日本ソフトウェア科学会第 28 回大会 (2011 年度) 講演論文集

GRoundTram による ATL の双方向化の実現

篠埜 功　胡 振江　日高 宗一郎　稲葉 一浩　加藤 弘之　中野 圭介

モデル変換記述言語 ATL は現在単方向変換のみサポートしている．我々の既存研究により，ある形で記述されたグラフ構造の変換は双方向化できることを示し，それに基づいて GRoundTram と呼ばれるシステムを実装した．ある変換 trans は，組 (trans, trans') がある良い性質を満たすような逆変換 trans' が得られるとき，双方向化されるという．双方向モデル変換はソースモデル上の編集をターゲットモデルに反映させるだけでなく，その逆方向も行うために使われる．我々の最近の既存研究により，実用的双方向モデル変換に向けた第一歩として，ATL を，GRoundTram システムの変換言語である UnQL 言語にエンコードすることにより ATL の核に対する双方向化の基本的枠組みを提示した．現在この枠組に基づきアルゴリズム構築および実装を進めており，本発表ではその途中経過を報告する．

1　はじめに

ATL[16][15][1] はモデル変換の記述に広く使われている言語であり，Eclispe のプラグインとして開発環境が提供されている．ATL のプログラムはソースモデル中の要素をどのように変換してターゲットモデルを構築するかを記述した規則の集まりから成る．規則においては OCL 式を使用でき，さまざまな条件式や算術演算，文字列操作などを記述できる．さらにターゲットモデルを構築する際に各要素間の繋がりを変更することができる．これにより ATL は広範囲のモデル変換を宣言的に記述することができる．

ATL は広く実用に用いられているが，ソースモデルからターゲットモデルへの一方向の変換しか記述できず，QVT や TGG[19] においてサポートされる双方向の変換が現状では記述できない．双方向変換はモデルの同期，整合性検査，リバースエンジニアリングにおいて重要な役割を果たしている [9]．ATL の双方向化への試みは ATL の仮想機械のバイトコードレベルにおいてなされたが [22]，ATL バイトコードに多くの制限が課され，また ATL プログラムを書くユーザが理解して制御するのが難しい側面もある．

このような低レベルの ATL の双方向化とは別のアプローチとして，我々はより高いレベルにおける漸進的な双方向化のアプローチを取る．これは，ATL の核となる部分を双方向化し，他の双方向化できない部分とうまく共存できるようにするというものである．この共存は ATL プログラム中の各規則がソースモデル中の各要素からターゲットモデル中の要素への写像の記述であるという面でモジュール化されていることにより可能になる．核となる部分は将来徐徐に拡張することができ，それに合わせて実際に双方向化できる部分が拡大する．モデル変換の基本単位である ATL の各規則をどのように双方向化するかということが我々のアプローチにおいて核となる部分であるが，既存の双方向変換言語を用いることによりこの問題を解決する．

双方向変換は，元はデータベースのビュー更新問題 [4] において考案され，現在ではプログラミング言語の分野においても注目されている技術である．これまでにいくつかの双方向変換言語が提案され

Bidirectionalization of ATL with GRoundTram
Isao Sasano, 芝浦工業大学, Shibaura Institute of Technology.
Zhenjiang Hu, Soichiro Hidaka, Hiroyuki Kato, 国立情報学研究所, National Institute of Informatics.
Kazuhiro Inaba, 国立情報学研究所，ただし現在はグーグルに所属, National Institute of Informatics, currently in Google.
Keisuke Nakano, 電気通信大学, The University of Electro-Communications.

[7][17][14][20][5][21]，それらにおいては双方向変換を特徴付ける put-get や get-put など，何らかの性質が満たされることが保証されている．しかしながら，これらのうちのほとんどの言語においては木や文字列といったデータ構造しか扱えない．モデルは本質的にはグラフ構造を持つため，ATL の双方向化に用いるには，グラフ構造を扱える双方向変換言語が望まれる．近年，我々の既存研究[11] により，UnQL[8] という良く知られたグラフ問い合わせ言語を双方向グラフ変換言語として用いることができるということを示した．さらに我々は，GRoundTram (Graph Roundtrip Transformation) [2][12] と呼ばれる双方向グラフ変換システムを開発した．このシステムにおいては UnQL 言語を双方向グラフ変換の記述に用いることができる．

筆者等による既存研究[18] により，GRoundTram による ATL の双方向化に関する基本的枠組を提案した．これは，実用的な双方向モデル変換の実現に向けた第一歩であり，モデルをグラフ構造で表現し，ATL の核となる部分を UnQL 言語に変換することにより，ATL の双方向化を行おうとするものである．現在この基本的枠組に従ってアルゴリズム構築および実装を進めており，本発表においてはその途中経過を報告する．

本論文の構成は以下の通りである．2 章で ATL, GRoundTram システム, UnQL 言語の概要を示す．3 章でモデルとグラフ構造間の変換アルゴリズムおよびその実装について述べる．4 章で ATL の規則から UnQL 言語への変換アルゴリズムおよびその実装について述べる．5 章で ATL の核となる部分の双方向化を用いて ATL フルセットに対応するシステムをどのように構築するかについて議論する．6 章で本論文のまとめについて述べる．

