半構造データのためのデータモデルと操作言語

田島敬史†

本論文では，ここ数年行われている半構造データのためのデータモデルと操作言語に関する研究について概観し，主な研究についての比較と考察を行う．また，これらの研究は従来のオブジェクト指向データベースやハイパーテキストに関する研究とも関係している．そこで，これらの研究との比較も行う．これらの比較から，本論文では，半構造データのデータモデルおよび操作言語を設計する上で特に重要な点は，以下の二点であると考える．まず一点目は，データモデルの設計の段階で，いわゆる従来の意味での「データ」と，従来のデータモデルでのスキーマ情報にあたるデータとを，区別無く扱えるようにするのが望ましいという点である．二点目は，操作言語は，データベース中のデータ構造の一部分を抜き出す狭義の「問い合わせ」操作だけでなく，データベース中のデータを再構成するような操作が表現できるべきであり，そのためにはなんらかのポインタの操作のための機構が必要になるという点である．また，今後の半構造データに関する研究の展望についても簡単に述べる．

Data Models and Query Languages for Semistructured Data

KEISHI TAJIMA†

In this paper, we survey, compare, and discuss the recent proposals on data models and query languages for semistructured data. These researches are also related to researches on object-oriented databases and hypertexts in the past. The comparison with those researches are also made. From those discussions, we consider that the following two points are key in the design of data models and query languages for semistructured data. First, a data model for semistructured data should model both "data" in the traditional sense and data corresponding to schema information in the traditional data model in a uniform way. Secondly, a query language for semistructured data should be able to express not only "selecting queries", which extract substructure from a database, but also "restructuring queries", which transform the structure of a database into another structure. To express restructuring query, functionalities for pointer manipulation are needed. In the last part of this paper, we also discuss a prospect on the future researches on semistructured data.

1. 半構造データ周辺の最近の状況

近年，半構造データ（semistructured data）と呼ばれる種類のデータのためのデータモデルやデータ操作言語に関する研究がさかんに行われており，1997年5月に開かれたACM PODS/SIGMOD'97ではWorkshop on Semistructured Dataと題した併設ワークショップ[1]が開催された．また，1997年1月に開かれたInternational Conference on Database Theory '97（ICDT'97）では当時Stanford大学に滞在していたSerge Abiteboul（現在はINRIAにもどった）が招待論文として半構造データに関するサーベイ論文[2]を発表し，ついで1997年の5月に開かれたACM PODS'97ではPennsylvania大学のPeter Bunemanが半構造データに関するチュートリアル[3]を行った．また，1998年には国際雑誌Information SystemsのVol. 23, No. 8が半構造データの特集号となっている．また，1999年の1月に開催されるInternational Conference on Database Theory '99では，Workshop on Query Processing for Semistructured Data and Non-Standard Data Formatsと題したワークショップが開催される予定である．

現在のところ，半構造データに関する研究の二大勢力はJenifer Widom, Hector Garcai-Molina（および上述のSerge Abiteboul）らを中心とするStanford大学系のグループと，Peter BunemanやDan Suciu（現在はAT&T Bell研）らを中心とするPennsylvania大学系のグループである．そこで，これらの二つのグループの研究を中心に，その他の重要な論文も含めながら，筆者なりの見方で整理し解説を行う．

† 神戸大学情報知能工学科
Dept. of Comp. and Sys. Eng., Kobe University

2. 半構造データとは

半構造データという言葉は，一般的に次のような特徴を持ったデータを指すために用いられているようである[2),3)].

(1) 構造が不規則なデータの集まりで，はっきりしたスキーマが定義できないようなデータ群.

(2) データベース以外の形で蓄積されていて，従来のデータベースのようにあらかじめスキーマが提示されているわけではないようなデータ群.

(3) スキーマはあるが，非常にゆるやかなスキーマで，全てのデータが厳密にスキーマに適合してるとは限らないようなデータ群

つまり,「schema-less」というキーワードで表現されるようなデータ群を指す.

しかし，全くデータ構造が無いようなデータ，例えば画像データ等の全くの生のデータ（raw data）のようなものは半構造データとは呼ばない．半構造データは上のような特徴を持ちつつ，かつその内部にはデータ構造を持っているものを指す．つまり，データの構造はあるが，その情報がスキーマとしてデータとは独立してあらかじめ与えられているのではなくて，データそのものの中に一緒に埋め込まれているのである．その意味で,「self-describing」というキーワードも用いられる.

より具体的には，従来のデータベースでは，関係名（関係データベースの場合）またはクラス名（オブジェクト指向データベースの場合）や属性名に関する情報は，データインスタンスとは独立したスキーマの中に記述されていた．これらの情報が，一般的な半構造データのためのモデルでは，データ中のタグとして与えられる形になっている.（データモデルについては詳しくは後述する.）

半構造データの例としては，以下のようなものが挙げられる.

- （WWW 上の html を含む）一般のハイパーテキストデータ.（上述の特性 1.）
- SGML 等の構造化文書の一部.（上述の特性 1 または 3.）
- WWW 上にページのリストや html 文書の中の表の形で蓄積されているデータなどで，実は共通の構造を持つデータの集合なのだが，事前にスキーマ情報が与えられていないデータ.（上述の特性 2.）
- データウェアハウスやデータ交換で用いられるようなデータフォーマットで記述されるデータ.（上述の特性 3.）一般に data warehousing やデータ交換では，多様なデータを柔軟に扱うために，比較的少ないプリミティブでゆるやかに定義されたフォーマットが用いられる．H. Garcia-Molina らの研究は，data warehousing のための，データモデルの研究が出発点になっている[4),5)].
- LaTeX の bib ファイルデータや，その他の文献データベースなど，一応データ項目があらかじめ決められておりスキーマは定義可能だが，オプショナルな項目などの自由度が高く，また世の中に蓄積されているデータが必ずしもきちんとスキーマに適合していないようなデータ.（上述の特性 1 または 3.）
- ACeDB[6)] という遺伝生物学者達の間では著名な（らしい）遺伝子情報データベース.（上述の特性 3.）このデータベースはスキーマを持つが，このスキーマは緩やかな制約を与えるものであり，各データが必ずこのスキーマに厳密に合致するとは限らない．このような遺伝情報データベースが Buneman のグループの初期の研究のモチベーションになっている[7)].
- 世の中に存在する様々な応用ソフトウェア特有のデータ形式で記述されたデータのうちの一部．例えば，VRML データは一つの仮想空間を，シーングラフと呼ばれる，様々な型を持つノードからなるグラフ構造によって記述する.（上述の特性 1.）

これらのデータを一言で言うと，従来のデータベースという枠組のもとで厳格にモデル化されて格納されているデータが「固い」データであるのに対して,「軟らかい」データであると表現できるかもしれない．このような「軟らかい」データに対しても，従来「固い」データに対して行われてきたような，set-oriented でかつ宣言的な問い合わせや操作を記述し効率的に処理したいというのが，半構造データに関する研究の中心となる目的である.

このように，半構造データのための操作言語とはスキーマを持たない軟らかいデータを扱うためのものなのだが，そのような言語は従来の固いデータに対しても時として非常に有用である．それは，これらのスキーマを持つデータに対しても，スキーマを意識しないで問い合わせを書きたいような場合があるからである．これには以下のような場合が考えられる.

- エンドユーザのためのインタフェースとして，そのデータベースのスキーマを知らなくてもある程度の問い合わせが書けるようにしたい場合.

- スキーマが頻繁に変更されるため，あまりスキーマに依存しない，スキーマの変更に強い問い合わせを書きたい場合．
- 本質的にデータ構造には依存しない問い合わせを書きたい場合．例えば，「データベース中の全ての文字列を求める」という検索など．

また，現在提案されている半構造データのための言語では，通常のスキーマに基づく問い合わせを記述することも可能である．例えば，後述する半構造データのための言語である UnQL は，関係をエンコードしたデータを入力し関係をエンコードしたデータを出力する限りにおいては関係論理と等価であり，入れ子関係をエンコードしたデータを入力し入れ子関係をエンコードしたデータを出力する限りにおいては入れ子関係論理と等価である[8]．

このように，半構造データの操作言語に関する研究は，単に最近の新しい応用に現れるある特定のデータ群のための枠組の研究ではなく，従来のデータベース言語の枠組をより一般的な形に拡張するための研究でもあるといえる．従来のデータベースの教科書では第一章で「データベースはスキーマを持ち，スキーマがデータベース中のデータの構造を一意に決定する」と説明され，それ以降の章の様々な部分がこの概念に基づいて書かれている．半構造データに関する研究は，この第一章を「スキーマはあるかもしれないし，ないかもしれない」，あるいは「スキーマは一つのデータ構造を決めるか，あるいはもう少し緩く，データ構造に関するある制約を定義する」と書き直す作業であると考えられるわけである．(データ構造を一意に決めるのではなく，ある制約を課すようなスキーマに関する研究については後述する．) それに従って，第二章以降の章も書き直す必要があるわけだが，現在のところ，「第二章：データモデル」，「第三章：検索・操作言語」のあたりまで書き直しが行われてきている．そこでこの論文では，半構造データのためのデータモデルおよび検索・操作言語に関する研究についてを中心に述べる．

3. 半構造データのためのデータモデル

半構造データに関する研究では，前述のようなデータを統一的に表現できるようなデータモデルを一つ仮定し，そのデータモデルに基づいて様々な枠組を設計することになる．そこで，まずこれまでの研究で提案されているデータモデルについて紹介していく．

3.1 Object Excahnge Model (OEM)

Object Exchange Model (OEM) は，Stanford 大のグループが異種データ統合ステムである TSIMMIS[4] システムのために設計した言語である．TSIMMIS システムは，様々な資源からのデータを，ある統一的なデータモデルにマップすることによって統一的なインタフェースを提供するが，その統一的なデータモデルとして OEM が用いられる．OEM はできるだけ多様なデータを表現できるように意図して設計されており，以下のような特徴を持つ．☆

- self-describing である．すなわちスキーマ情報はデータの中に記述される．
- オブジェクトとオブジェクト識別子の概念を基本とする．
- オブジェクトは集合型，または基本データ型である．各オブジェクトがどの型を持つかは，そのオブジェクト中のタグで記述される．
- 各データは，集合型のオブジェクトをノード，基本データ型のオブジェクトをリーフ，オブジェクト識別子による参照をエッジとするグラフを構成する．
- 各オブジェクトは，そのオブジェクトの意味を表すラベルを持つ．

