半構造データによるデータベースのための対応データモデルと問い合わせ言語CRQLの提案

（平成15年11月28日　原稿受付）

工学研究科　横尾徳保
工学研究科　重松保弘

Proposal of Correspondence Data Model and the Query Language CRQL for Databases Consisting of Semi-structured Data

by Noriyasu YOKOO
Yasuhiro SHIGEMATSU

Abstract

We propose Correspondence Data Model and the query language CRQL for databases consisting of semi-structured data. The Correspondence Data Model naturally conforms to semi-structured data, and CRQL has powerful "add operation" and declarative "path expression" as typical features. Semi-structured data is inappropriate to traditional database management system, because the structure of semi-structured data is frequently not rigid and not complete. Most semi-structured data is considered to be network structure, therefore the data models which have good compatibility with network structure are suitable for semi-structured databases. In Correspondence Data Model, data is represented by labeled correspondence which can be considered to be equal to labeled directed graph. Labeled correspondence consists of set of three-tuple (label, initial vertex, terminal vertex). The correspondence algebra enables systematic data manipulation. The powerful "add operation" and declarative "path expression" in CRQL release users from troublesome operations.

1．はじめに

近年，半構造データと呼ばれる"schema-less"というキーワードで表現されるようなデータの研究が盛んに行われている[1),2)]．半構造データにおいてはその構造が不規則であったり，また既知ではないなど，従来のデータベースシステム（リレーショナルデータベース）が要求する固定的かつ規則的で完全なデータという条件が満たされないことが多いため，その適用が困難である．半構造データのためのデータモデルとしては，OEM（Object Exchange Model）[3),4)]やEGM（Edge-labeled Graph Model）[5),6)]が有名である．例えば，OEMに基づいたデータベースシステムの問い合わせ言語であるLorelにおいては，(i)オブジェクトの型に依存しない比較やワイルドカード等の導入により，厳格な型付けが緩和され，また(ii)正規表現とパス表現の導入により宣言的な問い合わせができるようになったことから，半構造データに柔軟に対応できるようになっている．しかし，これらのデータモデルにおいては，オブジェクト参照によりグラフ構造が表現できるものの，主たる構造はルートを持つ木構造である．これに対して，半構造データは一般的にはグラフ構造であるといえる．

そこで，本論文では半構造データによるデータベースのための対応データモデルと問い合わせ言語CRQL（CoRrespondence Query Language）を提案する．対応データモデルはグラフ構造を持つデータを概念理解そのままに記号化することができ，またスキーマの動的な変化に柔軟に対処することができる木構造を全く意識しないデータモデルである[7)]．対応データモデルでは実世界の構造記述にラベル付き対応を用い，これは実質的にラベル付き有向グラフで実世界の構造記述を行うことと等価である．また，対応データモデルではリレーショナルデータモデルと同じようにデータ操作言語を定義しており，柔軟かつ簡便なデータの操作を体系的に実現している．また，CRQLは対応データモデルの特性を考慮して，問い合わせを簡潔に記述できるように工夫して設計されており，特にデータの追加およびパス表現が特徴的な問い合わせ言語である．また，スキーマフリーである

ことから，スキーマの更新履歴管理機能を持つ．

以降，対応とラベル付き対応における用語を定義した後，対応データモデルの概要について述べ，データ操作言語の記述例から定義した対応代数の表現能力や記述性について議論する．また，問い合わせ言語 CRQL の特徴について述べ，その有用性を示す．

2．ラベル付き対応

対応データモデルでは，実世界の構造記述にラベル付き対応を用いる．まず，ラベル付き対応の概念についての説明と用語の定義を行う．

2.1 対応とラベル付き対応

A と B をある集合とする．ある規則 f によって，A の元 a に対してそれぞれ 1 つずつ B の部分集合 $f(a)$ が定められる（$a \neq a'$ に対して，$f(a)$ と $f(a')$ に同じ要素が含まれてもよい）とき，その規則 f のことを A から B への対応と呼び，A の元 a に対して定まる B の部分集合 $f(a)$ を f による a の像と呼ぶ．また，f が A から B への対応であることを式 1 のように表現する．一般に，対応 f におけるすべての像が B の元であるとき，f は写像と呼ばれる．

$$f: A \rightarrow B \qquad (1)$$

この対応 f にラベル集合 L を付与し，ラベル $l(\in L)$ 毎に異なる対応 $f\{l\}$ として扱えるようにしたものをラベル付き対応と呼び，式 2 のように表現する．

$$f\{L\}: A \rightarrow B \qquad (2)$$

2.2 ラベル付き対応とラベル付き有向グラフ

式 2 における L，A，B，をそれぞれラベル付き有向グラフのラベル付き矢（以降，ラベル矢）の集合，始点の集合および終点の集合ととらえることで，ラベル付き対応はラベル付き有向グラフとみなすことができ，相互の変換を容易に行うことができる．ラベル付き対応の例として，ラベル付き対応“K高校のある生徒の家族”を図 1 に示す．なお，ラベル付き対応を概念的にとらえる場合は，ラベル付き有向グラフ表現を用いることにし，有向グラフ表現した対応を単に対応と呼ぶ．また，以降はラベルのない対応は扱わないため，“ラベル付き”は省略することにする．