2 準備

ここでは ATL，UnQL，および GroundTram システムの概要を示す．

2.1 ATL

本論文では以下の ATL 言語のサブセットを用いて双方向化の本質的部分を示す．このサブセットは ATL の破壊的な部分は含まない．また，議論を簡潔にするため，OCL 式のほどんどの部分を除外している．この ATL のサブセットはフルセットと同等の記述力は持たないが，モデル変換の双方向化への我々のアプローチを示すには十分である．

$$
\begin{array}{rcl}
ATL & = & \textbf{module}\ id; \textbf{create}\ id : id; \\
 & & \textbf{from}\ id : id; rule^{+} \\
rule & = & \textbf{rule}\ id\ \textbf{from}\ inPat\ \textbf{to}\ outPat^{+} \\
inPat & = & id : oclType \\
outPat & = & id : oclType\ (binding^{*}) \\
binding & = & id \leftarrow oclExp \\
oclExp & = & id \\
 & | & id.id \\
 & | & string \\
 & | & oclExp + oclExp
\end{array}
$$

ATL は規則の集まりから成り，それぞれの規則はソースモデル中の要素に適用される変換を記述したものである．規則は上記構文中の $rule$ により表され，$rule$ 中の $inPat$ によって各規則がどの要素に適用されるかが決められる．ATL の詳細については，ATL の web page[1] の文書に記述されている．

ここでは ATL 言語の直感的な意味を図 1 の例を用いて説明する．この例は MTiP2005[6] というワークショップの告知においてモデル変換言語の記述力を測る自明でないベンチマークとして提供された class2RDBMS という例を簡略化したものであり，`Class2Table` および `Attribute2Column` という 2 つの規則から成る．

ATL ではソースモデルとターゲットモデルのメタモデルをそれぞれ指定しなければならない．図 1 の変換におけるソースモデルのメタモデルを図 2 で記述されるメタモデルとし，ターゲットモデルのメタモデルを図 3 で記述されるメタモデルとする．ATL の開発環境では，メタモデルは ECore 図や KM3(kernel meta meta model)[3] によって記述される．図 2 と図 3 のメタモデルは KM3 で記述されている．

ここでは，Person クラスが name と address という 2 つの String 型の属性を持つということを表す，図 4 のモデルをソースモデルの例として用いる．このモデルは，図 1 中の 2 つ規則により，図 5 のターゲットモデルに変換される．

次章以降で，この具体例を用いて ATL の双方向化

```
rule Class2Table {
    from
        s : ClassDiagram!Class
    to
        t : Relational!Table (
            name <- s.name,
            col <- s.attr
        )
}

rule Attribute2Column {
    from
        s : ClassDiagram!Attribute
    to
        t : Relational!Column (
            name <- s.name
        )
}
```

図 1　ATL によるモデル変換の記述例

```
package Class {
    abstract class NamedElt {
        attribute name : String;
    }

    class Class extends NamedElt {
        reference attr[*] :
            Attribute oppositeOf owner;
    }

    class Attribute extends NamedElt {
        reference type : Class;
        reference owner :
           Class oppositeOf attr;
    }
}

package PrimitiveTypes {
    datatype Boolean;
    datatype Integer;
    datatype String;
}
```

図 2　図 1 の変換の入力メタモデルの KM3 による記述

の核となるアイディアを示す.

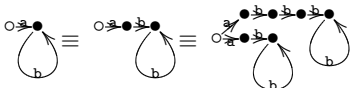
2.2　UnQL

ここでは，UnQL 言語[8] について簡潔に概要を述べる．UnQL はグラフを対象とし，関係データベースの問い合わせに使われる SQL の select-where 構文に相当する構文を持つ．特に，構造再帰 (structural recursion) の構文を持ち，これにより入力グラフを再帰的に辿ることができる．UnQL の形式的な定義は省略し，以下で UnQL の基本的な概念を述べる．

2.2.1　グラフデータモデル

UnQL におけるグラフは根 (root) を持ち，閉路を含んでいてもよい有向グラフであり，辺 (edge) に順序はない．また辺にラベルのあるグラフであり，すべての情報は辺のラベルとして表される．辺のラベルは 123 や 42 のような整数，`"hello"`のような文字列，あるいは `name` や `attr` などの記号列である．

UnQL において，2 つのグラフは，それらが**双模倣** (*bisimilar*) であるとき等しいと考える．直感的には，双模倣とは，閉路を展開したり部分グラフをコピーすることにより得られるグラフは元のグラフと区別せず，また，根から到達できない部分は無視するという概念である．例えば，以下の 3 つのグラフは互いに双模倣である．



UnQL の各構文は双模倣性を保存する．つまり，UnQL で記述された問い合わせの入力として 2 つの双模倣なグラフが与えられたとき，それらの結果は双模倣である．この双模倣の概念は問い合わせ最適化[8] や双方向化[11] において重要な役割を果たす．ユーザが 2 つの双模倣なグラフを区別したい場合は，特別なラベルを持つ**タグ**辺を加えることにより双模倣性が成り立たなくなり，別々のグラフとして扱うことができる．

(query)	Q	::=	$\mathbf{select}\ T\ \mathbf{where}\ B, \ldots, B$
(template)	T	::=	$Q \mid \{L : T, \ldots, L : T\} \mid T \cup T \mid \$G \mid f(\$G)$
		|	$\mathbf{if}\ BC\ \mathbf{then}\ T\ \mathbf{else}\ T$
		|	$\mathbf{let}\,\mathbf{sfun}\ f\ \{Lp : Gp\} = T$
			$\quad \mid f\ \{Lp : Gp\} = T$
			$\quad \ldots$
			$\mathbf{sfun}\ f'\ \{Lp : Gp\} = T$
			$\quad \mid f'\ \{Lp : Gp\} = T$
			$\quad \ldots$
			$\ldots$
			$\mathbf{in}\ T$
(binding)	B	::=	$Gp\ \mathbf{in}\ \$G \mid BC$
(condition)	BC	::=	$\mathsf{not}\ BC \mid BC\ \mathsf{and}\ BC \mid BC\ \mathsf{or}\ BC$
		|	$\mathsf{isEmpty}(\$G) \mid L = L \mid L \neq L \mid L < L \mid L \leq L$
(label)	L	::=	$\$l \mid \mathtt{a}$
(label pattern)	Lp	::=	$\$l \mid Rp$
(graph pattern)	Gp	::=	$\$G \mid \{Lp : Gp, \ldots, Lp : Gp\}$
(regular path pattern)	Rp	::=	$\mathtt{a} \mid \_ \mid Rp.Rp \mid (Rp \mid Rp) \mid Rp? \mid Rp*$

