_{\mathrm{inj}}$ の定義から，

$$xmodDiv^{\#}_{\mathrm{inj}}\ s = \textbf{ if } s = 0 \textbf{ then } (h_0, 0) \textbf{ else } ((\mathsf{G}_{(s+s/2)\%2}, \mathsf{G}_{s\%2}), s/2)$$

となるが，$\underline{h_0 = (\mathsf{G}_0, \mathsf{G}_0) \text{ と定める}}$ ことで

$$xmodDiv^{\#}\ s = ((\mathsf{G}_{(s+s/2)\%2}, \mathsf{G}_{s\%2}), s/2)$$

と簡略化することができる．

(ii) $xmodDiv^{\#}_{\mathrm{inj}}(sumMul_n(\mathsf{M}_i, s))$ の計算

ジグソー化条件 (3) の帰結部の右辺 $(f^{\#}(g(v, s)))$ について期待される結果を計算しておく．

$$\begin{aligned} &xmodDiv^{\#}_{\mathrm{inj}}(sumMul_n(\mathsf{M}_i, s)) \\ &= xmodDiv^{\#}_{\mathrm{inj}}(i + s \times n) \\ &= ((\mathsf{G}_{(t+t/2)\%2}, \mathsf{G}_{t\%2}), t/2) \textbf{ where } t = i + s \times n \end{aligned}$$

(iii) $pieces$ の特定

ジグソー化条件 (3) について，$(h, s) = (\mathsf{M}_i, s)$ とおく．(i) と (ii) から，或る j に対し，

$$pieces\ (\mathsf{M}_i, (\mathsf{G}_{(s+s/2)\%2}, \mathsf{G}_{s\%2})) = ((\mathsf{G}_{(i+s\times n+(i+s\times n)/2)\%2}, \mathsf{G}_{(i+s\times n)\%2}), \mathsf{M}_j)$$

が成り立ち，ジグソー化条件 (3) は，この j が

$$\begin{aligned} (i + s \times n)/2 &= sumMul_n(\mathsf{M}_j, s/2) \\ &= j + s/2 \times n \end{aligned}$$

を満たすことを期待している．よって，

$$\begin{aligned} j &= (i + s \times n)/2 - s/2 \times n \\ &= (i + s\%2 \times n)/2. \end{aligned}$$

すなわち，

$$\begin{aligned} &pieces\ (\mathsf{M}_i, (\mathsf{G}_{(s+s/2)\%2}, \mathsf{G}_{s\%2})) \\ &= ((\mathsf{G}_{(i+s\times n+(i+s\times n)/2)\%2}, \mathsf{G}_{(i+s\times n)\%2}), \mathsf{M}_j) \\ &= ((\mathsf{G}_{(i+(s+s/2)\%2\times n+(i+s\%2\times n)/2)\%2}, \mathsf{G}_{(i+s\%2\times n)\%2}), \\ &\quad \mathsf{M}_{(i+s\%2\times n)/2}) \end{aligned}$$

を得る．これにより，

$$pieces\ (\mathsf{M}_i, (\mathsf{G}_j, \mathsf{G}_k)) = (((\mathsf{G}_{i+j\times n+l/2)\%2}, \mathsf{G}_{l\%2}), \mathsf{M}_{l/2}) \textbf{ where } l = i + k \times n$$

が導出された．

$nsToGray_n$ を計算するジグソーモデルでは，最後に $map\ fst$ という関数を適用する必要があるが，これはグレイコードの各桁に対応する曲線が一意ではないと考えれば無視できる手続きである．たとえば，グレイコードの桁の型を

$$\textbf{data } \mathsf{GrayBit}^2 = \mathsf{G}^0_0 \mid \mathsf{G}^1_0 \mid \mathsf{G}^0_1 \mid \mathsf{G}^1_1$$

とし，G^0_i も G^1_i もグレイコードにおける i という桁に対応するようなジグソーモデルを考えればよい．これらは，上で得られたピースにおける τ_{H} 型の $(\mathsf{G}_i, \mathsf{G}_0)$ と $(\mathsf{G}_i, \mathsf{G}_1)$ に対応する．この型を用いることにより，

$$pieces\ (\mathsf{M}_i, \mathsf{G}^k_j) = (\mathsf{G}^{l\%2}_{i+j\times n+l/2)\%2}, \mathsf{M}_{l/2}) \textbf{ where } l = i + k \times n$$

