解 説

基
専

双方向変換

― 古典的なビュー更新問題から，プログラミング言語によるアプローチとソフトウェア開発への応用まで―

日高宗一郎（法政大学情報科学部）

双方向変換とは

双方向変換とは，2つ（もしくはそれ以上）の情報源の間で変換を介して一貫性を維持する仕組みのことを指す．データの変換は情報システムの中で欠かせない操作であるが，双方向変換により，一貫性を保ちながら，その変換をこえて双方向に更新を伝搬させることができる．二方向で計2つの変換をプログラムし一貫性を応用ごとにプログラマが保証するのに比べて，双方向変換では1つのプログラムだけで済み，一貫性が自動的に保証される．この一貫性維持の問題自体は古く，70年代の関係データベースにおけるビュー更新問題に遡ることができる．しかし近年，この古い問題に新しいプログラミング言語的アプローチが適用されたり，モデル（ソフトウェアなどの成果物を表現する**図-1**のような属性つきグラフ）を対象とした双方向変換がソフトウェア工学分野で研究されるなど，計算機科学のさまざまな分野で盛り上がりを見せている．双方向変換をテーマにソフトウェア工学，プログラミング言語，データベース，グラフ変換などの複数のコミュニティが議論しあうワークショップも立ち上がっている☆1．また，プログラミング言語分野で提案されたアプローチの1つはHaskellライブラリ実装が12万ダウンロードを超えており，日本でもユーザの勉強会が開催されている．本稿では，このような古くて新しい問題について，言語的アプローチを中心にさまざまなアプローチを取り上げ，それぞれの特徴やさまざまな応用，今後の展望について紹介する．

☆1 http://bx-community.wikidot.com/bx-events/

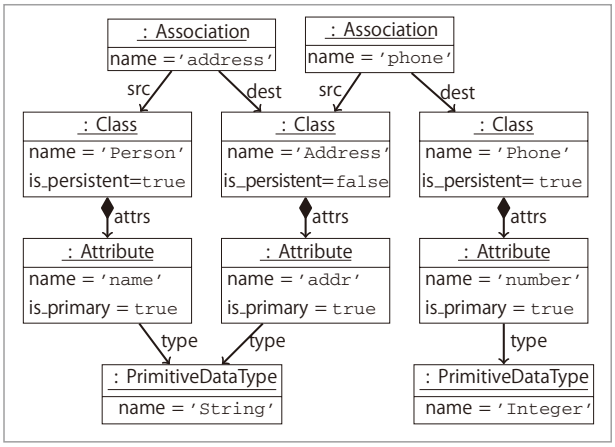

図-1　モデル駆動ソフトウェア開発で用いられるモデルの例

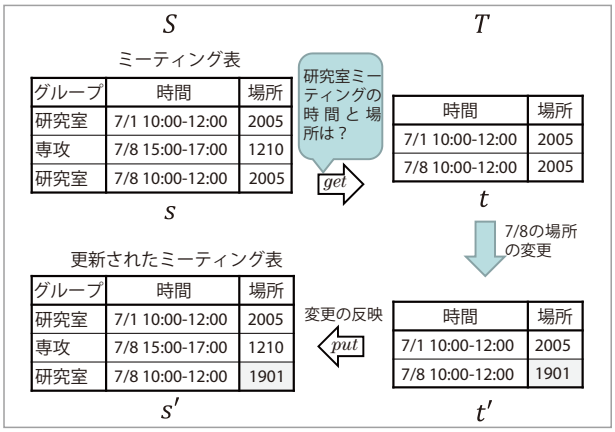

図-2　ミーティングスケジュールに関する双方向変換の例

▶基本的な双方向変換の定義と満たすべき性質

図-2は，ある研究室のミーティングを管理している表である．この表には実際はもっと膨大な列が含まれているとして，関心のある研究室ミーティングの時間と場所だけを取り出す変換を考える．変換の結果は右上の表のようになる．ここで，7/8の研究室ミーティングの場所を，発表者の都合で2005から1901に変更したいとする．この変更も先ほど作成したコンパクトな表の上で行え，その結果を元の大きな表に反映できると便利である．ただし，こ

の反映により更新されたミーティング表を変換することにより研究室ミーティング表を取り出すと，上記の変更後の表が再び得られなければならない．このような満たすべき性質を定式化し，更新を安全に伝搬させる変換が，本稿の主題である双方向変換である．ミーティング表の変更を，右側の研究室ミーティング表に反映させるのは，前述の抜き出しの変換を再度適用すればよいが，この両方向の伝搬ができるため双方向変換と呼ばれる．ここで注意されたいのは，最初の変換が情報を捨てていることである．具体的には，グループ名を格納する「グループ」列全体と，研究室以外のグループである専攻のミーティング情報が捨てられている．そのため，更新された研究室ミーティング表から，元の大きい表を直接作ることができず，捨てられた情報を含む元のミーティング表を参照しながら生成する必要がある．このことについては後ほど触れる．