図 6 UnQL の構文

2.2.2 Query Syntax

UnQL の問い合わせの構文を図 6 に示す．Select-where 構文 $\mathbf{select}\ T\ \mathbf{where}\ B, \ldots, B$ が問い合わせの構文の全体である．この構文は，$\mathbf{where}\, B, \ldots, B$ により，条件を満たす部分グラフを変数に束縛し，テンプレート式 T にしたがって結果を構築する．テンプレート式 T について，式 $\{L_1 : T_1, \ldots, L_n : T_n\}$ は L_i をラベルとし，T_i を指す n 個の外向辺 (outgoing edge) を持つ 1 つの新しい頂点を生成する．和 $G_1 \cup G_2$ はグラフ G_1，G_2 の根を 1 つにまとめたグラフを構築する．たとえば，$\{L_1 : g_1\} \cup \{L_2 : g_2\}$ は $\{L_1 : g_1, L_2 : g_2\}$ と等しい．テンプレート式において，キーワード **sfun** を用いることにより，ユーザは次節に定義する**構造再帰**を記述できる．束縛条件部分 B において，ラベルの値を比較したり，正規表現を用いてグラフを辿ることができる．

2.2.3 構造再帰

グラフ上の関数 f は，以下の等式で定義されるとき構造再帰と呼ばれる．

$$
\begin{aligned}
f(\{\}) &= \{\} \\
f(\{\$l : \$g\}) &= e \\
f(\$g_1 \cup \$g_2) &= f(\$g_1) \cup f(\$g_2),
\end{aligned}
$$

ここで式 e は変数 $\$l$，$\$g$，および $f(\$g)$ の形の再帰呼出を含んでいてもよいが，$\$g$ 以外のグラフへの関数 f の適用は含んではならない．1 つ目と 3 つ目の等式はすべての構造再帰で共通であるので，UnQL においてそれらは省略する．2 番目の等式については，ラベルで場合分けをするのが普通であるので，パターンマッチングを用いることができるようになっている．これにより if-then-else の入れ子の使用を避けることができる．例えば，

$$
\begin{aligned}
&\mathbf{sfun}\ f\ (\{\$l : \$g\}) = \\
&\quad \mathbf{if}\ \$l = \mathtt{class}\ \mathbf{then}\ e_1 \\
&\quad \mathbf{else\ if}\ \$l = \mathtt{interface}\ \mathbf{then}\ e_2 \\
&\quad \mathbf{else\ if}\ \$l = \mathtt{int}\ \mathbf{then}\ e_3 \\
&\quad \mathbf{else}\ \ldots
\end{aligned}
$$

のような if-then-else の入れ子は，パターンマッチン

```
package Relational {
    abstract class Named {
        attribute name : String;
    }

    class Table extends Named {
        reference col[*] :
            Column oppositeOf owner;
    }

    class Column extends Named {
        reference owner :
            Table oppositeOf col;
    }
}

package PrimitiveTypes {
    datatype Boolean;
    datatype Integer;
    datatype String;
}
```

図 3　図 1 の変換の出力メタモデルの KM3 による記述

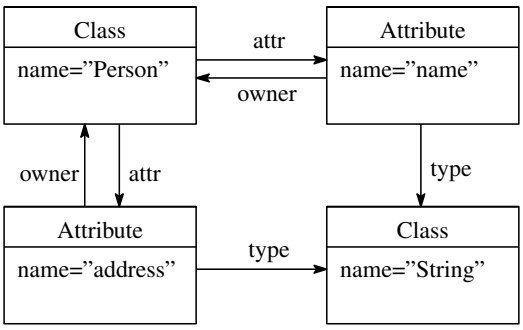


図 4　ソースモデル例

グを使うと

$$
\begin{array}{llll}
\textbf{sfun} & f\ (\{\texttt{class} : \$g\}) & = & e_1 \\
\mid & f\ (\{\texttt{interface} : \$g\}) & = & e_2 \\
\mid & f\ (\{\texttt{int} : \$g\}) & = & e_3 \\
\vdots & & &
\end{array}
$$

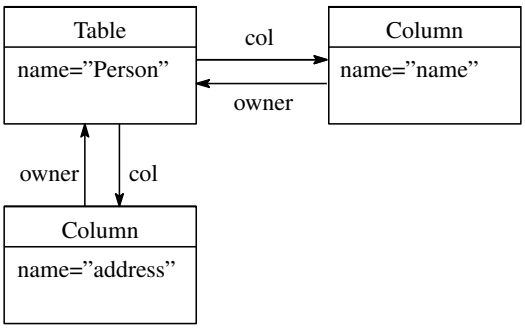


図 5　図 4 のソースモデルに図 1 の規則を適用して得られるターゲットモデル

のように記述することができる．次の例は，構造再帰の単純な例を示す．

$$
\begin{array}{llll}
\textbf{sfun} & a2d\_xc(\{\texttt{a} : \$g\}) & = & \{\texttt{d} : a2d\_xc(\$g)\} \\
\mid & a2d\_xc(\{\texttt{c} : \$g\}) & = & a2d\_xc(\$g) \\
\mid & a2d\_xc(\{\$l : \$g\}) & = & \{\$l : a2d\_xc(\$g)\}
\end{array}
$$

この関数は，引数に受け取ったグラフにおいて，a というラベルの付いた辺のラベルをすべて d で置き換え，c というラベルのついた辺を縮約 (contract) したグラフを返す．

構造再帰はこのように単純であるが，自明でないモデル変換を記述するのに十分な記述力を持つ [13].