3．対応データモデル

対応データモデルでは実世界を対応の集合とみなし，その対応に対するデータの操作を対応代数で行う．ここでは，対応による構造記述，対応代数，および対応の追加，削除，修正操作について述べる．前述した通り，対応はそれと等価な有向グラフとして表現でき，逆に有向グラフを対応として表現することができる．したがって，実世界を有向グラフで表現することができれば，それがそのまま対応で表現できることになる．

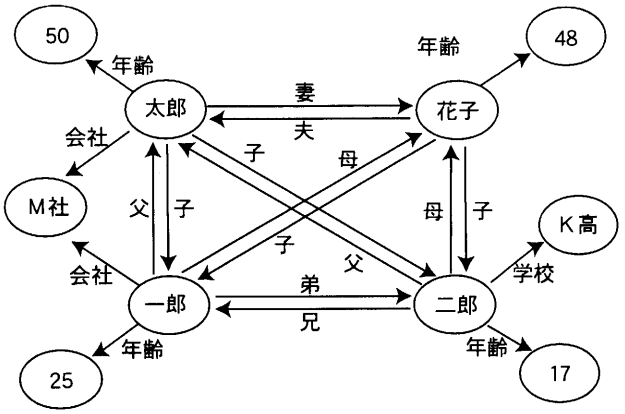

図 1　ラベル付き対応“K高校のある生徒の家族”

3.1 対応による構造記述

対応 $f\{L\}: A \rightarrow B$ をラベル $l \in L$，始点 $a \in A$ および終点 $b \in B$ の組の集合と考え，式 3 のように定義する．対応における 1 組のラベル，始点，終点のことを対応要素，対応要素の集合のことを対応と呼び，対応の集合を対応データベースと呼ぶ．また，始点および終点のデータは，データが自己記述形式でデータの型情報を持つものとし，データの型には制限を設けない．集合型や組型などの型も許し，また対応が入れ子構造になることも許す．ただし，ラベルのデータ型は文字列型に限定し，またデータの一意性はユーザが保証するものとする．

$$f\{L\}: A \rightarrow B = \{(l, a, b) \mid f\{l\}(a) = b,\ l \in L,\ a \in A,\ b \in B\} \qquad (3)$$

対応データモデルは次の特徴を持つ．

(1) グラフ構造を概念理解そのままに構造化できる．

(2) ノード（始点または終点）毎に独自のラベル矢（ラベル）を持つことができる．

(3) 各ノードは同一ラベルのラベル矢を複数持つことができ，また任意のノードに向かう同一ラベルのラベル矢が複数存在してもよい．

(1)の特徴から，グラフ構造を持つデータの自然な構造化が可能になり，同時にデータの信頼性が向上する．また(2)の特徴により，例えば“ある学校の生徒”に関するデータベースを作成するとき，生徒毎に異なったラベルを定義することが可能になる．さらに(3)の特徴により，1 対多の関係をそのまま表現することができ，例えば“趣味”のような複数の事柄が該当するような場合も問題

が生じない．また，このとき任意のノードに向かうラベル矢にも制約がかからない．

スキーマ　対応データモデルでは同種のノードにおいてもノード毎に固有のラベルを持たせることができる．これは，対応のスキーマ（どのノードがどのようなラベルを持つかという対応の枠組）が予め決定されるのではその柔軟性が損われてしまうためである．例えば，図1において“二郎が柔道をしていて三段である”ことが明らかになった場合，その事実を即座に追加できると非常に効果的であるが，そのためにはデータベース設計時には予期できないラベルの追加が必要になり，対応のスキーマが予め固定されてしまうとこのような操作が不可能になる．

3.2　データ操作言語

対応データベースでは，実世界の情報を対応を用いてモデル化／記号化し，計算機に取り込む．このデータベースに対し，任意の対応やノードまたはラベルを取り出したり，データの追加，あるいは不要になったデータの削除を行うために，対応に対するデータ操作言語が必要となる．そのデータ操作言語として対応代数を定義した．対応代数においては，表1に示す対応演算を規定した．以降，対応演算をその演算結果から2種類に分類して説明する．