双方向変換の用途はこれ以外にも異なるブラウザ間のブックマークなどのデータ同期があるが，状況ごとに両方向の変換をそれぞれ別個に記述していては，満たすべき性質の保証が困難であり，また変換自体を拡張・変更した場合には再度保証する必要が生じてしまう．そのため，近年，満たすべき性質が自動的に保証されるプログラミング言語が提案され，扱うデータの多様化や変換自体の高度化などの研究が盛んに行われている．応用面でも，後に紹介するようなソフトウェア開発への用途が研究されている．

以下ではまず，双方向変換の基本的な性質を，より単純な例と多少数学的な表現を借りつつ見ていく．

双方向変換は2つの集合の間の変換であり，変換対象の一方をソース，他方をターゲットと呼ぶことにする．また，ソースからターゲットをつくる変換を順方向変換と呼ぶ．先の例では，ソース側の集合は可能なミーティング表全体，ターゲット側の集合は研究室ミーティング表全体，順方向変換がミーティング表から，その一部を取り出すことにより研究室ミーティング表を計算する問合せである．変換が双方向であるためにもう1つの変換(逆方向変換)も用意する．先の例では更新された研究室ミーティング表から更新されたミーティング表を生成する変換にあたる．双方向変換が有用であるためには，先に述べた満たすべき性質である「行って戻れる」という性質が必要である．最も分かりやすい双方向変換は，ソースとターゲットが同じ情報を持っているものである．たとえば，順方向変換が，時間と場所を列ごと入れ換えるという具合である．この場合，逆方向変換では列を再度入れ換えればよい．しかし，それでは制約が強すぎる場合もある．たとえば研究室ミーティング表のようにソースの一部の情報を抽出するような場合は順方向変換により情報は失われる．そのような変換も扱うためには，逆方向変換は元のミーティング表を参照しなければならない．たとえば，グループ名は列「グループ」から取り出すことにより復元できる．ここで,「専攻」というグループ名は「研究室」以外の勝手な名前としても，同じ問合せによりターゲット側で同じ研究室ミーティング表を得ることができることに注意されたい．ここでは，そのうち「最も驚きの少ないもの」を選んでいる．たとえば，ミーティング表をまったく変更しないで逆方向変換してもグループ名が変わるのは不自然であり，ここではグループ名が変わることがないようにしている．

双方向変換と，その「行って戻れる」という性質は，双方向変換を以下のように形式的に扱うことで，簡潔な式にまとめられる．

双方向変換のソース側とターゲット側の集合をそれぞれS，Tとし，順方向変換を関数$f: S \to T$で表す．ソース側の変換対象$s \in S$（集合Sのある要素s）をソース，同様にターゲット側$t \in T$をターゲットと呼ぶことにする．ミーティング表の双方向変換の最も簡単な例として挙げた，列を単に入れ換える例では，順方向変換が全単射になっているといえる．このような場合，逆方向変換は順方向変換同様の一引数の関数とすることができる．たとえば，任意の実数$s \in \mathbb{R}$について$t = f(s) = 2s \in \mathbb{R}$なら$t \in T$について$f$の逆関数$f^{-1}$を使って$s = f^{-1}(t) = t/2 \in \mathbb{R}$という具合である．しかし，たとえば先の研究室ミーティング表のようにソースの一部の情

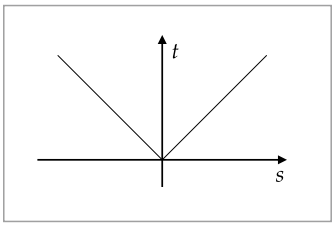

図 -3
全単射でない関数の例：
$t=|s|$

報を抽出するような場合は一般に全単射とはならない．そのような変換も扱うためには，逆方向変換は逆関数ではなく，別の関数 $g: T \times S \to S$（T と S の要素をそれぞれ 1 つずつ取り，S の要素を返す関数）を用意し，元の値 $s \in S$ も参照できるようにする．たとえば，f が**図 -3** のような絶対値関数

$$f(s)=|s| \tag{1}$$

ならば，s の符号（正ならば 1，負ならば -1）を sign(s) として

$$g(t, s) = \mathrm{sign}(s)|t| \tag{2}$$

とすれば，順方向変換で失われた符号の情報を復元することができる．ここで，ミーティング表の例で挙げたのと同じように，$f(s)=t$ を満たす s は一般に複数ある．たとえば $g(t, s) = -t$ としても，$|-t|=|t|$ なのでターゲットはソースの絶対値という関係を満たす．よって，g の定義では，その中から「最も驚きの少ない」ようにしている．たとえば，$s=-1$ から順方向変換で $t=1$ が得られるが，これを変更せずに逆方向変換して $s=1$ が得られるのは不自然である．