一つ目の特徴からわかるように，OEM は半構造データのモデルの一種になっている．スキーマ情報は，各オブジェクトの型を表すラベルと，各オブジェクトの意味を表すラベルによって記述されている．

より形式的に定義すると，OEM のデータは *object* の集合であり，各 *object* は以下の構造を持つ．

$$object ::= \langle oid, label, \mathsf{set}, \{oid, oid, \ldots, oid\}\rangle \mid \langle oid, label, type, value\rangle$$

$$type ::= \mathsf{string} \mid \mathsf{integer} \mid \ldots$$

一行目の形式が集合型のオブジェクト，二行目の形式が基本データ型のオブジェクトを表す．*label* は任意の文字列である．データベースインスタンスは，このような構造を持つオブジェクトの集合であり，全体としては以下のような一つのグラフと二つの関数からなる三つ組みを構成すると解釈される．

$DB = \langle (V_a \cup V_c, E), n, v\rangle$

V_a : 基本データ型のオブジェクトを表すノード．

V_c : 集合型のオブジェクトを表すノード．

E : オブジェクト間の参照を表すエッジ．$E \subseteq V_c \times (V_a \cup V_c)$

n : ノードのラベルを与える関数．

☆ ここで解説する OEM は初期のバージョンの OEM である．その後の変更されたモデルについては後述する．

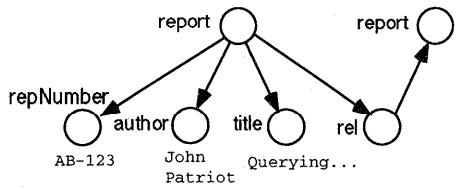


図1 OEMによるデータの例
Fig. 1 An example of OEM data

$(V_a \cup V_c) \rightarrow L$

v : 原子型オブジェクトの値を与える関数. $V_a \rightarrow D$

L : このデータベースで許されるラベルの集合.

D : このデータベースで許される原始型の値の集合.

実際のデータの例は以下のようになる[9].

⟨&r1, report, set, {&r1n, &r1a, &r1t, &r1r}⟩
⟨&r1n, repNumber, string, 'AB-123'⟩
⟨&r1a, author, string, 'John Patriot'⟩
⟨&r1t, title, string, 'Querying ...'⟩
⟨&r1r, rel, set, {&r2}⟩
⟨&r2, report, set, {...}⟩
(以下省略)

これは，文献データベースのデータの例である．一行目のオブジェクトは report というラベルを持ち，あるレポートを表している．このオブジェクトは集合型であり，4個の *oid*，すなわちオブジェクトへの参照を持っている．参照されいてる 4 つのオブジェクトは，それぞれ repNumber, author, title, rel というラベルを持ち，レポート番号，著者，タイトル，関連論文を表すオブジェクトである．これらのうち，レポート番号，著者，タイトルは基本データ型 string のオブジェクト，関連論文は集合型のオブジェクトで表現されている．これらのデータを図に表すと図1のようになる．図1からもわかるように，OEM のデータは基本的にはノードにラベルが振られたグラフである．

3.2 エッジラベル付きグラフ

Pennsylvania 大のグループも Stanford 大のグループとは独立に，エッジラベル付きグラフ（edge-labeled graph）と呼ばれる，よく似たデータモデルを提案している[10]．このデータモデルは以下のような特徴を持つ．

- やはり self-describing である．
- 基本的には nested record である．ノードを一意に識別する識別子のような概念は導入しない．ただし，tree-marker によりサイクルも表現できる．

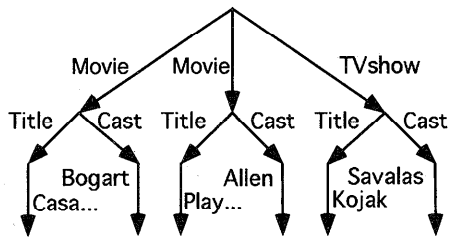


図2 エッジラベル付きグラフによるデータの例
Fig. 2 An example of edge-labeled tree

よってグラフを構成する．

- グラフ中の各エッジに，その意味を示すラベルを付けられる．また，整数や文字列などの基本型の値も，その値をラベルとして持ち，それ以上先が無いエッジとして表現される．
- 扱うデータは *tree* と *label* の二種である．

データは全て *tree* であり，*tree* は以下のように定義される．

$$tree ::= \{\} \mid \{label \Rightarrow tree\} \mid tree \cup tree \mid tree\text{-}marker$$

$$label ::= int \mid string \mid \ldots \mid symbol$$

実際のデータの例は，以下のようになる[10]．(以下の記述では，∪ 演算子で書くべき部分を $\{a, b, \ldots\}$ の形で略記している．)

{Movie ⇒ {Title ⇒ {"Casablanca" ⇒ {}},
Cast ⇒ {"Bogart" ⇒ {}}},
Movie ⇒ {Title ⇒ {"Play it again, Sam"⇒ {}}
Cast ⇒ {"Allen" ⇒ {}}},
TVshow ⇒ {Title ⇒ {"Kojak" ⇒ {}},
Cast ⇒ {"Savalas" ⇒ {}}}}

これを図に書くと，図2のようになる．上の例は，tree になっている例だが，tree-marker を使ってグラフを表現することもできる．例えば，以下の記述は，根から出て根自身に戻るエッジ一つだけからなるグラフを表す．

X where X = {A ⇒ X} end

where は，tree-marker と組み合わせてサイクルを含むグラフを定義するための構文である．

図2からわかるように，基本的にはこのモデルと OEM は良く似ているが，ラベルがノードに振られるか，エッジに振られるかが大きく異なっている．

3.3 オブジェクト指向データモデル

ここで，従来のオブジェクト指向データモデルはどのように表現されるのかを考えてみる．

従来のオブジェクト指向データモデルのオブジェクトは，関係データモデルのような統一されたモデルはないが，おおむね以下のような構造を持っていたと言

える．

$$
\begin{aligned}
object ::= &\langle oid, class, [att : value, \ldots, \\
&\qquad att : value]\rangle \mid \\
&\langle oid, class, \{value, \ldots, value\}\rangle \\
value ::= &\ c\_value \mid p\_value \mid oid \\
c\_value ::= &\ [att : value, \ldots, att : value] \mid \\
&\{value, \ldots, value\} \\
p\_value ::= &\ integer \mid string \mid \ldots
\end{aligned}
$$

上で定義したデータモデルでは，レコード構成子や集合構成子によって構成される任意の複合型の値，オブジェクト識別子，クラス名（あるいはクラスへのポインタ）の三つ組をオブジェクトとしている．データモデルによっては，その他にリスト構成子も含める物[11]や，原子型の値もオブジェクトになりうるとする物[12]もあるが，データベースにおいてもっとも重要なのはレコード型と集合型なので，ここでは省略した．また，上の定義中の *c_value* は，アイデンティティを持たない複合型の値を表している．O_2[12]などでは，このような複合型の値とアイデンティティを持つオブジェクトの双方をユーザが扱えるように提供しているが，GemStone[13]や ORION[14]などで用いられているデータモデルでは，ユーザが扱う原子型以外のデータは全てオブジェクトであり，これらと区別して複合型の「値」を（データモデルの定義中には現れるとしても）ユーザが直接操作するデータとして提供することはしない．

前述の図 1 に示されたデータの例を，従来のオブジェクト指向データモデルで表現する場合は，例えば次のような記述になるであろう．

```
⟨&r1, report, [repNumber: 'AB-123',
               author: 'John Patriot',
               title: 'Querying ...',
               rel: {&r2}⟩
⟨&r2, report, ...⟩
      （以下省略）
```

このデータ構造をグラフの形に表現する方法は色々と考えられるが，ここまでに述べた半構造データのためのデータモデルと対比できる形で，グラフで表現しようと思えば，各 *object*，*c_value*，*p_value* をノード，それらの間の参照（値として *oid* を持つ場合は，それは *object* への参照と考える）をエッジとし，また，各 *object* のクラス名または各 *p_value* の値自身をノードのラベル（*c_value* の場合は空のラベルと考える），各参照に対応する属性名（集合型の *object* からの参照の場合は空のラベルと考える）をエッジのラベルとするのが自然であろう．すなわち，ノードとエッジの双方にラベルを持つグラフを構成する．

3.4 Instance-Based オブジェクトモデル

ところで，半構造データということが言われるようになる以前から，オブジェクト指向データモデルの一種に，instance-based のオブジェクトモデルというものがあった[15)~17)]．これは，クラスの概念を排除し一つ一つばらばらの構造を持つオブジェクトを基本としようというもので，各オブジェクトは以下のような構造になる．

$$
\begin{aligned}
object ::= &\langle oid, [att : value, \ldots, att : value]\rangle \mid \\
&\langle oid, \{value, \ldots, value\}\rangle \\
value ::= &\ p\_value \mid oid \\
p\_value ::= &\ integer \mid string \mid \ldots
\end{aligned}
$$

これは，各エッジにラベルがあるグラフであり，各オブジェクトが同じ名前の属性を複数持てない点を除けば，ほぼエッジラベル付きグラフと同じものである．実際，図 2 に示したデータの例では，同じラベルのエッジを複数持つノードがないので，このデータを instance-based のオブジェクトモデルによるグラフで表現しても，全く同じグラフになる．

3.5 各モデルの比較

ここで，ここまでに挙げた四つのモデルの相違について考えてみる．

構造の異種性

まず，オブジェクト指向データモデルと残りの三つのモデルでは，以下の点が大きく異なる．

- オブジェクト指向データモデルでは，同じラベルを持つノード（すなわち同じクラスに属するオブジェクト）は同じラベルのエッジ（すなわち同じ属性）を持つが，他のモデルでは特にそのような制限は無い．

また最初の二つのモデルと後者二つのオブジェクト指向データモデルでは以下の点が大きく異なる．

- 後者二つのモデルでは，同じラベルを持つ複数のエッジが同じノードを起点とする（すなわち同じオブジェクトが同じ名前の属性を複数持つ）事を許さないが，最初の二つのモデルでは特に制限は無い．