3.2.1　演算結果が対応の演算

対応和演算　対応$f\{L1\}$と対応$g\{L2\}$の和$f\{L1\}\cup g\{L2\}$とは，$f\{L1\}$と$g\{L2\}$の対応要素の和集合を求める演算であり，式4のように定義する．なお，対応差演算に関しての定義は割愛する．

$$f\{L1\}\cup f\{L2\}=\{(l,\ a,\ b)\ |\ (l,\ a,\ b)\in f\{L1\}\vee(l,\ a,\ b)\in g\{L2\}\}\quad(4)$$

対応共通演算　対応共通演算とは，2つの対応$f\{L1\}$と対応$g\{L2\}$における対応要素の共通集合を求める演算であり，定義式は割愛する．なお，対応共通演算は対応差演算を用いて表現できる．

選択演算　選択演算とは，対応からラベル，始点または終点が条件を満たす対応を生成する演算であり，ラベルに対して条件を課すlabel選択，始点に対して条件を課すdomain選択および終点に対して条件を課すrange選択の3種類の演算を定義する．ただし，条件判定の真偽は常に定まるものとする．label選択演算，domain選択演算およびrange選択演算は，それぞれ式5，式6，式7のように定義する．なお，定義においてθは2項の関係演算子であり，Cは対応のラベル，始点または終点との比較の対象である．

選択演算は，対応からある特定の対応を抽出する演算であり，これは対応データベースに対するデータ操作を行う場合に最も基本となる操作である．また，label選択演算は条件を満たすラベル集合L'を予め生成しておくことで，対応の定義から$f\{L'\}$と表現することができる．

$$f[\{L\,\theta\,C\}]=\{(l,\ a,\ b)\,|\,(l,\ a,\ b)\in f\{L\}\wedge l\,\theta\,C\}\quad(5)$$

$$f[\theta\ C\{L\}]=\{(l,\ a,\ b)\,|\,(l,\ a,\ b)\in f\{L\}\wedge a\,\theta\,C\}\quad(6)$$

$$f[\{L\}\,\theta\,C]=\{(l,\ a,\ b)\,|\,(l,\ a,\ b)\in f\{L\}\wedge b\,\theta\,C\}\quad(7)$$

表1　対応演算の一覧とその説明

演算結果	分類	演算名	説明
対応	二項演算	対応和演算	対応（要素）の和集合を求める
		対応差演算	対応（要素）の差集合を求める
		対応共通演算	対応（要素）の共通集合を求める
	単項演算	label選択演算	ラベルが条件を満たす対応を求める
		domain選択演算	始点が条件を満たす対応を求める
		range選択演算	終点が条件を満たす対応を求める
対応要素の構成要素	−	label演算	対応からラベルの集合を取得する
		domain演算	対応から始点の集合を取得す
		range演算	対応から終点の集合を取得する

3.2.2　演算結果が対応要素の構成要素の演算

対応要素の構成要素とは，ラベルの集合，始点の集合または終点の集合のことである．また，これらの構成要素に対しては，通常の集合と同様に和，差，共通集合演算などが利用できるものとする．

label演算　label演算とは，対応$f\{L\}$における対応要素のラベルの集合を取り出す演算である．この演算には任意の始点から出ているラベル矢のラベル集合を求めるdomain-label演算とそれとは逆に任意の終点へ向かうラベル矢のラベル集合を求めるrange-label演算の2種類があり，式8，式9のように定義する．なお，定義式においてNはノードの集合を表す．label演算は2ノード間の関係を求める問い合わせである．

・domain-label演算

$$lab[=Nf\{L\}]=\{l\,|\,(l,\ a,\ b)\in f\{L\}\wedge a\in N\}\quad(8)$$

・range-label 演算

$$lab[f\{L\}]=N]= \{l \mid (l,\ a,\ b)\in f\{L\}\wedge b\in N\} \quad (9)$$

domain 演算 domain 演算とは，対応$f\{L\}$における対応要素の始点の集合を取り出す演算であり，$dom[f\{L\}]$と記述し，式10のように定義する．domain 演算は，有向グラフの視点で考えると，ラベル矢が出ているノードを取り出す演算である．

$$dom[f\{L\}]=\{a \mid (l,\ a,\ b)\in f\{L\}\} \quad (10)$$

range 演算 range 演算とは，domain 演算とは逆に対応$f\{L\}$における対応要素の終点の集合を取り出す演算であり，$ran[f\{L\}]$と記述し，式11のように定義する．有向グラフの視点で考えると，ラベル矢が指しているノードを取り出す演算である．

$$ran[f\{L\}]=\{b \mid (l,\ a,\ b)\in f\{L\}\} \quad (11)$$

3.3 データ検索

ここでは対応演算による問い合わせ文の記述例を示す．対応演算の演算結果として得られるものは，対応，ラベルの集合，始点の集合および終点の集合のいずれかであり，これらの演算を任意に組み合わせて問い合わせ文を記述することになるが，記述例における注意事項を３つ挙げておく．