「行って戻れる」という性質は，順方向変換を *get*，逆方向変換を *put* とすると，以下のようにまとめられる．

$$put\ ((get\ s), s) = s \tag{GetPut}$$
$$get\ (put\ (t, s)) = t \tag{PutGet}$$

GetPut は，行って何もしなければ元の値に戻ることができること，PutGet は，ターゲットのすべての情報をソースに伝えられること（だから *get* で常にターゲットが復元できる）を示している．いくつかの *put* の定義が上記の性質を満たすかどうか確認してみると，$put(t, s)=-t$ とすると，GetPut は満

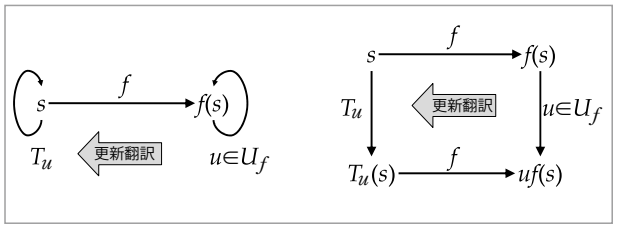

図 -4　ビュー更新問題における Acceptability（左）と Consistency（右）

たさないが，PutGet は満たし，(2) の定義だと両方満足することが分かる．ここで，もし $t<0$ のような値への変更を許すと，PutGet は決して満たすことができないことが分かる．なぜなら，このような変更は順方向変換の値域を逸脱しているため，再び順方向変換を施したときに復元できないからである．よって *put* はそのような入力を拒絶，つまり一般には部分関数として実現されるか，$f: \mathbb{R} \to \mathbb{R}^{+} \cup \{0\}$ のようにターゲットの範囲を明示する．

このように，$t=|s|$ のような与えられた関係を保存しながら変更を双方向に伝えあうような変換が双方向変換である．

▶双方向変換のタイプと満たすべき性質

非対称型（ビュー更新型）双方向変換とその性質

順方向変換で情報を失うかもしれない場合も含めた双方向変換の定式化として，上記ではプログラミング言語分野で多用されるもの [1] を使った．しかし，引用した論文のタイトルにもあるように，双方向変換はデータベース分野ではビュー更新問題として，双方向変換としての性質が古くから定式化されている [2]．ビュー更新問題とは，データベース $s \in DB$ から問合せ $f: DB \to V$ を通じて構成されるビュー $f(s)$ 上の更新 $u: V \to V$ を，データベース上の更新 $T_u: DB \to DB$ に翻訳することにより反映させることである．最初に紹介した研究室ミーティング表の更新はまさにビュー更新問題である．このとき満たすべき性質は，

$$(u \circ f)(s) = f(s) \Rightarrow T_u(s) = s \tag{Acceptability}$$
$$u \circ f = f \circ T_u \tag{Consistency}$$

となっている（**図 -4**）．ここで，$u \circ f$ は関数 u と

fの合成で，先にfを適用し，次にuを適用することを意味する．Acceptabilityは，ビュー上での更新uが恒等関数の場合は，対応するデータベース上の更新も恒等関数となる，つまりビュー上での更新がない場合は，データベース上の更新もないことを示しており，先のGetPutに対応している．一方，Consistencyは，ビューをつくってからビュー上でuにより更新する操作と，uをデータベース上の更新T_uに翻訳し，その更新のあとにビューをつくる操作が等しいことを示している．つまり，更新されたビュー$(u \circ f)(s)$が，その更新を反映させてできたデータベース$(T_u(s))$から再度ビューを構築することにより再現できることを示しており，先のPutGetに対応している．

対称型（相互同期型）双方向変換とその性質

上記では，一貫性維持対象の2つの情報源のうち，ターゲットに必ずしもすべての情報が伝わらない場合を扱ったため，順方向変換は1引数の関数で表現されていた．つまり，ターゲットがソースの一部の情報を抽出したものとなっていた．しかし，後に紹介するデータ同期の例のように，ターゲットからソースへも，必ずしもすべての情報が伝わらない場合がある．そこでもう少し一般化すると，順方向変換も古いターゲットを引数に加えた二引数関数とする対称な枠組みとして，上記のような場合を定式化することができる．一貫性の表現も単一の関数では表現できなくなり，改めて関係$R \subset S \times T$で表現する．ここで$S \times T$はSとTの要素のすべての組合せの集合である．ソース$s \in S$とターゲット$t \in T$が一貫しているとき，$(s, t) \in R$とする．このとき，順方向変換を右向き矢印を使って$\vec{T}$，逆方向変換を反対の$\overleftarrow{T}$とすると，GetPutやPutGetに対応する以下の性質

$$(s,t) \in R \Rightarrow \vec{T}(s,t) = t \wedge \overleftarrow{T}(s,t) = s \qquad \text{(Hippocraticness)}$$

$$\left(s, \vec{T}(s,t)\right) \in R \wedge \left(\overleftarrow{T}(s,t), t\right) \in R \qquad \text{(Correctness)}$$