2.2.4　双方向変換の意味

通常，問い合わせは一方向でのみ実行される．つまり，入力の環境 (変数からグラフへの写像)ρ が与えられたとき，問い合わせ Q は評価され，結果のグラフ $G = \mathcal{F}[\![Q]\!]^{\rho}$ を生成する．ユーザが結果のグラフ G を編集し，G' になった状況を考える．例えば，新しい部分グラフを追加したり，ラベルを変更したり，あるいは辺を削除したりできる．我々の既存研究 [11] において，結果のグラフ上の編集を元の入力グラフに適切に反映させる，**逆方向変換の意味**を与えた．より形式的には，編集された結果のグラフ G' と入力の環境 ρ が与えられたとき，更新後の環境 $\rho' = \mathcal{B}[\![Q]\!]^{\rho}_{G'}$ を計算できる．

上記において，“適切に反映” というのは，次の 2 つの性質が成り立つことである．

$$\mathcal{F}[\![Q]\!]^{\rho} = G \text{ ならば } \mathcal{B}[\![Q]\!]^{\rho}_{G} = \rho \quad \text{(GetPut)}$$

$$\mathcal{B}[\![Q]\!]^{\rho}_{G'} = \rho' \text{ ならば } \mathcal{B}[\![Q]\!]^{\rho}_{\mathcal{F}[\![Q]\!]^{\rho'}} = \rho' \quad \text{(WPutGet)}$$

性質 (GETPUT) は，結果のグラフ G に何の編集もなされなかった場合，入力の環境は変化しないということを表す．性質 (WPUTGET) は，[10] で提示された性質 (PUTGET) を弱めたものである．性質 (PUTGET) は $G' \in$ Range($\mathcal{F}[\![Q]\!]$) かつ $\mathcal{B}[\![Q]\!]^{\rho}_{G'} = \rho'$ ならば $\mathcal{F}[\![Q]\!]^{\rho'} = G'$ が成り立つという性質である．性質 (PUTGET) は，結果のグラフが G' に編集され，それが順方向変換の値域に入っていた場合，この編集は，入力グラフに対し，順方向変換をもう一度適用した場合に同じグラフ G' を生成するように反映されるということを表す．これに対し，性質 (WPUTGET) は，編集された結果のグラフと，逆方向変換ののち順方向変換をもう一度行った場合に得られる結果は異なっていてもよいが，逆方向変換をもう一度適用した場合に，一度目の逆方向変換の適用と同じ結果が得られるということを表す．

2.3 GRoundTram システム

我々の既存研究により，グラフ構造上で双方向変換を UnQL 言語で記述することのできる GRoundTram というシステムを開発した．GRoundTram システムにおいては，図 7 に示されるように，双方向評価器によりソースグラフとターゲットグラフの間で双方向に評価が行われる．

図 8 に GRoundTram システムのスクリーンショットを示す．ユーザはまず左側にあるソースグラフと UnQL で記述された変換を読み込む．オプションでソースメタモデルとターゲットメタモデルを KM3 で記述することもできる．これらを読み込んだのち，forward ボタン (右矢印アイコン) を押すことにより順方向変換が行われ，ターゲットグラフが右側に表示される．ユーザはグラフィカルにターゲットグラフを編集でき，そののち backward ボタン (左矢印アイコン) を押すことにより逆方向変換が行われる．ターゲットグラフだけでなく，ソースグラフも編集することができる．ソースモデルとターゲットモデルがそれぞれのメタモデルに適合しているかどうかを，左右にある check アイコンを押すことにより検査することができる．変換自体も**静的に**検査することができる．これは，与えられたソースメタモデル，ターゲットメタモデル，および変換に対し，ターゲットグラフが**常に**与えられたターゲットメタモデルに適合することを検査できるという意味である．もし適合しない場合がある場合は，その具体例のグラフが表示される．

図 8 はソースとターゲットの間のトレース情報も赤色で示している．ユーザが左右どちらかにおいて部分グラフを選択すると，他方の対応する部分グラフに色が付けられる．これにより，ターゲットグラフのどの部分を編集したらソースグラフのどの部分に影響するか，またその逆方向についても予測するのに役立つ．

3 モデルとグラフ構造間の変換

GRoundTram システムを使うためには，モデルをグラフでエンコードし，変換結果のグラフをデコードしてモデルに戻す必要がある．ここではエンコーディングアルゴリズムを与える．デコーディングアルゴリズムはエンコーディングの逆であり，省略する．我々が提案する ATL の核に対する双方向変換の全体像は図 10 のようにまとめられる．

モデルからグラフへエンコードするアルゴリズムを図 9 に示す．図 9 において，得られるグラフは UnCAL [8] という言語を用いて記述している．ここでは UnCAL 言語についての説明はせず，アルゴリズムの直感的意味を示す．