- 対応の和，差，共通演算，選択演算の結果，および$f\{L\}$は対応を表す．
- label 演算の結果はラベルの集合，domain および range 演算の結果はノードの集合を表す．
- domain 選択演算において$f[=C\{L\}]$のように“=”で比較を行う場合は$f\{L\}[C]$と略記する．つまり，この$f\{L\}[C]$はCの像を表すのではなく対応$f[=C\{L\}]$を表す．

図２に対応“K高校とT高校の生徒の部活動状況”を示す．ここでは，この対応“K高校とT高校の生徒の部活動状況”と図１に示した対応“あるK高校の生徒の家族”から“昇と同種の部活動を行っているK高校の生徒とその父と母を求める”という問い合わせを生成していく．対応“あるK高校の生徒の家族”を対応“家族”，対応“K高校とT高校の生徒の部活動状況”を対応“部活動”と略記する．

(1) 昇の部活を特定するために対応“部活動”{部活}[昇]の range をとる．

$$ran[\text{“部活動”}\{\text{部活}\}[\text{昇}]] \quad (12)$$

(2) K高校の生徒を見つけるために対応“部活動”{生徒}[K高]の range をとる．

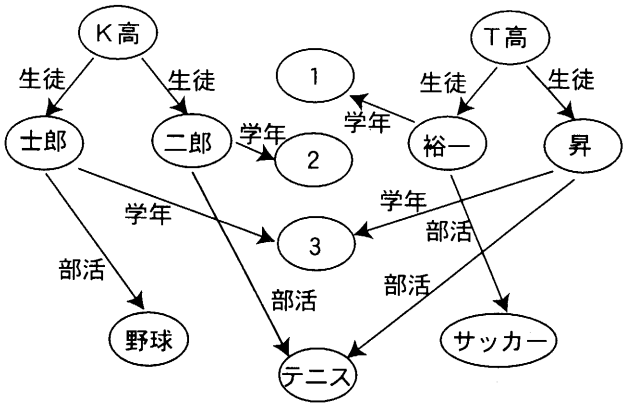

図２ 対応“K高校とT高校の生徒の部活動状況”

$$ran[\text{“部活動”}\{\text{生徒}\}[\text{K高}]] \quad (13)$$

(3) 昇と同種の部活をしているK高校の生徒を特定するために対応“部活動”{部活}[式13]（K高校の生徒とその部活を示す対応）から，その終点が式12（昇の部活）と等しいものを選択し，その domain をとる．

$$dom[\text{“部活動”}[\text{式13}]\{\text{部活}\}=\text{式12}]] \quad (14)$$

(4) 昇と同種の部活をしているK高校の生徒が式14と特定できたので，対応“家族”から“家族”{父，母}[式14]を取り出す．

$$\text{“家族”}\{\text{父，母}\}[\text{式14}] \quad (15)$$

したがって，問い合わせ“昇と同種の部活動を行っているK高校の生徒とその父と母を求める”を示す問い合わせ文は式16のようになる．

$$\begin{aligned}&\text{“家族”}\{\text{父，母}\}\\ &\quad dom[\text{“部活動”}[ran[\text{“部活動”}\{\text{生徒}\}[\text{K高}]]]\\ &\quad [\{\text{部活}\}=ran[\text{“部活動”}\{\text{部活}\}[\text{昇}]]]]] \end{aligned} \quad (16)$$

このように基本的な問い合わせは対応演算を組み合わせることで記述可能である．また，対応の和や差などを用いることで様々なビューでの対応を求めることもできる．また，例としては示さなかったが，選択演算と range 演算を組み合わせることで，パスをたどるような検索が実現でき，さらに domain 演算を組み合わせることで，パスを遡るような検索も実現でき，対応代数は十分な表現能力を持つといえる．

3.4 データ操作

対応追加 対応に対して新たな対応を追加することを対応追加という．有向グラフの視点で考えると，新たなラベルもしくは新たなノードとラベルを追加する操作である．この操作は，単純に追加する対応を加えることで実現で