オブジェクト指向データモデルでは一つの属性に複数のオブジェクトを参照させたければ，上で示したオブジェクト指向データモデルで表現したレポートデータベースの例の中の rel 属性のように，参照したい複数のオブジェクトを要素とする集合型の値（あるいはオブジェクト）を用意し，その値（あるいはオブジェクト）を一つの属性が参照することになる．よって，この二つ目の違いは，別の言い方をすれば，最初の二つ

のモデルでは集合型の属性と集合型でない属性を区別しないということであるとも言える．

これらの違いはいずれも，スキーマが曖昧で，検索を記述する時点では構造が未確定なデータ群を扱うための配慮から来ている．

ラベルの扱い

次に，これら四つのモデルの最も大きな違いは，どこにラベルをつけるかという点である．OEM ではノードにラベルがつけられ，エッジラベル付きグラフや instance-based オブジェクトモデルではエッジにラベルがつけられ，通常のオブジェクト指向データモデルではその両方にラベルがつけられる．基本的には，四つのうちのあるモデルで表現されたデータを他のモデルを使ってエンコードするのは難しくないので，どのモデルも表現力に大きな違いは無い．例えば，OEM で表現されている図 1 のデータは，エッジラベル付きグラフでは以下のように記述できるであろう．

```
{report ⇒ R1, report ⇒ R2}
where R1 = {repNumber ⇒ {"AB-123" ⇒ {}},
            author ⇒ {"John Patriot" ⇒ {}},
            title ⇒ {"Querying ..." ⇒ {}},
            rel ⇒ {Report ⇒ R2}},
      R2 = {...}
           (以下省略)
      end
```

また逆に，ラベルエッジつきグラフで表現されている図 2 のデータは，OEM では以下のように記述できるであろう．

```
⟨&e1, Entertainment, set, {&m1, &m2, &t1}⟩
    ⟨&m1, Movie, set, {&m1t, &m1c}⟩
        ⟨&m1t, Title, string, 'Casablanca'⟩
        ⟨&m1c, Cast, string, 'Bogart'⟩
    ⟨&m2, Movie, set, {&m2t, &m2c}⟩
        ⟨&m2t, Title, string, 'Play it again, Sam'⟩
        ⟨&m2c, Cast, string, 'Allen'⟩
    ⟨&t1, TVshow, set, {&t1t, &t1c}⟩
        ⟨&t1t, Title, string, 'Kojak'⟩
        ⟨&t1c, Cast, string, 'Savalas'⟩
```

しかし，それぞれの長所，短所もいくつか考えられる．まず，ノードにラベルをつける利点としては，以下の点が考えられる．

- 各オブジェクト（またはレコード）は，あるクラス（またはリレーション）に属するという，従来のデータモデルでの形式を踏襲しており，理解しやすい．

一方，以下のような点では，ノードよりエッジにラベルを付ける方が有利であると思われる．

- エッジラベル付きグラフでノードにつけられたラベルを表現するには，そのノードに「node label」という枝を一つ加えればすむのに対し，エッジにつけられたラベルを OEM で表現するのは，一つのノードが，異なるラベルを持つ複数のエッジから参照される場合を考えると，煩雑になる．
- 概念的にも，あるノードの持つ意味は，そのノードへの参照毎に定義できると考えた方が，より自然に柔軟な表現ができる．
- ノードにラベルをつけると，ノードとノードをマージするという操作が表現しにくくなる[3]．一方，エッジとエッジをマージするという操作はあまり必要性が無いので，エッジにラベルを付けても問題にならないと思われる．

また，Buneman らによるエッジラベル付きグラフモデルの他のモデルと比べて最も特徴的な点は，全てを *tree* 型と *label* だけで表現している点である．OEM や instance-based オブジェクトモデルでは，属性名にあたるノードやエッジのラベルと，基本型の値が区別されているが，エッジラベル付きグラフでは，基本型の値も，その先にそれ以上エッジがないエッジのラベルとして表現されているので区別がない．この属性名と属性値が同じ形で表現されているという点は重要である．これはつまりデータベース中の従来の意味での「データ」と，スキーマ情報とが区別無く取り扱われていると言うことである．これにより，後述する検索の際に，「スキーマ情報の検索」や「スキーマ情報とデータを混合した検索」が自然に書けることになる．

このように，現時点では半構造データを表現するモデルとしてはエッジにラベルを付けるエッジラベル付きグラフが有力なようである．実際，上述の OEM モデルは実は初期のバージョンの OEM であり，OEM はその後エッジにラベルをつけるモデルに変更された[2),18)]．変更後の OEM のデータ構造の定義は以下のように記述できるであろう．

$$object ::= \langle oid, \text{pf set}, \{\langle label, oid\rangle, \ldots, \langle label, oid\rangle\}\rangle \mid \langle oid, type, value\rangle$$

$$type ::= \text{string} \mid \text{integer} \mid \ldots$$

上記の構造を持つオブジェクトの集合からなるデータベースインスタンスは，以下のようなグラフを構成すると解釈される．

$DB = \langle (V_a \cup V_c, E), n, v\rangle$

V_a ： 基本データ型のオブジェクトを表す

ノード

V_c : 集合型のオブジェクトを表すノード.

E : オブジェクト間の参照を表すエッジ. $E \subseteq V_c \times (V_a \cup V_c)$

n : エッジのラベルを与える関数. $E \to L$

v : 原始型オブジェクトの値を与える関数. $V_a \to D$

L : このデータベースで許されるラベルの集合.

D : このデータベースで許される原始型の値の集合.

n が $(V_a \cup V_c) \to L$ から $E \to L$ になった点が，変更されている点である.

同一性の扱い

ラベルに関しては，新しい OEM でもエッジラベル付きグラフでもエッジに付ける形となったが，もう一つの OEM とエッジラベル付きグラフの大きな違いはオブジェクト識別子を導入するかどうかである．前述のように，エッジラベル付きグラフは基本的に値に基づく同一性を用いる nested record であって，ノードを一意に識別する識別子のようなものは導入しない．このモデルでは，木の間の等価性はグラフ間の bisimulation による．これは，ノードの識別子のようなものがない時に，ユーザが値に基づいて識別可能な同一性を定義したものであり，直感的には，根からたどれるパスの集合が等しいグラフは bisimilar であると考えればよい.

このため，エッジラベル付きグラフでは，例えば，{name ⇒ "Allen", name ⇒ "Allen"} という tree と {name ⇒ "Allen"} という tree は区別されない．この例は，tree は label と subtree の対の集合であるという定義からも，区別されないのが当然と見えるかもしれないが，例えば，X where X = {A ⇒ X} end と X where X = {A ⇒ {A ⇒ X}} end なども区別されない．前者の例は，オブジェクト指向データモデルの関係データモデルに対する優位性を論じる時に言われていた問題と同じものである．このため，エッジラベル付きデータモデルでは，後述の再構成検索などにおいてオブジェクト識別子の特殊な管理を必要としない代わりに,「値としては同一な二つの異なるデータ」というものを表現できないことになる.

なお，bisimulation についてもう少し詳しく述べておくと，bisimulation の実際の定義は，ϵ-エッジ，すなわちパスのナビゲーションにおいて自由に遷移できるエッジを含むグラフに対して定義されているので，ϵ-エッジで終わるパスに関する考慮が必要となる．しかし，ϵ-エッジを含む任意のグラフは，ϵ-エッジを含まない bisimilar なグラフを持っているので，ϵ-エッジを含まないグラフだけを考える場合には，上述のような直感的な解釈で十分であろう．ϵ-エッジを含むグラフを考えた場合，bisimulation による同値類は regular tree に対応することになる.

次に，オブジェクト指向データモデルとの比較を考える．前述のように，一部のオブジェクト指向データモデルでは，アイデンティティを持つオブジェクトとアイデンティティを持たない複合型の値の双方をユーザに提供するが，複合型の値が用いられるのは主にアイデンティティを考える必要のない属性値や一時的な計算結果としてであり，問い合わせ言語などは基本的にはアイデンティティを持つデータを扱うことを主眼としている．半構造データのためのデータモデルでも，アイデンティティを持つノードと持たないノードの双方を提供するということも考えられなくはないであろうが，一般に半構造データのためのデータモデルは多様な異種性のあるデータを一つのシンプルな枠組みで表現することを目標にしているということもあり，これらの双方を一つのモデルに組み込むということは，あまり考えられないようである.

3.6 関係データモデルとの関係

前述の OEM やエッジラベル付きグラフは，従来の関係データモデルの関係，あるいは入れ子関係をエンコードできる．例えば，**図 3** は簡単な関係をエッジラベル付きグラフモデルで表現した例である.

逆に，このようなグラフ構造を，関係を使って表現する事も可能である．例えば，次のようなスキーマの関係を用いればよいであろう.

relation node = [nid]

relation edge = [source, label, destination]

nid は各ノードに一意に割り当てられる識別子，source と destination は各々エッジの始点と終点のノードの識別子である．OEM ではノードに識別子が与えられているので，OEM で表されているデータを関係データモデルでエンコードする場合には，それらをそのまま nid として用いれば良いであろう．一方，エッジラベル付きグラフではノードの識別子という概念はないので，エッジラベル付きグラフモデルで表現されているデータを関係データモデルでエンコードする場合には，与えられたデータを表現する適当なグラフを一つ想定し，そのグラフ中のノードに識別子をあらたに割り振ってやってから，このグラフを関係の形にエンコード必要がある．このようにした場合，本来のデー

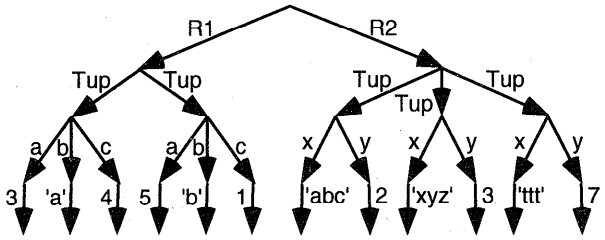


図 3 関係を表現したグラフ
Fig. 3 A graph representing a relation

タにはない「識別子」が関係を用いてエンコードされたデータの中には現れることになるので，エッジラベル付きグラフ上の操作の意味を関係を用いた表現上の操作にマップするのが難しくなるのではないかという心配も考えられるが，後述するエッジラベル付きグラフのための操作言語である UnQL で表現される操作では，値として等しい部分グラフに対しては常に集合的に同じ操作が行われるので，特に問題は生じない．