を満たすものは制約maintainer（あるいは単にmaintainer）[3]と呼ばれる．Hippocraticnessは，すでに一貫性のあるソースとターゲットに同期を適用しても，同じもの（順方向変換では元のターゲット，逆方向変換では元のソース）が得られることを表しており，非対称の枠組みでのGetPutに対応する．一方Correctnessは，同期を適用した後の値は必ず一貫性のあるものが得られることを表しており，非対称の枠組みにおいて更新後の値tに対して，*put*で必ず*get* $s=t$を満たすようなsが得られるというPutGetに対応している．

双方向変換へのさまざまなアプローチ

これまで，双方向変換の説明で，ある一定の条件を満たす一組の変換が必要であることを述べた．しかし，実際に2つの変換を別々に記述し，常に条件を満たすよう保証することは容易ではない．プログラマの立場からは，1つのプログラムの記述で，その言語設計により自ずから条件が満たされるようになっているとありがたい．以下で紹介するアプローチは，いずれもこの要望を実現するものである．

▶コンビネータアプローチ

コンビネータ（結合子）アプローチは，プログラムを構成する要素ごとに満たすべき性質（well-behavedness ＝ 振舞いの良さ）つまりGetPutとPutGetを満たす*get*と*put*の動作を定義しておき，電子回路を組むようにこの基本要素を組み合わせて双方向変換プログラムを構成できるようにしたものである[1]．基本要素の結合（combination）に基づくことからコンビネータアプローチと呼ばれる．一般に，双方向変換が複雑になるほど，*get*と*put*を独立に記述し，well-behavednessを満たすことを個別に証明するのは大変である．コンビネータアプローチでは，コンビネータを組み合わせてできた大きなプログラムは各コンビネータの定義によりwell-behavednessを満たすため，そのような個別の証明は不要である．**図-5**左の基本単位の中で，右向き三角が*get*，左向き三角が*put*を表現している．両関

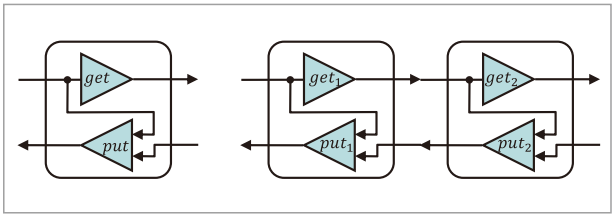

図 -5　Lens の基本単位（左）とその直列合成（右）

数の引数の数に対応して，前者は 1 入力，後者は 2 入力である．はじめから *get* と *put* をペアにしておくのがポイントであり，ペンシルバニア大学のグループが提案したこのペアは lens と呼ばれる．この 2 つの双方向変換を直列に合成したものが図 -5 右である．まさに電子回路のように単にそのまま繋ぎあわせただけであるが，組合せ前の 2 つの lens がそれぞれ GetPut を満たすならば，組み合わせた後も GetPut を満たすように絶妙に配線されている．確認してみよう．

$$
\begin{aligned}
& put\ (get\ s,\ s) \\
=& \{\text{配線の仕方より}\} \\
& put_1\ (put_2\ ((get_2\ (get_1\ s)),\ (get_1\ s)),\ s) \\
=& \{\text{右側の lens の GetPut より}\} \\
& put_1(get_1\ s,\ s) \\
=& \{\text{左側の lens の GetPut より}\} \\
& s
\end{aligned}
$$

同様に PutGet の性質も個々の lens の PutGet から導くことができる．読者各位でぜひ確かめられたい．

この直列つなぎ以外にも well-bahavedness を保ちつつ lens を組み合わせる複合 lens が同時に提案されている．Lens をこのような一定の方法で組み合わせるという意味で，コンビネータと呼ばれる．プログラマは lens コンビネータを自由に組み合わせるだけで大きな双方向変換プログラムを構成することができる．