きる．既存の対応を$f\{L\}$，追加する対応を$Add\{L_{add}\}$とすると，対応追加は対応演算を用いて次のように記述できる．

$$f\{L\} \leftarrow f\{L\} \cup Add\{L_{add}\} \tag{17}$$

対応削除　対応から任意の対応を削除することを対応削除と呼ぶ．有向グラフの視点で考えると，典型的にはラベル矢を削除する操作となる．対応データモデルでは対応そのものだけではなく対応の構成要素である始点の集合，終点の集合，ラベルの集合に対しても和，差，共通集合の集合演算を許すため，対応削除を行う方法は次の3種類で記述できる．

(1)　対応の削除（式18）

既存の対応$f\{L\}$から対応$Del\{L_{det}\}$を削除する．

(2)　始点または終点の削除（式19，20）

既存の対応$f\{L\}$（始点の集合A，終点の集合B）から始点a，終点bを削除する．

(3)　ラベルの削除（式21）

既存の対応$f\{L\}$からラベルlを削除する．

$$f\{L\} \leftarrow f\{L\} - Del\{L_{det}\} \tag{18}$$
$$A \leftarrow A - \{a\} \tag{19}$$
$$B \leftarrow B - \{b\} \tag{20}$$
$$L \leftarrow L - \{l\} \tag{21}$$

対応修正　対応の始点の集合，終点の集合，ラベルの集合の任意の要素の値を他の値に修正することを対応修正と呼ぶ．有向グラフの視点で考えると，ノードもしくはラベル矢のラベルを修正する操作である．対応演算では，対応要素のラベルや始点，終点を個々に修正できないため，修正対応を既存の対応に加え，さらに被修正対応を取り除く．この操作から分かるように対応修正は対応追加と対応削除の組み合わせで実現できる．

3.5　リレーショナルデータモデルとの比較

ラベル，始点，終点の三つ組の集合という対応データモデルのデータ表現形式がリレーショナルデータモデルに見かけ上類似していることから，リレーショナルデータモデルとの差異を明確にしながら，対応データモデルの特徴について述べる．一般的に，比較する項目としては，(1)データの表現能力，(2)データ操作言語，(3)データ操作，(4)データ更新時異状の4つが挙げられる．ここでは，(1)データの表現能力と(4)データ更新時異状の2つの項目について，実世界“K高校の生徒の特技と家族”をそれぞれ1リレーション，1対応でモデル化しそれらを比較する．それぞれのモデリング結果を図3および図4に示す．

データ表現能力　対応データモデルとリレーショナルデータモデルを実世界の表現能力という観点から比較する．図4から分かるようにリレーション“K高校の生徒の特技と家族”ではある生徒が特技を複数持っている場合や家族が多い場合，二郎の特技が剣道であるという情報が3つのタップルに現れたり，三郎の家族に四郎がいるという情報が2つのタップルに現れたりと，冗長な表現になってしまう．これはリレーショナルデータモデルにおいてはリレーションは第1正規形でなければならないためであり，これ以上タップルを減らすと実世界の情報が失われてしまう．このような問題を改善するには，逆に高次の正規化が必要になる．

生徒	特技	家族
二郎	剣道	太郎
二郎	剣道	花子
二郎	剣道	一郎
三郎	英会話	四郎
三郎	英会話	桃子
三郎	スキー	四郎
三郎	スキー	桃子

図3　リレーション“K高校の生徒の特技と家族”

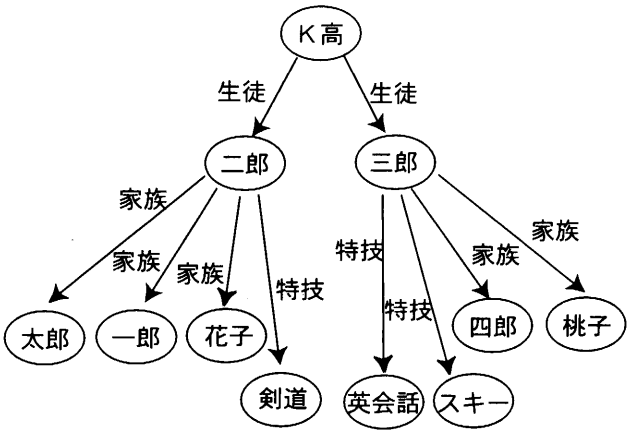

図4　対応“K高校の生徒の特技と家族”

一方，対応データモデルにおいては1対多の関係を簡便に記述できるため，特殊な操作を必要とせずに表現でき，冗長な表現も現れない．また，ノード毎に固有なラベルを持たせることが可能であるため，例えば生徒毎に任意の情報を持たせることができる．そのため，このようなデータに対しては対応データモデルの方がリレーショナルデータモデルより，実世界をわかりやすく表現でき，適合性が高いと考えられる．また，リレーショナルデータモデルで知られているデータ追加時異状，データ削除時異状，データ修正時異状のような異状も，同様の理由から対応データモデルでは発生しない．