しかし，エンコードできるから，では関係データモデルでやればよいというわけにはいかない．その理由としては以下のような点が挙げられる．

- 確かにエッジラベル付きグラフに対する単純な問い合わせは関係論理 + 再帰問い合わせ（recursive query）で書けるが，後述する再構成検索など，それだけではうまく書けないような問い合わせがある．
- OID を導入したモデルを使う場合には，関係データモデルには無い OID の管理が必要である．

3.7 ハイパーテキストのためのデータモデルとの関係

前述のようなグラフ構造を持つデータであり，長く研究されて来ているものにハイパーテキストがある．ハイパーテキストの形式的なモデルには様々なものがあるが，ここでは以下の二つに分類して考える．

- ハイパーテキストを「ノード」と「ノード間の参照」でモデル化するもの．
- ノード間の「リンク」もオブジェクトであるとするもの．

このうち前者については，基本的には instance-based のオブジェクト指向データモデルと良く似たものになる [19)~21)]．一方，後者の場合は，おおむね以下のようなデータモデルになる [22)~24)]．

$$node ::= \langle nid, [att : value, \ldots, att : value] \rangle$$
$$links ::= \langle lid, nid, nid, label \rangle$$

nid は各ノードに一意に割り当てられる識別子，lid は各リンクに一意に割り当てられる識別子である．ノード中の $att : value$ は，各ノードの属性やその内容文書などのデータであり，これらはハイパーテキストのモデルでは通常ノードとしては表現されない．

このモデルでのノード間の関連の表現の仕方は，各識別子をキー属性と考えれば，関係データモデルでの関連の表現の仕方と同じものである．実際，文献 24) などでは，このデータモデル上に関係論理とほぼ等しい検索言語を定義している．

3.8 高機能データモデルのアプローチ

OEM やエッジラベル付きグラフのアプローチは，世の中で用いられている様々なアプリケーション毎のデータフォーマットをエンコードするためのシンプルな枠組を提供するという立場であったが，逆にこれらの多様なデータフォーマットを全て直接表現できるような，非常に高機能なデータモデルを提供するという考え方もある．しかし，どんなに高機能なデータモデルを提供しても，次々に新しく登場する世の中の全てのデータフォーマットを表現することは永久に不可能であろう．よって，このアプローチはあまり現実的ではない．

ただし，これは新しいデータフォーマットに柔軟に対応できるように，これらをエンコードできる枠組が必要であるというだけで，そのような単純なデータモデルに追加する形で，多くのデータフォーマットに共通して現れるような機構を取り込む事を否定するものではない [2)]．

3.9 Minimailist アプローチ

エッジラベル付きグラフは非常に単純なデータモデルであるが，さらにより単純なデータモデルは無いかと追求する立場もある．例えば，Buneman は 文献 3) の中で，最も単純なデータ構造として Lisp のデータ構造を考えてみる可能性について触れている．Lisp のデータ構造は低レベル過ぎるので，もちろんユーザが直接このデータ構造を扱う言語を書くべきものではないと思われるが，言語の基礎となる枠組を考察する

上では，興味深いアプローチであろう．

4. 簡単な問い合わせのための言語

では次に，これまでに述べたようなモデルの上にどのような言語が定義されるのかについて見て行く．

4.1 パス表現による問い合わせ言語

グラフ構造を持つデータに対する問い合わせ言語として最も基本的なものに，従来のオブジェクト指向データモデルやハイパーテキストモデルで用いられていたパス表現（path expression）[25] を使った言語がある．例えば次のようなスキーマを持つオブジェクト指向データベースを考える．

Company : [..., division : {Division}]
Division : [..., employee : {Employee}]
Employee : [name : string, ..., salary : integer]

すなわち，Company というクラスは divison という属性を通して各部署を表すオブジェクト（クラスは Division）の集合を参照しており，さらに各部署は employee という属性を通してその部署の全従業員（クラスは Employee）の集合を参照している．

このようなデータベースに対して，文献 26) で提案されているパス表現を用いた言語では，例えば以下のような検索が記述できる．

```
select X, Y.name
from Company X
where X.divison.employee[Y].salary < 100,000
```

この検索は，給料が 100,000 円以下の従業員の名前と，その会社の名前の組を返す．where 節で与えられたパス表現は，そのパターンにマッチする全てのオブジェクト列に展開され，またその時にパス中に集合型の属性が現れる場合については，これを各要素毎に展開してオブジェクト列を作る．パス表現中に現れる [Y] の Y には，各オブジェクト列中のその位置に現れる従業員オブジェクトが束縛される．また比較演算子 < はパス表現の末端の salary の値に適用される．

パス表現を用いた検索言語は，基本的にはこの例と良く似た枠組の上に様々な拡張を行っている．例えば，文献 27) ではパス表現をグラフに拡張した query graph を提案している．また文献 21) では単一のパス（彼らは walk と呼ぶ）だけでなく，互いに 関係し合うパスの集合（彼らはこれを hyperwalk と呼んでいる）を扱うような拡張を行っている．

4.2 属性抽象を含むパス表現

しかし，上記のようなパス表現による検索では，事前に各ノードの持つ属性名を把握していないと検索が書けない．また各ノードにどんな属性が定義されているのかという情報自身をデータとして検索するような検索も書けない．そこでパス表現を拡張し，属性名も値として扱い，属性名を値として取る変数を導入する事が行われる．例えば，前述の文献 26) では，属性名を値として取る変数を使って以下のような検索が記述できる．

```
select X.name
from Person X where X."Y.City = 'New York'
```

ここで，"Y につけられた " は，その変数が属性名を値として取る「属性変数」であることを表す．この検索は，Person クラスのオブジェクトのうち，なんらかの属性を通して City = 'New York' という属性を持つオブジェクトを参照しているようなものについて，その Person の名前を返す．また単に

```
select "Y
from Person X
where X."Y.City
```

と書けば，Person クラスのオブジェクトのうち，なんらかの属性を通して City という名前の属性を持つオブジェクトを参照しているものについての，その間に入っている属性名が返される．つまり，構造情報に関する検索が記述できる．

実は，データモデルのレベルでリンクそのものもデータであるとしているような場合は，このような記述が自然に可能となる．例えば，前述のような，リンクをデータとして扱うハイパーテキストのためのデータモデル上で上のような検索を記述すると，おおむね以下のような感じになる．

```
select L1.linktype
from Node N1, Link L1, Node N2, Link L2
where N1.nodetype = 'Person' ∧
      L1.source = N1 ∧ L1.dest = N2 ∧
      L2.source = N2 ∧ L2.linktype = 'City'
```

この検索は，単なる関係データベースでの典型的な join 検索である．前述のように，文献 24) で提案されている言語は，文献 24) の中でも述べられているように，実は通常の関係論理とほぼ等しい．しかし，上のような検索ができるというのは，データモデルのレベルでリンク自体もデータであるとしていたことによるのである．

4.3 GraphLog

上のように，属性名をデータとして扱う機構を言語に入れるか，あるいはデータモデルとしてはじめからリンクをデータとして扱えば，通常のパス表現や関係論理で，ある程度スキーマが未確定なデータに対する検索が記述できる．しかし，あるノードより下にある

任意のノードに関する検索など，任意の長さのパスを扱う必要がある検索では，再帰問い合わせ（recursive query）を導入する必要がある．

そのような再帰問い合わせを導入した言語として，GraphLog[22),23)] がある．GraphLog では，グラフ構造を前述のリンクをデータとするハイパーテキストのモデルとほぼ同じような形で表現し，このデータ上で Datalog（文献 28) 参照）とほぼ等しい言語を定義する事で，グラフをトラバースするような再帰問い合わせを表現可能としている．

4.4 Lorel による問い合わせ

Lorel[18)] は OEM 上の検索言語として開発された，オブジェクト識別子と論理に基づいた言語で，OQL[29)] などの SQL-like な言語に，正規表現を含むパス表現を記述できる機能と，構造の異なるデータを要素とする集合を扱うための様々な自動的な型変換の機能を加えたものに相当する．

Lorel での検索は例えば以下のようになる [2)]．

```
select R.name, R(.address)?.zipcode
from GuideMap.♯.restaurant R
where R.% grep 'cheap'
```

()?は，カッコの中の部分があってもなくても良いオプショナルでことを表し，♯は任意の長さのパスにマッチする wild card，% は任意のラベルにマッチする wild card である．この検索は，変数 GuideMap で参照されているオブジェクトからグラフを下にたどった任意の場所にある restaurant エッジで参照されているオブジェクトについて，その直下のいずれかの属性に 'cheap' という単語が現れるか調べ，現れるものについては，その直下の name 属性の値と，直下または address というエッジを一段挟んだ所にある zipcode 属性の値を返す．

4.5 UnQL による簡単な問い合わせ

UnQL[8)] は，エッジラベル付きグラフモデルの操作言語として開発された言語である．UnQL については，後ほどさらに詳しく解説するが，ここで簡単に UnQL ではこれまでに見たような検索はどのように記述されるのかを見ておく．

前述のようにエッジラベル付きグラフデータモデルは，*tree* と，そのラベルのために用いる *label* からなる．*label* は整数や文字列などの型のいずれかである．UnQL での操作の単位は *tree* 型であり，*tree* を入力として，*label* の集合，または *tree* を返す．*tree* も，「top-level の枝の集合」として定義されていた事に注意されたい．

UnQL による最も単純な問い合わせの例は，例えば

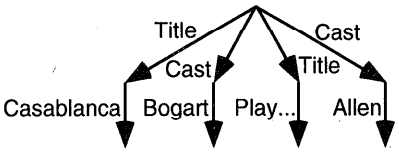


図 4 問い合わせの結果
Fig. 4 The query result

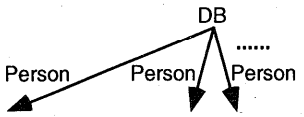


図 5 Person データを含む DB
Fig. 5 DB including Person data

以下のようになる．

```
select t
where Movie ⇒ \t ← DB
```

文法が，従来の SQL-like な文法とは少し違うので簡単に解説しておく．$e \leftarrow T$ の部分は generator と言って，T に束縛されている *tree* の top-level の各枝を生成し，e と比較する．そして，e のパターンとマッチするものについて，e 中で導入された変数についての束縛を行った後，select 節が評価される．最後に，実行された全ての select 節の評価の結果の集合和が返される．e 中で \ をつけられたシンボルは，そこで新たに導入された変数である．(\ は直感的には λ-calculus における λ のイメージである．)