のような簡潔な $pieces$ 関数の定義を得ることができる．

3 進表現からグレイコードに変換するジグソーモデルに対して，この考え方を適用すると $pieces$ に相当する関数 p は

$$\begin{aligned} &p\ (\mathsf{M}_0, \mathsf{G}^0_0) = (\mathsf{G}^0_0, \mathsf{M}_0); \quad p\ (\mathsf{M}_0, \mathsf{G}^1_0) = (\mathsf{G}^1_1, \mathsf{M}_1) \\ &p\ (\mathsf{M}_0, \mathsf{G}^1_1) = (\mathsf{G}^1_1, \mathsf{M}_0); \quad p\ (\mathsf{M}_0, \mathsf{G}^1_1) = (\mathsf{G}^1_0, \mathsf{M}_1) \\ &p\ (\mathsf{M}_1, \mathsf{G}^0_0) = (\mathsf{G}^1_1, \mathsf{M}_0); \quad p\ (\mathsf{M}_1, \mathsf{G}^1_0) = (\mathsf{G}^1_1, \mathsf{M}_2) \\ &p\ (\mathsf{M}_1, \mathsf{G}^1_1) = (\mathsf{G}^1_0, \mathsf{M}_0); \quad p\ (\mathsf{M}_1, \mathsf{G}^1_1) = (\mathsf{G}^0_0, \mathsf{M}_2) \\ &p\ (\mathsf{M}_2, \mathsf{G}^0_0) = (\mathsf{G}^0_1, \mathsf{M}_1); \quad p\ (\mathsf{M}_2, \mathsf{G}^1_0) = (\mathsf{G}^1_0, \mathsf{M}_2) \\ &p\ (\mathsf{M}_2, \mathsf{G}^0_1) = (\mathsf{G}^0_0, \mathsf{M}_1); \quad p\ (\mathsf{M}_2, \mathsf{G}^1_1) = (\mathsf{G}^1_1, \mathsf{M}_2) \end{aligned}$$

と定義できる．図 4 は，この $pieces$ 関数に対応する

ピース集合とそれを利用して得られる 3 進表現からグレイコードへの変換の例である．2 節の例と同じ曲線を用いて各辺の 0，1，2 を表現しているが，横方向の辺には M 字の切り込みが入っているものといないものがある．これらは，G_i^0 (切り込みなし) と G_i^1 (切り込みあり) を表し，どちらもグレイコードの桁 i に相当するが，計算過程では振舞いが異なる．ただし，結果が現れるボードの下端ではその情報は必要がないため，切り込みを無視して読めばよい．

4.3 グレイコードから n 進表現への変換

グレイコードから値への変換 *fromGray* は *foldr* によって与えられ，値から n 進表現への変換は *unfoldr* によって与えられるため，グレイコードから n 進表現への変換は以下のような関数 $grayToNs_n$ で定義される:

$$grayToNs_n = unfoldr\ modDiv_n \cdot foldr\ xsumMul\ 0$$

ここで，$modDiv_n$ は 3.2 節で，*xsumMul* は 4.1 節でそれぞれ定義した関数である．すなわち，グレイコードから n 進表現への変換はメタモルフィズム $\{\!|modDiv_n, xsumMul, 0|\!\}$ により与えられる．$modDiv_n$ も *xsumMul* も単射であるが，*pieces* を直接特定することが難しいため，$grayToNs_n$ を以下のように再定義する:

$$\begin{aligned}
&grayToNs_n = \\
&\quad map\ fst \cdot unfoldr\ modDiv_n^{\mathrm{ex}} \cdot foldr\ xsumMul\ 0 \\
&\mathbf{where}\ modDiv_n^{\mathrm{ex}} :: \\
&\qquad \mathsf{Int} \to \mathsf{Maybe}((\mathsf{DigitN}, (\mathsf{GBit}, \mathsf{GBit})), \mathsf{Int}) \\
&\qquad modDiv_n^{\mathrm{ex}}\ s = \\
&\qquad\quad \mathbf{if}\ s = 0\ \mathbf{then}\ \mathsf{Nothing} \\
&\qquad\quad \mathbf{else}\ \mathsf{Just}((\mathsf{N}_{s\%n}, (\mathsf{G}_{s/n\%2}, \mathsf{G}_{s\%2})), s/n)
\end{aligned}$$