このようにリレーショナルデータモデルでは高次の正規化が必要であるのに対して，対応データモデルではそのような操作の必要がない．また，リレーションが第一

正規形である必要があるのに対して，対応要素にはそのような制限もない．これは，対応データモデルでは，ノードがさまざまなデータ型のデータであることを想定しており，そのようなデータをリレーショナルデータモデルにおいて，ラベル，始点，終点の3つの属性名を持つ表では表現できないことを意味する．

4．問い合わせ言語 CRQL の設計

4.1 要求仕様

対応データモデルにおいては，操作対象が有向グラフであることから，最終的には GUI によるデータの操作の実現を目標としている．しかし，すべての機能を GUI でうまく実現できるわけではなく，問い合わせ言語においては特にそのような機能を簡便に記述できることが要求される．また，想定する構造が有向グラフということやスキーマフリーであることに起因する機能的なサポートも重要である．これらのことを考慮すると，対応データモデルにおける問い合わせ言語の主な要件としては次の6つが挙げられる．

(1) 対応完備である．
(2) 一般的な問い合わせを簡便に記述できる．
(3) グラフの向きを含めたパス表現による宣言的な問い合わせができる．
(4) 対応の入れ子構造に対する検索が簡便に記述できる．
(5) スキーマの更新履歴を管理する．
(6) 強力なデータの追加構文を持つ．

4.2 CRQL

CRQL で定義したコマンドと CRQL の構文の抜粋を拡張 BNF 記法を用いてそれぞれ表2，図5示す．CRQL においては，対象とする対応データベースの選択などを行う Main 状態，選択した対応データベースに対する操作を行う Command 状態，選択した対応に対して操作を行う Operation 状態の3つの状態に分けてそれぞれコマンドを定義した．コマンドは“；”までを区切りとして実行される．Command 状態および Operation 状態において，<Query> および <Expression> の指定ができる．<Query> においては，対応名に続けて“{}”で条件を指定することになるが，その後ろに“+”オプションを付けることで，ノードが対応であった場合，その対応も検索対象になる．また，構文図（図5）から <Query>, <Expression> ともにその記述は非常にシンプルであることが分かる．これは，対応データモデルにおいてはデータの形式が三つ組に限定されるため，結果の取得形式を含めて条件の指定が簡潔に指定できることによる．なお，Operation 状態における <Query> の対応名は省略することができる．また，ノードやラベルは用意した関数を用いて取得できる．

表2 コマンド一覧と説明

(a) Main 状態：

演算名	説明
create	対応データベースの作成
del	対応データベースの削除
connect	対応データベースへ接続，Command 状態へ移行
ls	存在する対応データベースの一覧表示

(b) Command 状態：

演算名	説明
set	変数に問い合わせ結果や演算結果の格納
in	対応の選択，Operation 状態へ移行
add	対応の追加
del	対応の削除
delhist	対応における履歴情報の削除
ls	存在する対応の一覧表示，対応名指定で対応要素の一覧表示

(c) Operation 状態：

演算名	説明
add	対応（要素）の追加
del	対応（要素）の削除
ls	存在する対応要素の一覧表示
readfile	対応のファイルからの読み込み
writefile	対応のファイルへの書き出し
nodelist	ノードの一覧表示
labellist	ラベルの一覧表示

(d) 共通：

演算名	説明
settime	時刻の設定（時刻指定なしで現在時刻）
time	時刻の表示
help	コマンドの一覧表示
quit	Operation 状態からの Command 状態への移行，Command 状態からの Main 状態への移行，main 状態におけるシステムの終了
end	システムの終了

CRQL の簡単な使用例を次に示す．connect コマンドを使用して，所望のデータベースに接続し，検索対象になる対応と条件を指定する．3.3節で例に挙げた問い合わせを実現する手順の一例を示す．対応データベース“友人”，“趣味”，“メモ”があり，“友人”に所望のデータが存在しているとすると，式16の問い合わせの手順は次のようになる．この問い合わせ自体が手続き的であるため，その記述が特に簡潔になるわけではない．また，問い合わせ結果は GUI を用いて表現することになる．