上の例では，DB の top-level の各枝のうち，Movie というラベルを持つ枝について，select 節が，その枝の一つ下の枝の集合を返し，最後にこれらの集合の和が返される．枝の集合はすなわち *tree* なので *tree* が返されることになる．この問い合わせは，DB という変数で図 2 のグラフが参照されていたとすると，**図 4** で表されるグラフを結果として返す．

次の例は，*label* の集合を返す場合の例である．

```
select s
where Movie ⇒ Title ⇒ \s ⇒ {} ← DB
```

この検索は，DB 中の全ての映画のタイトルの集合を返す．

また，これまでの節で構造情報に関する検索の例として挙げた，「Person クラスのオブジェクトのうち，なんらかの属性を通して City という属性を持つオブジェクトを参照しているものについて，その間に入っている属性の名前を返す」という検索は以下のようになる．ただし，ここでは，**図 5** のように，全ての Person オブジェクトへの参照が，DB に束縛されている *tree* の top-level において，Person というラベルを持った

エッジとして与えられていると仮定した．

select l

where Person $\Rightarrow \backslash l \Rightarrow$ City $\Rightarrow \backslash t \leftarrow$ DB

このように，Peter Buneman らによるエッジラベル付きグラフのモデルでは，属性名を表す情報も，前述の「映画のタイトル」などの情報も，ともに「ラベル」という共通の形で表現されているため，構造に関する検索と通常の検索が全く区別無く行える．

4.6 各言語の比較

この章で述べたような簡単な検索に関する限りは，各言語に本質的な違いはなく，データモデルのレベルで属性名自身もデータとしてモデル化されているか，あるいは問い合わせ言語の側で属性名を抽象する機構を用意するかさえしてやれば，ここであげたような例と同様の検索を書くことができる．半構造データに対するこれらの問い合わせを考える際には，言語設計よりもむしろ効率的な評価方法が問題となる．一方，言語の設計が問題となるのは，再構成検索と呼ばれる，より複雑な検索を考える場合である．そこで，次の章ではこの再構成検索について解説する．

5. 再構成検索（restructuring queries）

前節で挙げた問い合わせは，構造情報にしても通常の情報にしても，データベース中に記述されている情報のうちの一部を取り出して来るという問い合わせであった．これを狭義の問い合わせ，または部分抽出検索（selecting queries）と呼ぶことにする．

しかし，もう一つの重要なタイプの問い合わせとして，データベース中のデータを再構成する問い合わせというものがある．この「再構成する問い合わせ」という考え方は非常に重要である．例えば，Serge Abiteboul の 招待論文，文献 2) の章構成は，

Section 1. Introduction,
Section 2. Semi-Structured Data,
Section 3. Modeling Semi-Structured Data,
Section 4. Querying and Restructuring

となっている．また，Peter Buneman の文献 8) の章構成も，

Section 1. Introduction,
Section 2. The Data Model,
Section 3. UnQL: Selection Queries,
Section 4. UnQL: Restructuring Queries

と続く．

再構成が重要である理由として Serge Abiteboul は，ユーザのビューを構築するために重要であると述べている [2]．つまり，半構造データでは，構造が曖昧で，関係データモデルのような正規形も存在しないため，ユーザごとに目的にあった構造に再構成したい需要が高いということである．しかし，関係データモデルの時から，問い合わせとはそもそも再構成であった．問い合わせとは，データベース中の情報を使って，(1) ユーザが今見たい情報を (2) ユーザが見たい構造にして，返してくれるものだということである．関係データベースでは，問い合わせの結果は一つの関係であり，ユーザが見たい情報を一つの関係で表現するための形は，ほとんどの場合一意に決まっていたと考えられるので，後者の「見たい構造にして」の部分はそれほど重要ではなかった面がある．また，オブジェクト指向データモデルでは，後述のようなオブジェクト識別子に関わる問題から，問い合わせ＝再構成という考え方が犠牲になっていた面があった．しかし，半構造データの場合，ユーザが見たい情報をどのような形に表現するかに多くのバリエーションが考えられるので，後者の「見たい構造にして」という点が，より重要になると考えられる．以下，まずこの点についてもう少し解説し，その上で半構造データの再構成のための研究について見て行くことにする．

5.1 関係データモデルにおける再構成検索

関係データモデルにおける検索は，任意の関係（複数可）を入力とし，これらを再構成して，任意の構造の関係にして出力するものであった．(少なくとも，それを目指していた．) 一般的に，関係データモデルの問い合わせと聞いてまず思い浮かべるのは，selection-projection-join 検索と呼ばれるものである．これを見ると確かに，selection と projection という関係の一部を切り出す操作に，関係データモデルの鍵となる発明である join という操作を加えたものという印象も受ける．しかし，よく知られているように，関係代数は次のような形式で記述される関係論理と等価であった．

$$R = \{< v_1, v_2 > \mid \exists x.R_1(v_1, x) \wedge R_2(x, v_2) \wedge P(v_2)\}$$

これを見ると，関係データモデルでの問い合わせは，「データベース中の情報をもとに，任意の関係を生成しユーザに返すもの」であった（少なくともそれを目指していた）と言える．だからこそ，関係データモデルでは $view \equiv query$ であったのである．

5.2 オブジェクト指向データモデルにおける再構成検索

関係データモデル同様にオブジェクト指向データモデルにおいても，問い合わせはユーザが必要とする任意のデータ構造に再構成するものであってほしい．

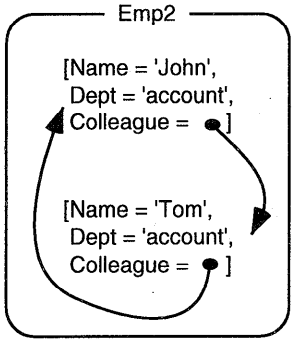


図 6 意図していた結果
Fig. 6 The intended result

しかし，前述のようなパス表現による問い合わせは，「データベース中のデータから必要な部分を抜き出す操作」すなわち，部分抽出検索を記述するためのもので，任意の構造への再構成をするには不十分であった．例えば，以下のような再構成検索を考えてみる．

```
Emp2 =
    select Name = x.Name,
           Dept = x.Dept,
           Colleague = (select y
                        from Emp y
                        where x.Dept = y.Dept)
    from Emp x
```

この問い合わせは，名前を表す属性 Name と所属部署を表す属性 Dept の二つの属性を持つ，従業員を表すクラス Emp のオブジェクトに対し，同じ部署に所属する従業員の集合を参照する属性 Colleague を加えるように再構成する問い合わせである．

しかし，オブジェクト指向データベースの再構成検索としては，上の問い合わせは不十分である．なぜなら，この再構成の結果として本来意図しているものは，**図 6** に示すような構造であるのだが，実際には上の操作の結果は**図 7** に示されるような構造になるからである．すなわち，新しく生成された Emp2 のインスタンスが Colleague 属性を通して参照するのは，やはり Emp2 のインスタンスであって欲しいのだが，上の問い合わせの結果中では，Colleague 属性で参照されるオブジェクトは Emp のインスタンスのままとなってしまう．このような問い合わせは，関係データモデルでは簡単な射影操作として記述される非常に基本的な問い合わせであるのに対し，それに相当する操作がオブジェクト指向データベースではうまく表現できない事になる．

そこで，図 6 に示されるような構造への変換を記述できるような言語が，提案されている [30]．この言語では，as という構文を導入する事で，上の問い合わせを以下のように記述できる．

```
Emp2 =
    select Name = x.Name,
           Dept = x.Dept,
           Colleague = (select (y as Emp2)
                        from Emp y
                        where x.Dept = y.Dept)
    from Emp x
```

(o as C) はオブジェクト o を，他のクラス C 中の，o に対応したインスタンスに変換する操作である．ここで，上の例では，as の引数のクラスとして，その as 構文を含む問い合わせ自身によって定義されているクラス Emp2 が再帰的に現れていることに注意されたい．よって，この as は，あくまで y を Emp2 中のオブジェクトへの「参照」に変換するだけで，(y as Emp2).Name のように変換されたオブジェクトの中身にアクセスする事はできない．

このような，オブジェクト識別子の操作に関する研究は，演繹オブジェクト指向データベース（deductive object-oriented database）の枠組の中では様々な研究が行われていた．例えば ILOG[31] では，上のような問い合わせは以下のように記述できる．(ただし，文献 [31] で用いられている ILOG では集合値を一般の値としては扱えないので，下の例では同じ部署に所属する従業員全ての集合を属性値とする代わりに，同じ部署に所属する従業員の組み合せごとに Emp2 を生成している．)

```
Emp2ID(*, id) ← Emp(id, name, dept)
Emp2(id2, name, dept, cid2)
                ← Emp(id, name, dept),
                  Emp(cid, name2, dept)
                  Emp2ID(id2, id)
                  Emp2ID(cid2, cid)
```

ここで，* は新しいオブジェクト識別子を生成するための構文である．この例では，* は id を引数とする Skolem 関手（Skolem functor）によって，各 id に対して一意な識別子を生成するものと定義される．Emp2ID に，新旧の識別子の対応を記憶しておくことで，Emp2 の生成の際に，Emp 中のデータと新しい識別子の対応が取れることになる．

ただし，ILOG のような明示的なオブジェクト識別子を使う言語でいうところのオブジェクト識別子は，物理的なデータ構造とは完全に独立であるので，この例のような場合は，必ずしも Emp2 のために新しい識別子を生成せずとも，以下のように Emp の識別子

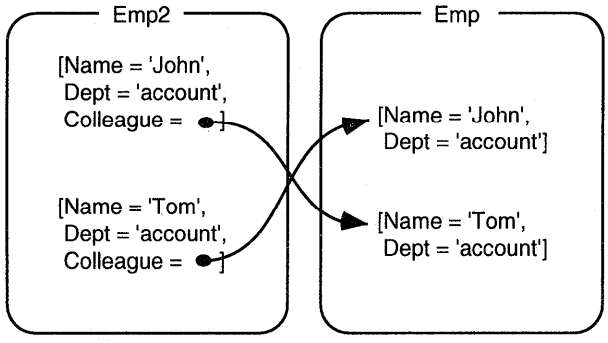


図 7 問い合わせの実際の結果
Fig. 7 The actual reslut

をそのまま使うこともできる.