Lens を使ったプログラミングの例を見てみよう．ここでは，Chrome[☆2] ブラウザの個々のブックマークデータから，いくつかのブラウザに共通な項目である，アドレス，追加日時 (ADD_DATE)，ページタイトルを抽出する変換を順方向変換とする双方向

```
1  <!-- Chrome -->
2  <A HREF="http://www.ipsj.or.jp/english/"
3   ADD_DATE="1449728423"
4   ICON="...">
5    IPSJ English Web Site</A>
6
7  <!-- Opera -->
8  <A HREF="http://www.ipsj.or.jp/english/"
9   ADD_DATE="1449729143" LAST_VISITED="0">
10  IPSJ English Web Site</A>
```

図 -6　各ブラウザのブックマークデータ（一部）

```
1  link_chrome =
2    hoist "*" ;
3    hd {} ;
4    hoist "A" ;
5    rename "*" "name" ;
6    rename "HREF" "url" ;
7    prune "ICON" (value "data:image/png") ;
8    wmap (mapsto "name" (hd {} ; hoist "PCDATA"))
```

図 -7　ブックマーク変換 lens の例（一部）

変換を lens を用いて記述する．ここでは**図 -6** のような HTML 形式でエクスポート／インポートされる形式を変換対象とする．**図 -7** は，上述のブックマーク変換 lens プログラムの例である．

（元祖）lens 自体が直接扱うデータは木構造であるため，ここでは HTML を lens の木構造に置き換えた上で双方向変換している[☆3]．Lens の木構造は，兄弟間で一意の枝ラベルから，その枝の先の順序のない分岐へのマッピングで表現される．hoist はその引数で指定された名前の枝を抽出するもので，; は lens の直列合成（図 -5 右：";" の左側が get_1, put_1, 右側が get_2, put_2 に対応），rename は枝ラベルのつけかえ，prune は第一引数で指定された枝を除去する．wmap はいわば高階 lens であり，各枝に lens を適用するが，ここでは name というラベルを持つ枝に対し (hd ; hoist "PCDATA") という lens を適用する．ブックマークの詳しい内部表現の説明は省略するが，タグ名と属性名を枝ラベルに格納し，順序のある子要素の並びは，二分木でエンコー

☆2　https://www.google.co.jp/chrome/

☆3　HTML も木構造の一種であるため適切なエンコーディングにより問題なく扱うことができるが，同一研究グループから文字列や関係に対応したものも後に提案されている．

```
# a_common = get link_chrome c_chrome
  {"ADD_DATE" ↦{"1449728423" ↦{}}
   "name" ↦{"IPSJ␣English␣Web␣Site" ↦{}}
   "url" ↦{"http://www.ipsj.or.jp/english/" ↦{}}}
# a'_common =
   {"ADD_DATE" ↦{"1449728423" ↦{}}
    "name" ↦{"IPSJ" ↦{}}
    "url" ↦{"http://www.ipsj.or.jp/" ↦{}}
# c'_chrome = put link_chrome (a'_common,c_chrome)
c'_chrome =
  {"*" ↦{"*h"↦
            {"A" ↦
               {"*" ↦
                  {"*h" ↦{"PCDATA" ↦{"IPSJ" ↦{}}}
                   "*t" ↦{}}
                "ADD_DATE" ↦{"1449728423" ↦{}}
                "HREF" ↦{"http://www.ipsj.or.jp/" ↦{}}
                "ICON" ↦{"data:image/png;..." ↦{}}}}
          "*t" ↦{} } }
# get link_chrome c'_chrome
{"ADD_DATE" ↦{"1449728423" ↦{}}
 "name" ↦{"IPSJ" ↦{}}
 "url" ↦{"http://www.ipsj.or.jp/" ↦{}}}
#
```

図 -8　Lens による双方向変換例

ドし，特別な名前 "*" の枝の先に置かれる．hd の第一引数は，*put* 時にソースデータに対応するものがない場合に与えるデフォルト値である．hd は先頭要素をリストから取り出す．Chrome には ICON 属性もあるが，prune で除去している（prune の第二引数は，新たに追加されたブックマークの *put* 時に，ソースデータに対応するものがないために用いるデフォルト値である）．**図 -8** に双方向変換の例を示す．1 行目で順方向変換 *get* を行い変数 a common に格納している．2 行目では *get* の結果を印字しているが，アドレスが url，追加時刻が ADD_DATE，ページタイトルが name とそれぞれラベルづけされて抽出されていることが分かる．5 行目でページタイトルを "IPSJ English Web Site" から "IPSJ" に変更し，アドレスから "english/" を削除し，9 行目で逆方向変換をしている．続く結果表示で，変更がそれぞれ対応する要素に伝搬されつつ，ICON 属性は保持されていることが分かる．20 行目では再び *get* しているが，その結果が a'_common と同一であるので，PutGet を満たしていることが分かる．

