データベースシステム 113－2
（1997. 7. 14）

複合型階層
On Complex Type Hierarchy

三浦孝夫 (Takao MIURA)　　塩谷勇 (Isamu SHIOYA)

産能大学 (SANNO College)
伊勢原市上粕屋 1573, 259-11
Kamikasuya 1573, Isehara, Kanagawa 259-11 JAPAN

Abstract

本研究では複合オブジェクトのための型階層に関する理論的基礎を論じる。この主たる目的は、モデル化のための基本機能、その意味および一貫性制約の記述力を拡張することにある。複合オブジェクトは、実体 (オブジェクト) とデータ構造値の組から構成されるが、データ構造の構成方法は特定のものを仮定しない。つまり、集合やタプル構成だけでなく、一般の構造構成子を用いる。

ここでは各複合オブジェクトが内包情報を有するインスタンス指向データベースを仮定し、どの情報も固有の内包情報 (型スキーマという) を持つとする。一般には、各インスタンスには無数の複合型が対応するが、記述を簡単にして近汎 (Least General) なものだけを選び、この記述が有限となる条件を示す。本研究の根幹には、複合オブジェクト間にどのような順序が導入できるかという問題がある。

In this work, we discuss the theoretical foundation for type hierarchy of complex objects to extend modelling concepts, semantics and constraints. Complex objects consist of entities and data-structure instances. We assume general constructions of data structures, i.e., not only sets and tuples but any structuring operators are allowed.

Our discussion uses an *instance-based* data model and each instance has its own intentional information called type schemes. Since there may be infinite complex type information to each instance, we explore simplified description in such a way that type schemes contain only *least general* types. We show the finiteness property of such description. The heart of our approach exists in semantic orders for of complex objects.

1 動機

データベースでは対象となる情報は予めスキーマの形で事前に分類され、しかもすべてのデータは問題領域とは独立に操作できるという仮定がある。これらの仮定によって、対象世界はデータモデルが提供する基本機能の組み合わせに依って記述され操作される。このフレームワークはデータベースパラダイムと呼ばれる[7]。各インスタンスは事実と対応し、スキーマはそれらに関する知識記述となる。データベース利用者およびデータベースシステムが処理の意図を確認し最適化方法を検証できる基本には、この仮定がある。その結果、大量のデータに対する適切で効率よい処理を行うきっかけを得る。

データベーススキーマは、一般には情報構造 (どのような情報をどのようにモデル化するかに関する記述) と一貫性制約 (すべてのインスタンスを意味の有る状態に保つ条件) を含んでいる。前者はデータ抽象化とも呼ばれる。データベースでは、型や述語はそれぞれ実体集合や連想集合に対応し、インスタンス (外延) から性質 (内包) を区別する。これが特別な言語を用いる理由であり、構文が内包性に、解釈が外延に対応する。複合オブジェクトの概念は新たな抽象化技法であり、利用者は複雑な関連を高水準の概念単位として記述することができる。利用者はデータ構造に関する知識を理解しており、複合オブジェクトを記述するのにふさわしい。

この分野は過去広範囲な研究がなされているしかし、ほとんどの議論は特定のデータ構造構成子 (例えば、`setof` や `recordof`) に基づいており、研究結果は構成子の性質に大きく依存する。一般化された構成子に関する議論は、これまでのところほとんどなされていない。

内包情報に成り立つ性質のうち内、特に `ISA` は重要である。これは対応する外延集合の間の包含条件を意味し、データ抽象化との関係からも、データモデリングの観点[1]からも知識表現の観点[2]からも大きい注目を浴びてきた。

本研究では、一般的な複合オブジェクトの型階層に関する考察を行う。特に、`ISA` と一般のデータ構造構成子の間の関連を明らかにする。この結果はデータモデルの機能を拡張するための基礎を与えるものとなる。これまでのところ、同種の議論を展開した研究を筆者らは知らない。

ここでは各複合オブジェクトが内包情報を有するインスタンス指向データベースを仮定し、どの情報も固有の内包情報 (型スキーマという) を持つとする。一般には、各インスタンスには無数の複合型が対応するが、記述を簡単にして近汎なものだけを選び、この記述が有限となる条件を示す。本研究は [10] の要約であり、詳細は該当文献を参照されたい。

2 実体に基づくデータモデル

2.1 基本概念

実体は対象世界の'もの'(object) を代理者であり、表現方法や属性特徴値などと独立して計算機世界に直接的に記述する方法である[?]。本稿では実体の集合を $\mathcal{E}$ と表す。実体型は'もの' が共通して持つ性質を表す内包概念であり、このとき実体 e がその型 t を持つという。実体型の集合は $\mathcal{T}$ によって表し、実体型 t の実体集合を $\Gamma(t)$ と表す。ひとつの実体は複数の型を持つことがある。特に、特殊な型 `@E` によってデータベース中のすべての実体を表す。