Emp2(id1, ..., d, id2) ← Emp(id1, ..., d),
Emp(id2, ..., d)

ここでは，前述の as によって実現される操作と同様の操作が ILOG でも記述できることを示すために上のような操作を考えた．ただし，前述のような実際にレコードを生成する言語では，前述のような (Y as Emp2) は参照のみが生成され，その中身にはアクセスできないといった管理を行ってやる必要がある点が ILOG 等の言語とは異なる．

また，ILOG においても，Emp と Emp2 で識別子を共有したくないような場合には，上のような操作が実際に必要になる．この点については，文献 31) の Remark 5.3 を参照されたい．

以上のように，再構成検索は検索の本質であり関係データモデルにおいては当然のものであったが，オブジェクト指向データモデルなどポインタによる参照を含むモデルでは，関係データモデルの場合のように単純ではない．では次に，半構造データに関しては再構成検索に関するどのような研究があるかを見て行く．

5.3 MSL における OID 管理

MSL[9] は，Lorel とは独立に OEM 上に定義された，もう一つの言語である．基本的には論理型の言語であり，Datalog + negation に，以下のような再構成のための機構を加えたものである．

- Skolem 関手によるオブジェクト識別子生成を使った，インクリメンタルなオブジェクトの定義
- 「その他の属性」を表現する rest 変数.

後者の rest 変数は，検索すべきオブジェクトのパターンを指定する時に，一部の属性についてのみパターンを記述し，そのオブジェクトが持っているその他の属性については，rest という名の変数に束縛するものである．

一方，次の例は，Skolem 関手によるインクリメンタルなオブジェクトの生成を行う問い合わせの例である [9].

⟨rep(s), report, set, {⟨id, category, str, s2⟩}⟩
← ⟨_, report, set, {⟨_, repNum, str, s⟩,
⟨id, area, str, s2⟩}⟩
⟨rep(s), report, set, {⟨id, year, int, i⟩}⟩
← ⟨_, report, set, {⟨_, repNum, str, s⟩,
⟨id, year, int, i⟩}⟩
⟨rep(s), report, set, {⟨id, year, int, str2int(s2)⟩}⟩
← ⟨_, report, set, {⟨_, repNum, str, s⟩,
⟨id, year, str, s⟩}⟩

このプログラムでは，report というラベルを持つオブジェクトから，新たにオブジェクトを生成し直す再構成を行っている．その際に，生成されるオブジェクトの識別子として，Skolem 関手を使って，もとのオブジェクトの repNum 属性の値に対して一意であるような識別子を生成している．また，各属性に対して，

- area という属性があったら，category に直す.
- year という整数の属性があったら，そのまま加える.
- year という文字列の属性があったら，整数に変換して加える.

という操作を行う.

半構造データでは，構造がばらばらのオブジェクト集合を扱うため，考えうる属性の組合せのパターンごとに変換ルールを記述するのは煩雑である．よって，ここでは上のように各属性毎にばらばらに変換ルールを記述し，ただし各ルールの rep(s) が，もとのオブジェクトのキーである属性 repNum に対して一意な識別子を生成するようにしている．これにより，

⟨id, report, set, {⟨id2, repNum, str, 'CS-71'⟩,
⟨id3, area, str, 'DB'⟩,
⟨id4, year, str, '1994'⟩}⟩

というオブジェクトがあった場合，前述のルールのう

ち，一番目のルールと三番目のルールの二つがばらばらに起動され，

⟨id5, report, set, {⟨id3, category, str, 'DB'⟩}⟩

と

⟨id5', report, set, {⟨id4, year, int, 1994⟩}⟩

という項が生成されるが，id5 と id5' は，rep('CS-71') という Skolem 関手により，同一の識別子が生成され，これらの項は

⟨id5, report, set, {⟨id3, category, str, 'DB'⟩,
⟨id4, year, int, 1994⟩}⟩

という一つのオブジェクトとして解釈される．

5.4 UnQL における再構成検索

UnQL では，グラフ全体を他のグラフに変換する問い合わせのための構文が用意されている．例えば，以下の問い合わせはグラフ中の全ての Cast というラベルを Featuring というラベルに変換する問い合わせである [8]．

```
traverse DB giving X
    case Cast ⇒ _ then X := {Featuring ⇒ X}
    case \l ⇒ _ then X := {l ⇒ X}
```

この traverse 構文では，変数 DB に束縛されている *tree* の各エッジを，以下に記述された case 文に従って変換していき，最終的に *tree* 全体を変換した別の *tree* を返す．また，X := {Featuring ⇒ X} の部分は，:= の右辺の内容を変換後の *tree* X の対応するエッジの部分の値とすることを表し，最後の ⇒ X の部分の X は，もとの *tree* でそのエッジに参照されていたノードに対応する X 中のノードを意味する．つまり直感的には，前述の文献 30) で用いられた as を使って，

```
traverse DB giving X
    case Cast ⇒ \t then X := {Featuring ⇒
                                (t as X)}
    case \l ⇒ \t then X := {l ⇒ (t as X)}
```

と書くことに相当する．

もう少し複雑な例として，TVshow というラベルを持つエッジの下についてのみ，Cast を Featuring に変更し，Movie の下については Cast のままとする問い合わせを以下に示す [10]．

```
traverse DB giving X, Y
    case TVshow ⇒ _ then X := {TVshow ⇒ Y}
    case Movie ⇒ _ then Y := {Movie ⇒ X}
    case Cast ⇒ _ then Y := {Featuring ⇒ Y}
    case \l ⇒ _ then X := {l ⇒ X},
                     Y := {l ⇒ Y}
```

この例では，X と Y の二つの *tree* を生成し，Y では Cast というエッジを Featuring に変換し，X ではそのままにしている．そして，X の中の TVshow というエッジの先は Y に向け，Y の中の Movie というエッジの先は X にもどしている．これによって，TVshow の下にある Cast だけを変更するという，やや tricky な問い合わせ例である．最終的な問い合わせ結果としては，giving 節に最初に書かれた X が返される事になっている．

ここで，UnQL の理論的基礎についてごく簡単に触れておく．UnQL の理論的基礎となっているのは structural recursion[32),33)] と呼ばれる，再帰的に定義されたデータ型に対する操作を定義するための枠組である．例えば，集合型のデータは以下のような再帰的に構成されるデータ構造として定義可能である．

$$set ::= \{\} \mid \{e\} \mid set \cup set$$

上の定義に沿って考えれば，集合型の値は，空集合であるか，singleton set であるか，singleton set からスタートして任意回の集合和の操作を加えて構成したものとみなせる．

この考え方に基づくと，集合に対する様々な操作 f は以下のようにして，より原始的な値と操作である e, p, u から構成できると考えられる．

$$\begin{aligned} f(S) &= e && \text{if } S = \{\} \\ & p(s) && \text{if } S = \{s\} \\ & u(f(S1), f(S2)) && \text{if } S = S1 \cup S2 \end{aligned}$$

再帰的なデータ型に対する操作をこのような形で定義して，その性質について考察するのが structural recursion の考え方である．

ここで，*tree* の定義を思い出してみると，やはり再帰的に定義されていた．よって，同様の方法で *tree* に対する操作が定義可能である．UnQL では，*tree* に対する操作を以下のように捉えている．

$$\begin{aligned} f(T) &= e && \text{if } T = \{\} \\ & p(l, f(T')) && \text{if } T = \{l \Rightarrow T'\} \\ & u(f(T1), f(T2)) && \text{if } T = T1 \cup T2 \end{aligned}$$

二行目にあらわれる再帰的な f がエッジの縦方向の再帰に対応し，三行目にあらわれる再帰的な f がエッジの横方向のならびに対応していることがわかる．UnQL における traverse の操作のセマンティクスは，このような枠組を使って定義されている．

6. その他の研究と今後の展望

前章までで，半構造データの研究の最も基本となるデータモデルと操作言語に関する研究について概観したので，この章では，半構造データに関するその他の研究について述べる．1998 年の時点で，半構造データに関する研究で特に多くの研究者が関心を持ってい

るトピックは以下の三つと思われる．

- 半構造データのためのスキーマの表現と，スキーマ情報の自動抽出
- 問い合わせ処理のための，索引付け（indexing），最適化，分散実行などの技術
- WWW データ管理などへの具体的な応用

以下，これらについて述べていく．

6.1 スキーマの表現とスキーマ情報の自動抽出

半構造データのためのスキーマというのは矛盾しているようだが，これは従来のスキーマのように完全に一つの構造を規定してしまうスキーマではなく，データベースの構造に関する，ある抽象化された情報を与えようとするものである．半構造データのためのスキーマに関する研究には，文献34) でも述べられているように，大きく分けて以下の二つの方向がある．

- データベースインスタンスに関する制約を表現するスキーマがユーザによって与えられることを場合を想定している研究．これらの研究では，データのグラフ構造に関する制約を表現する枠組みを設計し，あるデータベースインスタンスが，その制約を満たしているかを判定する技術を開発する事が必要となる．
- スキーマはあらかじめ与えられておらず，与えられたデータからスキーマにあたる情報を抽出しようとするもの．この場合は，どのような情報をスキーマ情報として抽出すべきかの決定と，またそのような情報を自動的に抽出するための技術の開発が必要となる．

6.1.1 データインスタンスの制約を表すスキーマ

上述の二つのうち前者の方向は，第2章で，半構造データの研究はデータベースの研究を「スキーマはあるかもしれないし，ないかもしれない」，あるいは「スキーマは一つのデータ構造を決めるか，あるいはもう少し緩く，データ構造に関するある制約を定義する」という前提に拡張する作業であると述べたうちの後者の考え方に対応した方向である．

このような，ある制約を表現していて，その制約を満たす任意の構造をそのインスタンスとするようなスキーマを考えようという研究としては，文献35) がある．文献35) は，エッジラベル付きグラフデータモデルのためのスキーマとして，エッジラベル付きグラフのエッジに，ラベルの代わりにラベルを引数とする述語を付けたものを提案している．例えば，図8(a) は，グラフ中の各パスの中に Dept というエッジは一度しか出て来ないという制約を表すスキーマである．また，図8(b) は，グラフ中の Paper というエッジは必ず Dept というエッジの下にしか出て来ないというスキーマである．