ここで，GBit は GrayBit の略記である．*unfoldr* で生成されるリストは $[(\mathsf{DigitN}, (\mathsf{GBit}, \mathsf{GBit}))]$ 型であり出力には不要な情報を含んでいるため，*map fst* によって削除している．

関数 *grayToNs* に対応するジグソーモデルは，メタモルフィズム $\{\!|modDiv_n^{\mathrm{ex}}, xsumMul, 0|\!\}$ をジグソー化することで得られるが，以下，4.2 節と同様の手順で，

$$\begin{aligned}
&pieces\ (\mathsf{G}_i, (\mathsf{N}_j, (\mathsf{G}_k, \mathsf{G}_l))) = \\
&\quad ((\mathsf{N}_{p\%n}, (\mathsf{G}_{p/n}, \mathsf{G}_{(i+l)\%2})), \mathsf{G}_{(p/n+k)\%2}) \\
&\qquad\qquad \mathbf{where}\ p = (i+l)\%2 + j \times 2 \\
&h_0 = (\mathsf{N}_0, (\mathsf{G}_0, \mathsf{G}_0)) \\
&v_0 = \mathsf{G}_0
\end{aligned}$$

を得る．4.2 節のときと同様に，型 DigitN を

$$\begin{aligned}
&\mathbf{data}\ \mathsf{DigitN}^{2,2} = \\
&\quad \mathsf{N}_0^{0,0} \mid \mathsf{N}_0^{0,1} \mid \mathsf{N}_0^{1,0} \mid \mathsf{N}_0^{1,1} \mid \mathsf{N}_1^{1,1} \mid \cdots \mid \mathsf{N}_{n-1}^{1,1}
\end{aligned}$$

のように再定義してもよい．これにより，

$$\begin{aligned}
&pieces\ (\mathsf{G}_i, \mathsf{N}_j^{k,l}) = (\mathsf{N}_{p\%n}^{p/n,(i+l)\%2}, \mathsf{G}_{(p/n+k)\%2}) \\
&\qquad\qquad \mathbf{where}\ p = (i+l)\%2 + j \times 2 \\
&h_0 = \mathsf{N}_0^{0,0} \\
&v_0 = \mathsf{G}_0
\end{aligned}$$

が得られ，これらを利用して，グレイコードから n 進表現へ変換するジグソーモデルを定義することもできる．

5. おわりに

文献 5) で提案されたジグソーモデルに対し，応用例とその導出について議論を行った．元の文献では具体的な導出法まで言及できなかったが，本稿ではグレイコードと位取り記数法との間の相互変換を例にとり，ジグソーモデルの導出を手順を示した．本論文の例で見たように，与えられたメタモルフィズムに対して直接ジグソー化条件を満たす関数 *pieces* を見つけられない場合でも，定義を少し変形することで導出できることもある．ただ，どのようにメタモルフィズムを変形すべきかについては自明ではないため，その究明が今後の課題である．

参考文献

1) Bird, R. and Gibbons, J. (2003) Arithmetic coding with folds and unfolds. In *4th Advanced Functional Programming, Lecture Notes in Computer Science*, 2638: pp. 1–26. Springer-Verlag.
2) Gibbons,J. (2007) Metamorphisms: streaming representation-changers. *Science of Computer Programming*, 65 (2): pp. 108–139. Elsevier.
3) Gray, F. (1953) Pulse code communication. *U.S. Patent 2,632,058*, March 17, 1953 (filed Nov. 1947).
4) Haupt, R. L., Haupt, S. E. (2004) Practical Genetic Algorithms, Second Edition. Wiley-Interscience.
5) Nakano, K. (2012) Metamorphism in Jigsaw. *Journal of Functional Programming*: to appear. Cambridge University Press.