Lens を提案した論文は，プログラミング言語分野の最高峰の会議の 1 つである POPL（Principles of Programming Languages）の 2005 年の Most Influential Paper に選ばれており，さまざまな言語でライブラリが実装されている．たとえば Haskell 言語のライブラリ lens[☆4] は 12 万ダウンロードを超えており，日本でもユーザが勉強会[☆5] を開催したりしている．また Red Hat 社は lens を元にした設定ファイル編集ツール Augeas[☆6] を提供している．

▶逆方向変換導出（プログラム変換）アプローチ

上記コンビネータアプローチやその拡張は，木や関係，文字列のような固定的なデータ構造には対応しているが，ユーザ定義のデータ構造には対応していない．また，一定の数の基本コンビネータが提供されているものの，その範囲で記述できない応用については，ユーザの責任で新たな基本コンビネータを実装し well-behavedness を確認しなければならない．Matsuda ら [4] は，データ構造が自由に定義でき，*get* に対し，*put* の定義の仕方の指針をプログラム変換という形で明快に示しており，lens とは異なりプログラム記述に特殊な構文等は不要である．lens では捨てるデータを prune などで明示しなければならなかったが，Matsuda らの言語では自動的に計算される．

▶双方向解釈アプローチ

これまで紹介した手法は，木や文字列，ユーザ定義データ型を扱うことができるが，グラフのような循環のあるデータを直接扱うことは困難であった．Hidaka ら [5] は，既存の非構造グラフ用問合せ言語 UnCAL に双方向の意味を付与し，ターゲットグラフ上の更新を UnCAL の逆方向解釈によりソースグラフ上に伝搬させる手法を提案した．

▶その他のアプローチ

先に紹介したアプローチはプログラミング言語に

☆4 https://hackage.haskell.org/package/lens
☆5 http://connpass.com/event/13929/ など
☆6 http://augeas.net

基づくアプローチともいえる．一方，ソース上の特定のパターンをターゲット上の特定の値に置き換える，書き換えに基づくアプローチでは，そのような書き換え規則の集合で変換を記述する．その中で特に双方向変換として最もよく知られている 3 つ組グラフ文法（Triple Graph Grammars：以下 TGG）は，モデル変換の業界団体標準である QVT（Query/View/Transformation）に影響を与えたとされており研究やツールの実装も盛んである．

また，先に述べた一貫性の表現に関数という方向性を持つものではなくさらに一般的な関係を制約を用いて表現し，片方のデータ（入力）のもとで制約解消ツールで他方のデータ（出力）を求める方法も提案されている．記述の自由度が高い反面，ツールが解を見つけることができない場合がある．

双方向変換のさまざまな応用

双方向変換は，これまで見てきたさまざまなアプローチにより，与えられた変換をこえて双方向に更新を伝播させることができる．変換という操作は情報システムにおける基本的で重要な操作であるため，双方向変換にもさまざまな応用がある．本章では，その中から主にデータの相互同期とモデル駆動ソフトウェア開発を取り上げ，その他の応用についても簡単に紹介する．

▶データ同期

先に紹介した lens の論文では，卑近な応用例として，複数のブラウザのブックマークの間の同期が取り上げられている．本稿でも具体例を示した通り，ブックマークの形式はブラウザごとにまちまちであり，他のブラウザでは保持されない独自の情報が含まれる場合もある．たとえば Chrome では ICON 属性を保持しているが筆者の手許にあった（若干古い）Opera[☆7] では簡単に確認したところ保持していなかった（図 -6）[☆8]．このように，一方が他方の抽象化ではなく，それぞれ独自の情報を持ちながら対応付けられている情報同士だけ更新を伝播させることは，非対称 lens の頭と頭を接続する（co-targetial composition：**図 -9** 左）ことで実現できる．

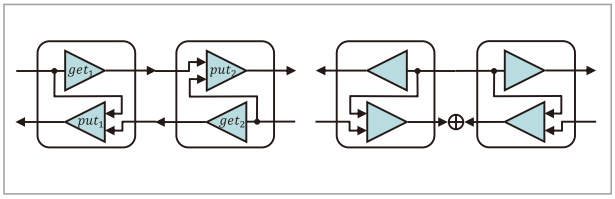

図 -9　2 つの lens のビュー側接続（左）とソース側接続（右）

$$\overrightarrow{T}\ (x, y) = put_2(get_1\ x, y)$$
$$\overleftarrow{T}\ (x, y) = put_1(get_2\ y, x)$$