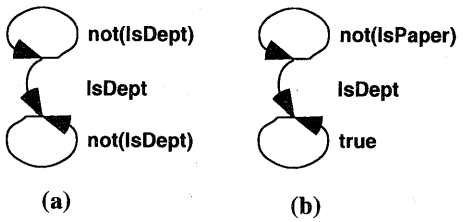


図8 スキーマの例
Fig. 8 An example schema

彼らの定義での，あるグラフがあるスキーマのインスタンスとなる条件は，ごく簡単に言えば，そのグラフ中の各ノード V に対し，あるスキーマ中のノード W が存在して，W が満たすべき条件を V が満たしている事である．よって，この定義ではある種のエッジが存在しないことを要求するスキーマは表現できるが，あるエッジが存在する事を要求するスキーマは表現できない．(例えば，「空の *tree*」は全てのスキーマにマッチする．) これは，彼らがこのスキーマを，問い合わせ処理（query processing）の際に，グラフ中の探索する必要がないとわかっている部分については，はじめから探索しないという最適化のために用いようとしているからである．しかし，逆にあるエッジが存在する事を要求するようなスキーマについて検討する必要性についても指摘している．

6.1.2 半構造データからのスキーマ情報の発見

後者の方向については，より多くの研究がなされているが，これらはさらにいくつかの種類に分類できる．

まず一つめの方向は，グラフ構造全体のおおざっぱな概要を表すものを抽出し，これをスキーマとして利用しようというものである．例えば，文献36) やそれを継承した文献34) では，現在のデータを表すグラフから各ノードが持つ具体的な値は捨象してどのようなラベルのパスがあるかだけを抽出し，簡略化されたグラフに変換したものを Representative Objects あるいは DataGuide と呼んで，スキーマとして用いている．これはつまり，データを表すグラフとラベルのみを見る限り等価となるような最小のグラフといえる．また，その際にどのような等価性を用いるかによって，minimal DataGuide と strong DataGuide という二つのバリエーションを考えている．この問題は，かつての複合オブジェクトの等価性の研究[37] と関連しているので，(もし，そのようなことが必要であれば) さらにいくつかバリエーションが考えられると思われる．これらの研究ではスキーマを，ユーザが問い合わせを

書くために必要なデータベースの内容に関する知識を与えるためのものとして主に用いている．

もう一つの方向は，従来のオブジェクト指向データベースのように，各ノードに対して型を与えようというものである．この方向の研究としては，文献38)とそれを継承した文献39)や，あるいは文献40)などがある．文献39)では，各ノードに入ってくるエッジのラベルの集合と，出ていくエッジのラベルの集合と，さらにそれらのエッジによってつながっているノードの型を調べ，それらが類似しているノードには同一の型を与えるようにしている．もちろん，無制限に型を定義して行けば，最悪の場合ノードの数だけ型が定義される事になるし，あまりに型の数を少なくすると，どの型にも正確には属さないノードの数が多くなる事になるので，ある最適な解を探すことになる．また，隣接する二つのノードの型付けは相互に依存することになるが，この研究では型付けの解を最大不動点を用いて定義している．

さらに別の方向として，半構造データ中の，実世界中の各実体に対応する部分を発見して切り出そうという研究がある．例えば，文献41)，42)では，同一の実体，例えば人物や団体に対応する情報が，WWWページなどの半構造データ中に分散して存在する時に，それらが同一の実体を表していることを判定して，一つに集めてくるためのヒューリスティックについて提案している．これは，関係データベースでキー属性として指定されていた属性にあたるものを発見しようとしていることにあたると考えられる．また，文献43)では，半構造データを表すグラフを各実体に対応する連結部分グラフに分割することによって問い合わせを容易にする事を提案している．これは，従来のオブジェクト指向データベースで composite object として定義されていたものに対応するものを発見しようとしていることにあたると考えられる．

その他にも，従来のデータベースでスキーマ情報の一部として考えられていたようなメタレベルの情報には様々なものが考えられる．よって，それら一つ一つについて，それに対応するような情報を半構造データから抽出する研究が今後の研究課題として考えられるであろう．

6.2 問い合わせ処理

第1章でも述べたように，1999年の ICDT では，半構造データの問い合わせ処理に関するワークショップが開かれる．このことからも，半構造データのための問い合わせ処理に関する研究が強い関心を持たれているのがわかる．

しかし，これまでのところ，半構造データのための問い合わせ処理について触れている論文で主要な雑誌，会議に発表されているのは，文献8)，34)，44)，45)といった程度で，専門のワークショップが開かれるわりには，それほど多くない．この他にも，研究は既に色々行われているようなのであるが，基本的な最適化処理などは従来のオブジェクト指向データベース等での研究で開発された技術を，半構造データ用に修正して適用する形になり，新奇性のある論文になりにくいようである．上に挙げた文献34)などで述べられている問い合わせ処理に関する内容についても，全く新しい技術を開発したというよりは，従来のオブジェクト指向データベースで用いられていた手法と似たような手法を適用し，どのような効果が得られるかについて述べているという感がある．文献44)では，データソースがネットワーク上で分散している環境で，問い合わせを分散処理するための問い合わせ分割について論じており，半構造データが従来の分散データベースのように関係やクラスといった明確な単位で分割されているのではない点に着目して新奇性を出しているようである．その意味では，今後は，文献45)のように，ゆるやかな制約のみを課すスキーマのもとでの，問い合わせの最適化に関する研究が中心になるのではないかと思われる．

6.3 具体的なデータへの応用

当初，半構造データの研究は，第3章で解説したような半構造データ一般を表現できるようなデータモデルを設計し，そのモデルの上で論じられる事が多かった．そのため，WWW や構造化文書などのデータは半構造データであると言いながらも，では具体的に WWW や構造化文書のデータが，どのようにこれらの半構造データのモデルにマップされて，どのように利用されるのかが，やや不明確な部分があった．しかしその後，WWW 等への具体的なデータへの応用に関する論文が出はじめている．例えば，文献46)では，半構造データの操作言語を用いた WWW サイトの管理について述べている．この研究では WWW サイトの生成を，以下の三段階に分けて考えている:

- 文書，画像などのデータを様々なデータソースから集め，
- これらをデータの単位として，WWW サイトの概念構造を構築し，
- 概念構造を表現しているデータを具体的な表現上の情報などを含む HTML データに変換する

そして，この二番目の WWW サイトの概念構造の表現とその操作に，半構造データを用いている．つま

り，HTML データそのものを半構造データのための操作言語で操作しようというのではなくて，WWW データの管理を抽象的な概念構造の操作と，具体的な HTML データの生成とに分けて考えて，前者の段階に半構造データのデータモデルと操作言語を用いようというのである．

その他にも，文献 47) では，一般のテキストデータをパーズして半構造データに変換するための枠組みについて，また，文献 48) では，関係データベースの内容を WWW ページなどの構造に変換するための枠組みについて提案している．また，XML のための操作言語に関する提案も既に WWW Consortium へ提出されている [49]．

6.4 今後の展望

第 2 章で述べたように，半構造データに関する研究は，スキーマに関する前提を変更して，データベースの教科書を修正していく作業にあたると考えれば，今後，他のさまざまなデータベースに関する技術と「半構造データの」という文脈の組合わせが研究課題となりうる．例えば，半構造データに temporal database の概念を組み合わせた研究 [50] や，半構造データ上の実体化ビューのメンテナンスに関する研究 [44]，また，ゆるやかなスキーマの研究とも関連するが，半構造データ上の一貫性管理に関する研究 [51] などが既に行われている．

また，上でも述べたが，これまでのデータベースではメタレベルのスキーマ情報として扱われていたような様々な情報を半構造データから自動抽出するための研究も，まだ研究の余地があると思われる．さらに，操作言語に関する研究も，決して終結したわけではなく，文献 52)，53) 等の論文が発表されており，まだ発展する可能性がありそうである．

7. ま と め

最後に，本稿において特に強調したい点について，再度，簡単にまとめておく．

- 半構造データは schemaless で self-describing なデータである．
- 半構造データに関する研究は，従来のスキーマの存在を仮定したデータベースのモデルを，スキーマはあるかもしれない無いかもしれないという形に拡張する研究である．
- スキーマ情報とデータを区別しないデータモデルが重要である．
- 本来，問い合わせ＝再構成であった．半構造データモデルにおいても再構成検索を記述できる操作言語が重要になる．

謝辞 研究活動を支援していただいている田中克己教授に謝意を表します．本研究の一部は，文部省科学研究費重点領域研究「高度データベース」（領域番号 275 （08244103））の援助を受けており，また，本研究の一部は，日本学術振興会未来開拓学術研究推進事業における研究プロジェクト「マルチメディア・コンテンツの高次処理の研究」によっています．ここに記して謝意を表すものとします．

参 考 文 献

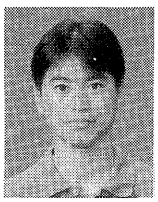
1) *Proceedings of the Workshop on Management of Semistructured Data (in Conjunction with ACM PODS/SIGMOD '97)* (1997). Available at `http://www.research.att.com/~suciu/workshop-papers.html`.
2) Abiteboul, S.: Querying Semi-Structured Data, *Proc. of ICDT*, LNCS, Vol. 1186, Springer-Verlag, pp. 1–18 (1997).
3) Buneman, P.: Semistructured Data, *Proc. of ACM PODS*, pp. 117–121 (1997).
4) Papakonstantinou, Y., Garcia-Molina, H. and Widom, J.: Object Exchange across Heterogeneous Information Sources, *Proc. of IEEE ICDE*, pp. 251–260 (1995).
5) Papakonstantinou, Y., Garcia-Molina, H. and Ullman, J.: Medmaker: A Mediation System Based on Declarative Specification, *Proc. of IEEE ICDE*, pp. 132–141 (1996).
6) Thierry-Mieg, J. and Durbin, R.: Syntactic Definitions for the ACeDB Database Manager, Technical report, MRC Lab. for Molecular Biology, Cambridge, CB2 2QH, UK (1992).
7) Buneman, P., Davidson, S. B., Hart, K., Overton, C. and Wong, L.: A Data Transformation Systems for Biological Data Sources, *Proc. of VLDB*, pp. 158–169 (1995).
8) Buneman, P., Davidson, S., Hillebrand, G. and Suciu, D.: A Query Language and Optimization Techniques for Unstructured Data, *Proc. of ACM SIGMOD*, pp. 505–516 (1996).
9) Papakonstantinou, Y., Abiteboul, S. and Garcia-Molina, H.: Object Fusion in Mediator Systems, *Proc. of VLDB*, pp. 413–424 (1996).
10) Buneman, P., Davidson, S. and Suciu, D.: Programming Constructs for Unstructured Data, *Proc. of Int. Workshop on DBPL*, electronic Workshops in Computing, Springer-Verlag (1995). Available at `http://www.springer.co.uk/eWiC/Workshops/DBPL5.html` (アクセスにはパスワードのために Springer の booklet が必要).