(CRQL の使用例：式16の問い合わせを行うまでの流れ)

```
> ls ;
友人　　趣味　　メモ
> connect 友人 ;
友人 >> ls ;
家族　　部活動
```

```
<Query> ::= (<Correspondence>)?{<State>}("+")?
<State> ::= <StateElement> ( ,<StateElement> )* | ε
<StateElement> ::= start:<Node> | label:<Label>
   | end:<Node> | path:<Path >
<Correspondence> ::=
   <OneCorrespondence> ( ,<OneCorrespondence> )*
<OneCorrespondence> ::=
   <CorrespondenceName> | <Variable> | <Query>
<Node> ::= ( not )? ( ( <Node> ( ( and | or ) <Node> )*
   | "("<Node>")") | <SimpleNode> (,<SimpleNode>) *)
<SimpleNode> ::=
   ( <NodeName> | '<RegularExpression>'
   | <NodeFunction> | ε ) ( condition )?
<Label> ::= (not)? ( <Label> ( ( and | or ) <Label> )*
   | "("<Label> ")"| <SimpleLabel> ( ,<SimpleLabel>)* ")"
<SimpleLabel> ::= ( <LabelName>
   | '<RegularExpression>' ) ( ( to | from ) <Node> )?
<Path> ::= (<NodeName> | '<RegularExpression>' )?
   (/ | //) (<LabelName> | '<RegularExpression>' )
   ( ( .<NodeName> | '<RegularExpression>' )?
   (/ | //) ( <LabelName> | '<RegularExpression>' ) )*
<Expression> ::= <Term> ( or <Term>)*
<Term> ::= <Factor> ( ( - | and ) <Factor> )*
<Factor> ::= <CorrespondenceName> | <Query>
   | <Variable> | "(" <Expression> ")"
```

図5　CRQL の構文（抜粋）

```
友人 >> 家族 {start :
    Domain (
      部活動 {start : K高校 : label : 生徒}
        {label : 部活, end :
          Range (部活動 {start : 昇, label : 部活}) }),
            label : 父, 母};
```

CRQL は，前述の要件を満たす問い合わせ言語であるが，特に要件 1，要件 3，要件 5，要件 6 について詳しく述べ，CRQL の有用性を示す．なお，要件 2 に関しては，要件 1 が満たされれば，データモデルにおける対応代数の記述能力の議論と CRQL の構文から明らかであり，要件 4 に関しては，<Query> のオプション指定で容易に実現できる．

対応完備性　対応演算として 9 種類の演算（表 1）を定義したが，独立な演算としては 7 種類（8 つの）演算が存在する．CRQL ではこれら 8 つの演算を次のように記述できるため，対応完備であるといえる．なお，*Label*(), *Domain*() および *Range*() は，それぞれ指定した対応のラベル，始点および終点の集合を求める関数である．

- 対応和演算（$f\{L1\} \cup g\{L1\}$）
  $f\{L1\}$ or $g\{L2\}$
- 対応差演算（$f\{L1\} - g\{L2\}$）
  $f\{L1\} - g\{L2\}$
- **domain** 選択演算（$f[\theta C\{L\}]$）
  $f\{start : \theta C\}$
- **range** 選択演算（$f\{L\}\theta C]$）
  $f\{end : \theta C\}$
- **domain-label** 演算　$label[= af]$
  $Label(f\{start : a\})$
- **range-label** 演算　$label[f = a]$
  $Label(f\{end : a\})$
- **domain** 演算（$dom[f]$）
  $Domain(f\{\})$
- **range** 演算（$ran[f]$）
  $Range(f\{\})$

パス表現　パスの表現方法は基本的には“NodeName/LabelName.NodeName/.../”の形式になる．ただし，ラベルだけを検索条件にする記述を簡便に行えるように，最初の“NodeName”と“.NodeName”に関しては省略可能とした．また，パス表現の場合に限ったことではないが，“’’”で囲んだ正規表現が利用できる．また，逆方向のパスは“//”で表現する．このようにすることで，ルートを意識することなく，パスの方向を含めた柔軟な検索が可能になる．

スキーマ更新の履歴管理　各々の対応要素に属性として登録時刻と削除時刻を持たせ，対応要素を削除しても削除時刻が設定されるだけで対応要素自体は削除しないようにすることで，スキーマの更新履歴管理を行う．検索時には，その時刻において有効な対応（要素）が検索の対象となる．過去に遡って検索を行いたい場合は，予め settime コマンドで時刻を設定しておく．履歴を削除したい場合には，作業時刻において有効でない履歴を一括削除する delhist コマンドを使用する．データの追加，削除を行う際に事前に settime コマンドを実行しておくことで，登録時刻や削除時刻を所望の時刻に設定することができる．

データの追加　データ追加コマンドは次の形式をとる．

```
add("+")?<NodeName>("{"<FilePath>"}")?
  |<CorrespondenceName>|<NodeFunction>
    (: <LabelName>
     : <NodeName>("{"<FilePath>"}")?
  |<CorrespondenceName>|<NodeFunction>)?
```

add コマンドの，“始点ノード名：ラベル名：終点ノード名”からなる対応要素を追加する．ノード名の後ろに“{}”でファイルパスを記述することでファイルを取り込むことができ，ファイルのデータ形式はユーザが指定する必要はない．また，add に“+”オプションを付け