まず，モデル中の各箱は識別子 n，フィールド，他の箱への辺を持ち，$(n, \{f_1, \ldots, f_k\})$ のように記述する．各 f_i はフィールドか，あるいは他の箱への辺を表す．フィールドは $l = atom$ という形で表記し，これはフィールド名が l で中身が $atom$ であるようなフィールドを表す．他の箱への辺は $\mathbf{reference}\, l \to s_j$ という形で表記し，これは l というラベルを持つ，箱 s_j への辺を表す．

図 9 のアルゴリズムは，上記構造の箱から成るモデルに対し適用される．図 9 において，関数 $trans$ は以下のように定義される関数である．

$$trans\,(l = atom) = \{l : \{atom\}\}$$

$$trans\,(\mathbf{reference}\, l \to s_j) = \{l : \&m_j\}$$

関数 $trans$ は引数としてフィールドあるいは他の箱への辺を受け取り，フィールドの場合は，フィールド名をラベルとして持つ辺と $atom$ をラベルとして持つ

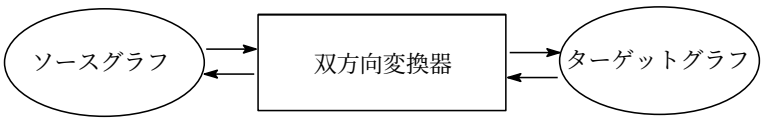


図 7　GRoundTram システム

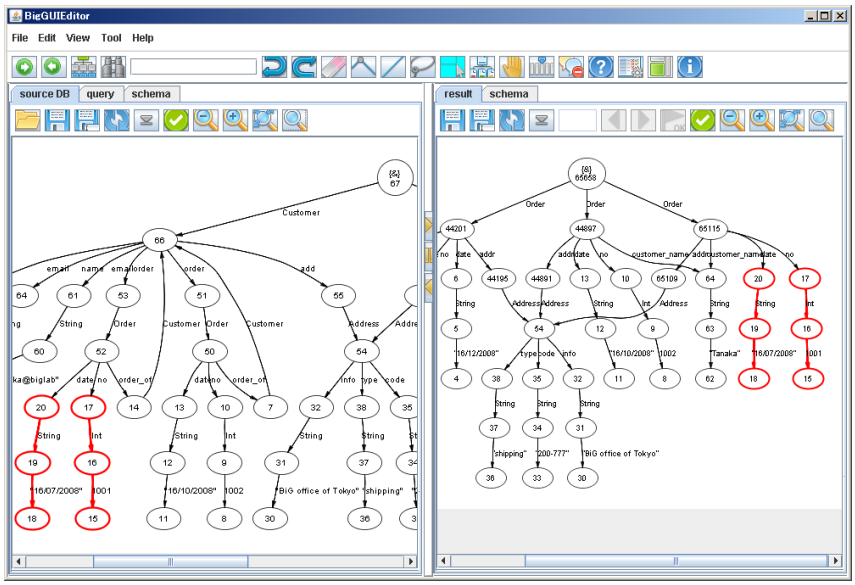


図 8　GRoundTram システムのスクリーンショット

辺からなるグラフへエンコードし，他の箱への辺の場合は，箱 s_j のエンコード結果のグラフの根に対応するマーカー $\&m_j$ を指す，l をラベルとする辺から成るグラフへエンコードする．

図 9 のアルゴリズムは，上記関数 $trans$ を用いて定義されており，入力モデル図中の箱に番号を付け，各箱を訪問し，各箱 s_i をグラフ g_i へエンコードし，それらのグラフの根およびそれらへの外向辺を持つ頂点から成るグラフを g_0 とし，それらを cycle 演算子で 1 つのグラフとしてまとめあげるものである．グラフ $g_0, g_1, \ldots, g_n$ はばらばらであるが，マーカーとよばれる印 $\&m_1, \ldots, \&m_n, \&src$ が付けられており，cycle 演算子を用いることにより，同じマーカー同士の間に ϵ 辺が張られる．

このアルゴリズムによるエンコード処理を，図 4 のモデルを使って説明する．まず，図 4 の左上の箱は，以下のような UnQL グラフにエンコードされる．

$$
\begin{aligned}
g_1 = \&m_1 := \{&\text{ClassName} : \{\text{Class} : \\
&\{\text{name} : \{''\text{Person}'' : \{\}\}, \\
&\text{attr} : \&m_2, \\
&\text{attr} : \&m_3\}\}\}
\end{aligned}
$$

ここで，デコーディングでモデルに戻すことができるようにするために，定数ラベル ClassName が用いられている．マーカー $\&m_2$ および $\&m_3$ は，2 つの Attribute クラスを以下のようにエンコードしたグラフ g_2 および g_3 の根に対応する．

$$
\begin{aligned}
g_2 = \&m_2 := \{&\text{ClassName} : \{\text{Attribute} : \\
&\{\text{name} : \{''\text{name}'' : \{\}\}, \\
&\text{owner} : \&m_1, \\
&\text{type} : \&m_4\}\}\}
\end{aligned}
$$

(1) 入力モデル図中のすべての箱に対して，1 から n までの自然数を割り当て，それらを $s_1, \ldots, s_n$ とおく．
(2) s_1 から s_n の箱を訪問する．各箱 s_i に対し，
$$(n, \{f_1, \ldots, f_k\}) = s_i,$$
$$g_i = \&m_i := \{\text{className} : \{n : (trans\ (f_1) \cup \cdots \cup trans\ (f_k))\}\},$$
$$g_0 = \&src := \{\epsilon : \&m_1, \ldots, \epsilon : \&m_t\}$$
とおく．
(3) $\&src \mathbin{@} cycle\ (g_0 \oplus g_1 \oplus \cdots \oplus g_n)$ を返す．