ここで $\overrightarrow{T}\ (x, y)$，$\overleftarrow{T}\ (x, y)$ の組は前述の maintainer である．図 -9 左の右側の lens は，元の lens を 180 度回転させたものである．順方向変換は左側の lens の $get : S \to M$ と右側の lens の $put : M \times T \to T$，逆方向変換は右側の $get : T \to M$ と左側の $put : M \times S \to S$ の関数合成で実現している．これに関しても，$(s, t) \in R$ を左右それぞれの lens（get_1 と put_1，get_2 と put_2）について

$$\exists\ m.\ get_1\ \ s{=}m \wedge get_2\ \ t{=}m$$

つまり左右の get から得られる共通の値 m（図 -8 では a_common）が存在するとすれば，Hippocraticness および Correctness を満たしていることを確かめられたい．m は s, t それぞれの情報の一部を捨てた共通部分が含まれていると捉えることができる．共通部分がうまく一致するように両 lens を設計する必要がある．

互いに独自のデータを持つ両端のデータを，どのように接続すればよいだろうか．Lens の場合，上記のような共通部分からソースを以下の性質をみたす $create$ 関数で常に求めることができる．

$$get\ (create\ t) = t \qquad \text{(CreateGet)}$$

☆7 http://www.opera.com/ja

☆8 ここでは Opera ブックマークの記述能力を問題としているのではないので，このまま話を進める．

```
1  # chrome2opera c_chrome c_opera =
2    put link_opera ((get link_chrome c_chrome), c_opera))
3  # c'_opera = chrome2opera c'_chrome c_opera
4   {"*" ↦{"*h" ↦
5              {"A" ↦
6                 {"*" ↦{"*h" ↦{"PCDATA" ↦{"IPSJ"↦ {}}}
7                           "*t" ↦{}}
8                  "ADD_DATE" ↦{"1449728423" ↦{}}
9                  "HREF" ↦{"http://www.ipsj.or.jp/" ↦{}}
10                 "LAST_VISITED" ↦{"1449728423"↦ {}}}}
11            "*t" ↦{}}}
12 #
```

図 -10　Lens による maintainer の構成とブックマーク間同期例（一部）

create は lens において逆方向変換でソースが欠如しているため *put* が使えない場合に用意されているものである．同期に関する使い方としては，*S*, *T* 共通部分がすでに用意されている場合，それを *m* として，*S*, *T* 側をデフォルト値として両方を初期化する．

$$s_0 = create\ m, t_0 = create\ m$$

以降は *S* 側を更新すると，$\vec{T}\ (s, t_0)$ では *T* 側でもともと含まれていて *M* に反映されない部分は維持され，*T* 側を更新すると *S* 側にしか含まれていない情報は残りつつ *T* 側の更新が *S* 側に反映される．

片側のデータ（たとえば *S* 側）だけ用意されている場合は，$\vec{T}\ (s, \Omega)$ で反対側を初期化する．ここで Ω は，値が与えられていないことを示し，$put(t, \Omega) = create\ t$ である．

図 -10 に，lens を上記の方法で 2 つ繋げて maintainer を構成し，Chrome のブックマークデータの変更を Opera のブックマークデータに伝搬させる例を示す．1 行目で前述の伝搬をさせる maintainer を定義している（逆向きも同様に定義できる）．3 行目で，図 -10 で得た更新後の Chrome ブックマークデータを用いて，同じ更新を Opera のブックマークデータに伝搬させている．Opera 固有の属性 LAST_VISITED は保持されていることが分かる．

▶モデル駆動ソフトウェア開発

効率的で堅牢なソフトウェア開発のために設計から実装までの工程をモデル変換で実現する手法の 1 つであるモデル駆動ソフトウェア開発では，モデルで表現されたソフトウェアの高次表現が，実装に近い表現に向けてモデル変換を通して精緻化されていくが，この変換が双方向になれば，下流で露見したバグなどの問題の修正を上流に反映させることで，高次表現の再利用時に問題の再現を防ぐことができる．このような期待もあって，双方向変換の研究はソフトウェア工学の分野でも盛んに行われている．その中でも前述の通り TGG に基づいたツールは精力的に開発が進められている．TGG のような書き換え規則に基づくアプローチでは，基本要素を自由に組み合わせることができるコンビネータアプローチと異なり，適切な双方向変換のために規則同士の干渉や整合性に留意して使用しなければならない．そのため，TGG などの書き換え規則に基づくアプローチでは，規則の解析を支援するツールが充実している．また，複数の規則を 1 つにまとめる手法も研究されている．

先に紹介した双方向解釈アプローチに基づく Hidaka 等の双方向グラフ変換も，モデル駆動工学への応用が研究されており，モデルとコードの協調発展 [6] などの例がある☆9．

▶その他の応用

データ同期の応用例では，非対称双方向変換のビュー側接続で実現していたが，より単純に図 -9 右のように組み合わせる（今度は左側の lens が左右反転していることに注意されたい）ことで，ソース側を共有し目的ごとのビューをそれぞれ更新する☆10 というシナリオもあり得る．必要な情報は共通のソースにすべて集約され，ビューは目的に応じて抜き出された情報で構成されている場合に有効である．このように，変更の伝搬が有効な変換が用いられるあらゆる場面への応用が期待される．