ることで，終点ノードが集合で与えられた場合，それを集合の要素ごとに対応要素の追加を行う(オプションなしの場合，終点ノードが集合である対応要素が追加される)．また，“始点ノード名：ラベル名：中間ノード名：ラベル名：…：終点ノード名”とするデータの追加を行うことができる．例えば，次のような記述ができ，このようなデータに対しては非常に効率的なデータの追加が可能である．

(記述例)

add＋　太郎：趣味：{野球，テニス，囲碁}
add　　太郎：趣味：柔道：段位：三段

このようなラベル矢の方向まで考慮したパス表現による柔軟な問い合わせ，シンプルなスキーマ履歴の管理および効果的なデータの追加を実現する構文がCRQLの特徴であり，特にパス表現とデータの追加に関しては，利用者の利便性を高め非常に有用であるといえる．また，このような機能はOEMにおける問い合わせ言語であるLorelにおいても実装されていない．

5．おわりに

本論文では，グラフ構造を持つデータを概念理解そのままに記号化することができ，またスキーマの動的な変化に柔軟に対処することができる半構造データベースのための対応データモデルと問い合わせ言語CRQLを提案した．また，体系的なデータ操作を実現する対応代数を定義し，その対応代数を用いて様々な問い合わせの記述が可能であることを示した．対応データモデルにおいてはリレーショナルデータモデルにおける正規化のような問題が発生せず，またスキーマフリーであることから，半構造データのモデル化に適している．また，設計した問い合わせ言語CRQLについて述べ，既存の半構造データのための問い合わせ言語と比較して，効果的な記述ができるようになっていることを示し，その有用性を確認した．

今後は，GUIの開発を含めてユーザ親和性の高い対応データベースシステムの開発を検討する．GUIベースの問い合わせ言語としては，[8)~10)] などが挙げられるが，データモデルの特徴を活かした実装を目指す．また，筆者らはこれまで，アルゴリズムの自然なプログラム化を目的とした集合指向言語SOLの設計ならびに開発[11)~13)]を行ってきており，SOL上への組み込み実装も検討している．

参考文献

1）田島　敬史：半構造データのためのデータモデルと操作言語，処論データベース，Vol. 40，No. SIG3（TOD1），pp. 152-170，1999.

2）宝珍　輝尚：グラフに基づくデータベースに対する集合指向の統合演算，情処論，Vol. 35，Num. 3，pp. 444-452，1994.

3）S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener: The Lorel Query Language for Semistructured Data, International Journal on Digital Libraries, 1 (1), pp.68-88, April 1997.

4）Y. Papakonstantinou, H. Garcia-Molina, and J. Widom: Object exchange across heterogeneous information sources, Proc. of the IEEE Int. Conf. on Very Large Data Bases (VLDB), pp.132-141, 1994.

5）P. Buneman, S.B. Davidson, G. Hillebrand, and D. Suciu: A Query Language and Optimization Techniques for Unstructured Data, Proc. of ACM SIGMOD Conf. on Management of Data, pp.505-516, June 1996.

6）P. Buneman, S.B. Davidson, M. Fernandes, and D. Suciu: Adding Structure to Unstructured Data, Proc. of Int. Conf. on Database Theory (ICDT), 1997.

7）横尾　徳保，重松　保弘：半構造データのための対応データモデルの提案と問い合わせ言語の設計，DEWS2003論文集，ISSN 1347-4413，8-B-04，2003.

8）S. Flesca, F. Furfaro, and S. Greco: XGL: a Graphical Query Language for XML, Proc. of Int. Database Engineering and Applications Symposium, 2002.

9）M.P. Consens, and A.O. Mendelzon: GraphLog: a Visual Formalism for Real Life Recursion, Proc. 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Databse Systems, pp.404-416, 1990.

10）T. Houchin: DUO: Graph-based Database Graphical Query Expression, Proc. 2nd FarEast Workshop on Future Database Systems, pp.286-295, 1992.

11）重松　保弘，吉見　康一，吉田　将：集合指向言語SOLとその言語処理系の開発，情処論，Vol. 30，Num. 3，pp. 357-365，1989.

12）重松　保弘，與那覇　誠，吉田　将：集合指向言語SOLのデータベースへの応用，情処論，Vol. 33，Num. 8，pp. 1041-1051，1992.

13）横尾　徳保，重松　保弘：集合指向言語SOLのマルチメディアデータ型と手続きに関する拡張とその評価，情処論，Vol. 43，No. 6，pp. 1930-1939，2002.