図 9　モデルからグラフへのエンコーディングアルゴリズム

$$g_3 = \&m_3 := \{\text{ClassName} : \{\text{Attribute} : \{\text{name} : \{''\text{address}''\} : \{\}\}, \text{owner} : \&m_1, \text{type} : \&m_4\}\}\}$$

マーカー $\&m_4$ は図 4 の右下の箱を以下のようにエンコードしたグラフ g_4 の根に対応する．

$$g_4 = \&m_3 := \{\text{ClassName} : \{\text{Class} : \{\text{name} : \{''\text{String}''\} : \{\}\}\}\}$$

最後に，得られたグラフ $g_1, \ldots, g_4$ の根およびそれらへの外向辺を持つ頂点から成るグラフ g_0 を

$$g_0 = \&src := \{\epsilon : \&m_1, \ldots, \epsilon : \&m_t\}$$

のように生成し，以下のように 1 つのグラフとしてまとめる．

$$\&src \mathbin{@} cycle\ (g_0 \oplus g_1 \oplus \cdots \oplus g_n)$$

これが，図 4 のモデルのエンコード結果のグラフを表す．

4　ATL 規則の UnQL への変換

ここでは与えられた ATL プログラムを UnQL へ変換するアルゴリズムを示す．ATL プログラムは，2.1 節で示したように規則の集まりから成る．我々の変換の戦略は，各 ATL 規則を，ATL 規則名を使って UnQL 言語の **sfun** による構造再帰の構文へ変換することである．

変換アルゴリズムは，以下のように ATL 言語の構造に沿って設計する．トップレベルの変換関数は *atl2unql* である．

atl2unql (**rule** r **from** *inPat* **to** *outPatSeq*) =
　sfun f (*inPat2arg inPat*) =
　　outPatSeq2unql outPatSeq
　　| f ({\$$l$: \$$g$}) = {\$$l$: r (\$$g$)}
　and
　sfun r ({ClassName : \$$g$}) =
　　{ClassName : f (\$$g$)}
　　| r ({\$$l$: \$$g$}) = {\$$l$: r (\$$g$)}

この関数は，ATL プログラム中の各規則に適用され，1 つの規則に対して 2 つの相互再帰 UnQL 関数を生成する．関数を生成する際，受け取った規則の名前 r を変換後の 2 つの関数のうちの 1 つの関数の名前として用い，もう 1 つの関数は名前 f を生成して作成する．*inPat* は規則がどの要素に適用されるかを決めるパターンである．このパターンは，*inPat2arg* を適用することにより変換し，生成される UnQL 関数 f の引数部分のパターンとなる．ATL 中のパターン $s : A$ は逆にして $\{A : \$s\}$ というパターンにエンコードする．記号 \$ は単に UnQL 言語において，パターン変数を記述する際に用いられる．

$$inPat2arg\ (s : A) = \{A : \$s\}$$

3 章で述べたように，モデル中の各要素はグラフ構造としてエンコードされるが，その際，関数がパターンマッチングによりエンコードされた要素を見付けられるようにする．このために，ClassName という定数ラベルを用いて各要素をエンコードした．これに対応し，関数 r は ClassName という定数パターンを含むパターンを受け取ったときに関数 f を呼び出す関数として生成する．

変換関数 *outPatSeq2unql* は ATL 規則中の出力パ

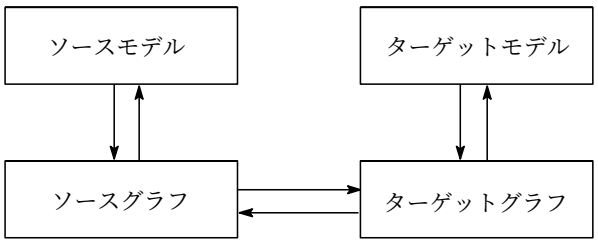


図 10　ATL の核に対する双方向変換の全体像

ターン列 *outPatSeq* に対して適用される．出力パターン列 *outPatSeq* 中の各パターン *outPat* は，生成されるターゲットモデルの各要素を決める．1 つの規則はソースモデル中の 1 つの要素からターゲットモデル中の 1 つ以上の要素を生成できる．ただし図 1 の例では 1 つの要素のみ生成している．出力パターン $t\ :\ c\ (binds)$ 中の変数 t はそれぞれターゲットモデル中のある要素に束縛されており，他の出力パターンを含む出力パターン列中で使用できる．出力パターン $t\ :\ c\ (binds)$ 中の変数 t はエンコードされた UnQL 関数中でローカルな相互再帰関数の関数名として用いる．これらのローカルな相互再帰関数はダミーの引数を取り，そのうちの 1 つの関数 t_1 がダミーのグラフ $dummy\ :\ \{\}$ に適用される．

$$
\begin{aligned}
&outPatSeq2unql\ (t_1\ :\ c_1\ (binds_1),\\
&\qquad\qquad t_2\ :\ c_2\ (binds_2),\ \ldots) =\\
&\quad \textbf{letrec}\\
&\qquad \textbf{sfun}\ t_1\ (\{\$dummy\ :\ \{\}) =\\
&\qquad\quad outPat2unql\ (c_1\ (binds_1))\\
&\qquad \textbf{and}\\
&\qquad \textbf{sfun}\ t_2\ (\{\$dummy\ :\ \{\}) =\\
&\qquad\quad outPat2unql\ (c_2\ (binds_2))\\
&\qquad \ldots\\
&\qquad \textbf{in}\ t_1\ (dummy\ :\ \{\})
\end{aligned}
$$