☆9　文献 6）が発表された ICSE'12 の models セッションは 4 件中 3 件がモデル変換に直接かかわるものであった．
☆10　ソース側で同時に書き込みが起こらないような排他制御（ロック等：図中では記号で表現している）は別途必要となる．

双方向変換の今後の展望

双方向変換の記述法に関して，本稿では主に *get* ベース，つまりプログラマは主に順方向変換を意識し，逆方向変換は順方向変換から導出されるアプローチを取り上げた．しかし *get* ベースの手法では，逆方向変換の結果が複数あり得る場合プログラマが制御するのが困難である．それに対し，*put* ベースではプログラマは与えられたソースと更新されたビューから，新たなソースを計算する *put* 関数を直接記述する[7]ため，そのような曖昧性は避けられる．このように，双方向変換の記述方式のさらなる進展も期待される．また，本稿の冒頭では双方向変換の対象を「2つ（もしくはそれ以上）」としたが，3つ以上の情報源を扱う仕組みは未解決であり，2016年秋の湘南合同ワークショップでもパネル討論の議題として熱心に議論され，モデル駆動工学に関する国際会議 MODELS の 2017 年の本会議でも発表される予定である．一方，変換プログラミングのシナリオとしても注目が高まっており，2010 年から開催されている変換ツールコンテストの 2017 年のシナリオとして双方向変換が採用されている☆11．

双方向変換は，かつて，これまでいくつか紹介したようなデータベース，プログラミング言語，グラフ変換，ソフトウェア工学の分野で個別に研究されてきたが，2008 年の湘南合同ワークショップを機に問題意識が共有されるようになり，以来分野横断的な努力が続けられている．例として双方向変換のベンチマークの整備が挙げられる．さまざまな双方向変換アプローチが提案される中で，有効性の客観的評価のために，共通の尺度が求められている．特に，どのような変換シナリオがサポートされれば有用な双方向変換といえるかについて，論文では具体例がしばしば用いられるが，執筆者ごとに例がまちまちでは，例自体の説明に多くの紙面を費やしてしまうこともあり，分野で共有される例の引用で済ませることも望まれていた．そこで双方向変換のシナリオの共通リポジトリが構築され，テストケース，任意の大きさの入力の生成プログラム，テストの成功失敗の自動判定プログラムなどがコミュニティサイト☆12で提供されている．このような分野横断の努力により，分野ごとの知識や経験がより効率よく集約され，双方向変換分野の発展が加速されることも期待される．

本稿では，双方向変換について，古典的なビュー更新問題などの基本的な問題設定から，プログラミング言語などのアプローチ，そしてソフトウェア開発などへの応用，今後の展望について紹介した．本稿を通じて，少しでも双方向変換を身近に感じていただければ幸甚である．

参考文献

1) 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).
2) Bancilhon, F. and Spyratos, N. : Update Semantics of Relational Views, ACM Trans. Database Syst., Vol.6, No.4, pp.557–575 (1981).
3) Meertens, L.: Designing Constraint Maintainers for UserInteraction (1998), http://www.kestrel.edu/~meertens/
4) Matsuda, K., Hu, Z., Nakano, K., Hamana, M. and Takeichi, M. : Bidirectionalization Transformation based on Automatic Derivation of View Complement Functions, ICFP, pp.47–58 (2007).
5) Hidaka, S., Hu, Z., Inaba, K., Kato, H., Matsuda, K. and Nakano, K. : Bidirectionalizing Graph Transformations, ICFP, ACM, pp.205-216 (2010).
6) Yu, Y., Lin, Y., Hu, Z., Hidaka, S., Kato, H. and Montrieux, L. : Maintaining Invariant Traceability through Bidirectional Transformations, 34th International Conference on Software Engineering, pp.540–550 (2012).
7) Pacheco, H., Hu, Z. and Fischer, S. : Monadic Combinators for "Putback" Style Bidirectional Programming, Proc. ACM SIGPLAN 2014 Workshop on Partial Evaluation and Program Manipulation, pp.39-50 (2014).

(2017 年 4 月 1 日受付)

☆11 Families to Persons Case, http://www.transformation-tool-contest.eu/

☆12 http://bx-community.wikidot.com/

日高宗一郎（正会員） hidaka@hosei.ac.jp

1999 年東京大学大学院電子情報工学専攻博士課程修了．同年学術情報センター助手．現在，法政大学教授．大学院情報科学研究科併任．言語処理系，データベースプログラミング言語等の研究に従事．博士（工学）．