11) Atkisson, M., Bancilhon, F., DeWitt, D., Dittrich, K. R., Maier, D. and Zdonik, S. B.: The Object-Oriented Database System Manifesto, *Proc. of Int. Conf. on Deductive and Object-Oriented Database Systems (DOOD)*, Elsevier Science Publ., pp. 40–57 (1989).
12) Lécluse, C., Richard, P. and Velz, F.: O_2, an Object-Oriented Data Model, *Proc. of ACM SIGMOD*, pp. 424–433 (1988).
13) Bretl, R., Maier, D., Otis, A., Penney, J., Schuchardt, B., Stein, J., Williams, E. H. and Williams, M.: The GemStone Data Management System, *Object-Oriented Concepts, Databases, and Applications* (Kim, W. and Lochovsky, F. H.(eds.)), Addison-Wesley, pp. 283–308 (1989).
14) Kim, W., Ballou, N., Chou, H.-T., Garza, J. F. and Woelk, D.: Features of the ORION Object-Oriented Database System, *Object-Oriented Concepts, Databases, and Applications* (Kim, W. and Lochovsky, F. H.(eds.)), Addison-Wesley, pp. 251–282 (1989).
15) LaLonde, W. R., Thomas, D. A. and Pugh, J. R.: An Exemplar Based Smalltalk, *Proc. of ACM OOPSLA*, pp. 322–330 (1986).
16) Sciore, E.: Object Specialization, *ACM TOIS*, Vol. 7, No. 1, pp. 101–122 (1989).
17) Tanaka, K., Nishio, S., Yoshikawa, M., Shimojo, S., Morishita, J. and Jozen, T.: Obase Object Database Model: Towards a More Flexible Object-Oriented Database System (invited paper), *Proc. of International Symposium on Next Generation Database Systems and Their Applications*, pp. 159–166 (1993).
18) Abiteboul, S., Quass, D., McHugh, J., Widom, J. and Wiener, J. L.: The Lorel Query Language for Semistructured Data, *International Journal of Digital Libraries*, Vol. 1, No. 1, pp. 68–88 (1997).
19) Gyssens, M., Paradaens, J. and Van Gucht, D.: Graph-Oriented Object Model for Database End-User Interface, *Proc. of ACM SIGMOD*, pp. 24–33 (1990).
20) Gyssens, M., Paradaens, J., Van den Bussche, J. and Van Gucht, D.: A Graph-Oriented Object Database Model, *IEEE Trans. on Know. and Data.*, Vol. 6, No. 4, pp. 572–586 (1994).
21) Amann, B. and Scholl, M.: Gram: A Graph Data Model and Query Language, *Proc. of ACM Hypertext*, pp. 201–211 (1992).
22) Consens, M.P. and Mendelzon, A.O.: Expressing Structural Hypertext Queries in GraphLog, *Proc. of ACM Hypertext*, pp. 269–292 (1989).
23) Consens, M. P. and Mendelzon, A. O.: GraphLog: A Visual Formalism for Real Life Recursion, *Proc. of ACM PODS*, pp. 404–416 (1990).
24) Minohara, T., Watanabe, R. and Tokoro, M.: Queries on Structures in Hypertext, *Proc. of Foundations of Data Organization and Algorithms (FODO)*, LNCS, Vol. 730, Springer-Verlag, pp. 394–411 (1993).
25) Mylopuolos, J., Bernstein, P. and Wong, H.: A Language Facility for Designing Database-Intensive Applications, *ACM TODS*, Vol. 5, No. 2, pp. 185–207 (1980).
26) Kifer, M., Kim, W. and Sagiv, Y.: Querying Object-Oriented Databases, *Proc. of ACM SIGMOD*, pp. 393–402 (1992).
27) Kim, W.: A Model of Queries for Object-Oriented Databases, *Proc. of VLDB*, pp. 423–432 (1989).
28) Ullman, J. D.: *Principles of Database and Knowledge-Base Systems*, Vol. I, II, Computer Science Press (1988).
29) Cattell, R. G. G.: *The Object Database Standard: ODMG-93*, Morgan Kaufmann (1994).
30) Nishimura, S., Ohori, A. and Tajima, K.: An Equational Object-Oriented Data Model and its Data-Parallel Query Language, *Proc. of ACM OOPSLA*, pp. 1–17 (1996).
31) Hull, R. and Yoshikawa, M.: ILOG: Declarative Creation and Manipulation of Object Identifiers, *Proc. of VLDB*, pp. 455–468 (1990).
32) Tannen, V., Buneman, P. and Naqvi, S. A.: Structural Recursion as a Query Language, *Proc. of Int. Workshop on DBPL*, pp. 9–19 (1991).
33) Buneman, P., Naqvi, S. A., Tannen, V. and Wong, L.: Principles of Programming with Complex Objects and Collection Types, *Theoretical Computer Science*, Vol. 149, No. 1, pp. 3–48 (1995).
34) Goldman, R. and Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, *Proc. of VLDB*, pp. 436–445 (1997).
35) Buneman, P., Davidson, S., Fernandez, M. and Suciu, D.: Adding Structure to Unstructured Data, *Proc. of ICDT*, LNCS, Vol. 1186, Springer-Verlag, pp. 336–350 (1997).
36) Nestorov, S., Ullman, J. D., Widom, J. and Chawathe, S.: Representative Objects: Concise Representations of Semistructured, Hierarchical Data, *Proc. of IEEE ICDE*, pp. 70–90 (1997).
37) Khoshafian, S. and Copeland, G. P.: Object Identity, *Proc. of ACM OOPSLA*, pp. 406–416

(1986).
38) Nestorov, S., Abiteboul, S. and Motwani, R.: Inferring Structure in Semistructured Data, *Proc. of Workshop on Management of Semistructured Data (in Conjunction with ACM PODS/SIGMOD '97)* (1997). Available at `http://www.research.att.com/~suciu/workshop-papers.html`.
39) Nestorov, S., Abiteboul, S. and Motwani, R.: Extracting Schema from Semistructured Data, *Proc. of ACM SIGMOD*, pp. 295–306 (1998).
40) Seo, D.-Y., Lee, D.-H., Lee, K.-M. and Lee, J.-Y.: Discovery of Schema Information from a Forest of Selectively Labeled Ordered Trees, *Proc. of Workshop on Management of Semistructured Data (in Conjunction with ACM PODS/SIGMOD '97)* (1997). Available at `http://www.research.att.com/~suciu/workshop-papers.html`.
41) Huffman, S. and Baudin, C.: Notes Explorer: Entity-Based Retreival in Shared, Semistructured Information Spaces, *Proc. of ACM CIKM*, pp. 99–106 (1996).
42) Nado, R. and Huffman, S.: Extracting Entity Profiles from Semi-structured Information Spaces, *Proc. of Workshop on Management of Semistructured Data (in Conjunction with ACM PODS/SIGMOD '97)* (1997). Available at `http://www.research.att.com/~suciu/workshop-papers.html`.
43) Tajima, K.: Querying Composite Objects in Semistructured Data, *Proc. of FODO* (1998).
44) Suciu, D.: Query Decomposition and View Maintenance for Query Languages for Unstructured Data, *Proc. of VLDB*, pp. 227–238 (1996).
45) Fernández, M. F. and Suciu, D.: Optimizing Regular Path Expressions Using Graph Schemas, *Proc. of IEEE ICDE*, pp. 14–23 (1998).
46) Fernández, M.F., Florescu, D., Kang, J., Levy, A. and Suciu, D.: Catching the Boat with Strudel: Experiences with a Web-Site Management System, *Proc. of ACM SIGMOD*, pp. 414–425 (1998).
47) Adelberg, B.: NoDoSE — A Tool for Semi-Automatically Extracting Semi-Structured Data from Text Documents, *Proc. of ACM SIGMOD*, pp. 283–294 (1998).
48) Toyama, M. and Nagafuji, T.: Dynamic and Structured Presentation of Database Contents on the Web, *Proc. of EDBT*, LNCS, Vol. 1377, Springer-Verlag, pp. 451–465 (1998).
49) Deutsch, A., Fernandez, M., Florescu, D. and Suciu, D.: XML-QL: A Query Language for XML (submission to the World Wide Web Consortium) (1998). Available at `http://www.w3.org/TR/NOTE-xml-ql`.
50) Chawathe, S. S., Abiteboul, S. and Widom, J.: Representing and Querying Changes in Semistructured Data, *Proc. of IEEE ICDE*, pp. 4–13 (1998).
51) Buneman, P., Fan, W. and Weinstein, S.: Path Constraints on Semistructured and Structured Data, *Proc. of ACM PODS*, pp. 129–138 (1998).
52) Mendelzon, A. O. and Milo, T.: Formal Models of Web Queries, *Proc. of ACM PODS*, pp. 134–143 (1997).
53) Arocena, G. O. and Mendelzon, A. O.: WebOQL: Restructuring Documents, Databases, and Webs, *Proc. of IEEE ICDE*, pp. 24–33 (1998).

(平成 10 年 9 月 20 日受付)
(平成 10 年 12 月 27 日採録)

(担当編集委員 **横田 一正**)



田島 敬史(正会員)

1991 年東京大学理学部情報科学科卒業. 1993 年東京大学大学院理学系研究科情報科学専攻修士課程修了. 1994 年より 1996 年まで京都大学数理解析研究所研究生. 1996 年東京大学大学院理学系研究科情報科学専攻博士課程修了. 理学博士. 同年より神戸大学工学部情報知能工学科助手. 主にデータベースシステムの研究に従事. 情報処理学会, ソフトウェア科学会, 各会員.