関数 *outPat2unql* は，以下のように，各出力パターンの *id* 以外の部分に適用される．

$$
\begin{aligned}
&outPat2unql\ (c\ (bind_1,\ bind_2,\ \ldots)) =\\
&\quad \{c\ :\ (bind2unql\ bind_1) \cup\\
&\qquad\quad (bind2unql\ bind_2) \cup\\
&\qquad\quad \ldots\}\\
&bind2unql\ (m \leftarrow oclExp) =\\
&\quad \textbf{select}\ \{m\ :\ \$g\}\\
&\quad \textbf{where}\ oclExp2unqlBinds\ \$g\ oclExp
\end{aligned}
$$

各束縛構文の右辺は OCL 式のサブセットである．束縛は以下で定義される関数 *oclExp2unqlBinds* により UnQL の select-where 構文の where 節へ変換される．

$$
\begin{aligned}
&oclExp2unqlBinds\ p\ v = p\ \textbf{in}\ \$v\\
&oclExp2unqlBinds\ p\ (vs\ .\ v) =\\
&\quad oclExp2unqlBinds\ \$g\ vs\ (\$g\ :\ \text{fresh})\\
&\quad \{v\ :\ p\}\ \textbf{in}\ \$g\\
&oclExp2unqlBinds\ p\ string = p\ \textbf{in}\ \{string\ :\ \{\}\}\\
&oclExp2unqlBinds\ p\ (e1 + e2) =\\
&\quad oclExp2unqlBinds\ \{\$l1\ :\ \{\}\}\ e1\\
&\quad oclExp2unqlBinds\ \{\$l2\ :\ \{\}\}\ e2\\
&\quad p\ \textbf{in}\ \{\$l1\ \texttt{++}\ \$l2\ :\ \{\}\}
\end{aligned}
$$

ドットで区切られた識別子の列 $vs\ .\ v$ は新しい変数を生成しつつ束縛の列へエンコードされる．文字列の結合は UnQL の文字列の結合++へエンコードされる．

上記アルゴリズムを図 1 の ATL の例に適用すると図 11 の UnQL 関数 `Class2Table` および `Attribute2Column` を得る．これらは関数 `f1` および `f2` とともに相互再帰的に定義されている．あとは，これらの関数を，ソースモデルをエンコードして得られたグラフに対し，順に

```
Attribute2Column  (Class2Table ($db))
```

のように適用すればよい．図 11 全体は UnQL による問い合わせであり，GRoundTram システムにより，エンコードされたグラフ上で双方向変換を行うことができる．`$db` は入力グラフ，すなわちエンコードされたソースモデルを表す．

上記のアルゴリズムは OCaml 言語で実装した．このアルゴリズムの計算量は入力モデルと規則の記述の大きさに関して線形である．

5 ATL の核に対する双方向化によるシステム構築手法

ここでは，ATL の核となる部分の双方向化を用いることにより ATL フルセットに対応するシステムをどのように構築するかについて議論する．ATL は宣言的な構文のみでなく破壊的な構文も含んでおり，フルセットすべてを双方向化に対応させることはできない．前章までに，ATL の核に関して双方向化手法を示したが，これを拡張していくだけでは限界があり，ATL フルセットに対応するシステムを構築するためには，本質的に双方向化できない部分に対して何らかの対処をしなければならない．以下ではこれに対して 3 つの対処法を提示する．

- ターゲットモデルとのマッチング
  既存の ATL システムで変換して得られるモデルと，双方向化に対応した ATL の核に含まれている規則のみを我々の枠組でソースモデルに適用して得られるモデルとのマッチングをとり，我々の枠組で得られた部分モデル上でのみ編集を許し，更新が反映された部分ソースモデルを元のソースモデルに埋め込む．
- ATL システムのリンクの利用
  ATL システムではソースモデルとターゲットモデルの対応する要素間をリンクしており，これを利用する．上に記したターゲットモデルとのマッチングと相補的に用いることができると考えられる．
- GRoundTram システムの拡張
  本質的に双方向化できない関数 f に対し，以下のように表面上双方向化された関数を構築する．

$$uni_F(f)\ (x) = f(x)$$
$$uni_B(f)\ (x, v) = \mathbf{if}\ v = f(x)\ \mathbf{then}\ ok\ \mathbf{else}\ error$$

この手法では，GRoundTram システム自体を拡張する必要があるが，最も現実的な手法と考えられる．

6 まとめ

本論文では ATL の双方向化に向けて，ATL のサブセットに対する双方向化を示し，さらに双方向化していない ATL の残りの部分との共存の方針について示した．現状では双方向化されるサブセットは小さいが，今後拡張することによって，実際に双方向化される部分を拡大していくことができる．OCaml によるプロトタイプ実装は `http://www.biglab.org/src/jssst11/index.html` で取得可能である．

謝辞 ATL 規則の単純な例を提供してくださった Massimo Tisi 氏および Frederic Jouault 氏に感謝する．本研究の一部は国立情報学研究所のグランドチャレンジプロジェクト「Linguistic Foundation for Bidirectional Model Transformation」，および科学研究費補助金 基盤研究 (B) 22300012，基盤研究 (C) 20500043，若手研究 (B) 20700035 の補助を得て行われた．

参考文献

[1] The ATL web site. `http://www.eclipse.org/m2m/atl/`.

[2] The BiG project web site. `http://www.biglab.org/`.

[3] ATLAS group. KM3: Kernel MetaMetaModel manual. `http://www.eclipse.org/gmt/atl/doc/`.

[4] François Bancilhon and Nicolas Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981.

[5] Davi M. J. Barbosa, Julien Cretin, Nate Foster, Michael Greenberg, and Benjamin C. Pierce. Matching lenses: Alignment and view update. In *ACM SIGPLAN International Conference on Functional Programming*, pages 193–204. ACM, 2010.

[6] Jean Bezivin, Bernhard Rumpe, Andy Schürr, and Laurence Tratt. Model transformation in practice workshop announcement. In *Satellite Events at the MoDELS 2005 Conference*, pages 120–127. Springer-Verlag, 2005.

[7] Aaron Bohannon, Benjamin C. Pierce, and Jeffrey A. Vaughan. Relational lenses: a language for updatable

```
select
letrec
sfun f1 ({Class : $s}) =
        letrec sfun t ({$dummy : {}}) =
                {Table :
                    (select {name : $a}
                     where $b in $s,
                           {name : $a} in $b)
                 U  (select {col : $c}
                     where $d in $s,
                           {attr : $c} in $d)}
        in t ({dummy : {}})
   | f1 ({$l: $s}) = {$l : Class2Table ($s)}
and
sfun Class2Table ({ClassName : $g}) = {ClassName : f1 ($g)}
   | Class2Table ({$l : $g}) = {$l : Class2Table ($g)}
and
sfun f2 ({Attribute : $s}) =
       letrec sfun t ({$dummy : {}}) =
               {Column:
                   (select {name : $a}
                    where $b in $s,
                          {name : $a} in $b)}
       in t ({dummy:{}})
   | f2 ({$l: $s}) = {$l : Attribute2Column ($s)}
and
sfun Attribute2Column ({ClassName : $g}) = {ClassName : f2 ($g)}
   | Attribute2Column ({$l : $g}) = {$l : Attribute2Column ($g)}
in
  Attribute2Column  (Class2Table ($db))
```

図 11　図 1 の ATL の例を UnQL 関数へエンコードした結果

views. In Stijn Vansummeren, editor, *PODS*, pages 338–347. ACM, 2006.

[8] Peter Buneman, Mary F. Fernandez, and Dan Suciu. UnQL: a query language and algebra for semistructured data based on structural recursion. *VLDB Journal: Very Large Data Bases*, 9(1):76–110, 2000.

[9] Krzysztof Czarnecki, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *International Conference on Model Transformation (ICMT 2009)*, pages 260–283. LNCS 5563, Springer, 2009.

[10] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. In *POPL '05: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages*, pages 233–246, 2005.

[11] Soichiro Hidaka, Zhenjiang Hu, Kazuhiro Inaba, Hiroyuki Kato, Kazutaka Matsuda, and Keisuke Nakano. Bidirectionalizing graph transformations. In *ACM SIGPLAN In-*

ternational Conference on Functional Programming, pages 205–216. ACM, 2010.

[12] Soichiro Hidaka, Zhenjiang Hu, Kazuhiro Inaba, Hiroyuki Kato, and Keisuke Nakano. GRoundTram: An integrated framework for developing well-behaved bidirectional model transformations (short paper). In *26th IEEE/ACM International Conference On Automated Software Engineering*, 2011.

[13] Soichiro Hidaka, Zhenjiang Hu, Hiroyuki Kato, and Keisuke Nakano. Towards a compositional approach to model transformation for software development. In *SAC'09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 468–475, New York, NY, USA, 2009. ACM.

[14] Zhenjiang Hu, Shin-Cheng Mu, and Masato Takeichi. A programmable editor for developing structured documents based on bidirectional transformations. *Higher-Order and Symbolic Computation*, 21(1-2):89–118, 2008.

[15] Frederic Jouault, Freddy Allilaire, Jean Bezivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, 2008.

[16] Frederic Jouault and Ivan Kurtev. Transforming models with ATL. In *Proceedings of Satellite Events at the MoDELS 2005 Conference*, pages 128–138. LNCS 3844, Springer, 2006.

[17] Kazutaka Matsuda, Zhenjiang Hu, Keisuke Nakano, Makoto Hamana, and Masato Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions. In *12th ACM SIGPLAN International Conference on Functional Programming (ICFP 2007)*, pages 47–58. ACM Press, October 2007.

[18] Isao Sasano, Zhenjiang Hu, Soichiro Hidaka, Kazuhiro Inaba, Hiroyuki Kato, and Keisuke Nakano. Toward bidirectionalization of ATL with GRoundTram. In *Theory and Practice of Model Transformations, Fourth International Conference, ICMT 2011*, volume 6707 of *LNCS*. Springer, June 2011.

[19] Perdita Stevens. Bidirectional model transformations in QVT: Semantic issues and open questions. In Gregor Engels, Bill Opdyke, Douglas C. Schmidt, and Frank Weil, editors, *Proc. 10th MoDELS*, volume 4735 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007.

[20] Janis Voigtländer. Bidirectionalization for free! (pearl). In *POPL '09: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages*, pages 165–176, New York, NY, USA, 2009. ACM.

[21] Janis Voigtländer, Zhenjiang Hu, Kazutaka Matsuda, and Meng Wang. Combining syntactic and semantic bidirectionalization. In *ACM SIGPLAN International Conference on Functional Programming*, pages 181–192. ACM, 2010.

[22] Yingfei Xiong, Dongxi Liu, Zhenjiang Hu, Haiyan Zhao, Masato Takeichi, and Hong Mei. Towards automatic model synchronization from model transformations. In *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, pages 164–173. ACM Press, November 2007.