連想は'もの' の間に成り立つ関連を表し、ひとつの関連はひとつの連想に対応する。連想の間に成り立つ共通した性質は述語として捉えられ、 実体 $e_1,..,e_n$ の間の述語 p の連想は $p(e_1...e_n)$ と表す。述語には予め決められた実体型 (定義型) が対応する。述語 p が $t_1,..,t_n$ を定義型とするとき、$e_i \in \Gamma(t_i)$ となる。述語 p の連想集合を $\Gamma(p)$ で表す, $\Gamma(p) \subseteq \Gamma(t_1) \times ... \times \Gamma(t_n)$ である。属性は、述語 $f(t_1t_2)$ で定義型 t_1 and t_2 の間に関数対応があるものをいう。このとき型 t_1 の各実体 e に対して高々ひとつの f 連想 $f(e,a)$ (ただし $a \in \Gamma(t_2)$) が存在する。

複合オブジェクト概念は実体とデータ構造値との組で表される。本来、データ構造は効率よい計算処理のための情報構造の実現方法であり、簡潔で固有の表現や操作をもたらす。複合型とはデータ構造の構成方法を規定しており、複合型 d に対応して複合値集合 $\Gamma(d)$ を考えることができる。型 t の実体 e と複合型 d の複合値 v の間の対応を$\lambda_d(e) = v$ または $e : v$ と表し、複合オブジェクトという[7]。実体 e は意味を、複合値 v は構造特性を表し、λ_d は t, d の間の述語と見れる。

[EXAMPLE 1] 以下では *Project Activity* データベースを例として用いる。ここには次のような実体型が用意されている : `Project`, `Authority`, `Committee`, `LeadersCommittee`, `CommitteeMember`, `OperatorGroup`, `ExpertList`, `Checker`, `Worker`, `QualifiedPerson`, `Leader`, `Chair`, `Person`, `Experience`。これらの意味は自明であろう。各型には属性が対応するものがある: `Project` は `Goal` と `Content` を, `Committee` は `Chair` と `Name` を, `OperatorGroup` は `Leader` を属性としている。本稿では用いないが、4 つの述語 `evaluate (Project, Autority)`, `supervise (Project, Committee)`, `work (Project, OperatorGroup)`, `nominate (Project, ExpertList)` がある。これらの意味も自明であろう。複合オブジェクトの例はあとで示す。

図 1 は *AIS* ダイアグラムを示している。これはスキーマの一部を図示しており、円は実体型や複合型を, 菱型は述語を、辺は述語定義を表す。矢印は属性対応を表す。包含関係は後述する `ISA` 関連を表している。円を重ね合わせる事で、型が実体や複合値を共有することを強調できる。□

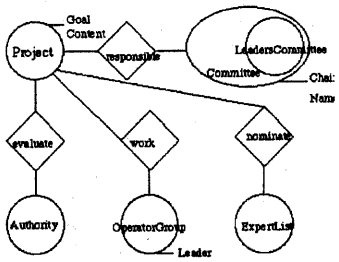

図 1: Project Activity Database Diagram

どの実体もその型情報を含んでいるが、AIS モデルや他の多くの意味モデルはこの意味で実現値ベースである[3]。$\tau(e)$ を実体 e の持つ実体型の集合とする: $\tau(e) = \{t \in \mathcal{T} \mid e \in \Gamma(t)\}$ これを e の型スキーマという。$\tau(e)$ は空ではなくまた有限とする。$\tau(e)$ 全体によってデータベースの実体型付けを表現しており、これをデータベースの型スキーマという。データベースが大量のデータを含むなら、個々に $\tau(e)$ を管理するため、スキーマの知識を効率よく用いることは困難となる。型階層や一般的な型制約などの型に関する知識を用いて、統一的で単純な機構が不可欠である。

型を制御する機構の代表的なものが汎化または `ISA` である。型 t_1, t_2 に対して、t_2 が t_1 の汎化 (または t_1 `ISA` t_2) であるとは、$\Gamma(t_1) \subseteq \Gamma(t_2)$ が成り立つときをいう。複数回適用することによって `ISA` 階層を得る。2 つの型 t_1, t_2 に対して、t_2 が t_1 の直接汎化であるとは、t_1 `ISA` t_2 で t_1 `ISA` t_3, t_3 `ISA` t_2 となる型 t_3 が存在しないときをいう。`ISA` は半順

序となる。

実体 e が型 t を持ち t ISA t' ならば、e は型 t' も持つ、つまり $t \in \tau(e)$ と t ISA t' から $t' \in \tau(e)$ が成り立つ。これを型整合性条件という。以下では、型整合性条件が成り立つとする。

[EXAMPLE 2] 例 1 において ISA 条件が与えられている。例えば、図 1 において、LeadersCommittee ISA Committee とは、リーダから構成されるどの委員会も (公式の) 委員会であることを表す。

```
Leader ISA QualifiedPerson
Leader ISA CommitteeMember
Chair ISA CommitteeMember
QualifiedPerson ISA Worker
Checker ISA Worker
Worker ISA Person
CommitteeMember ISA Person
```

このような ISA を用いて Leader ISA Person などを推論できる。図 2 はこのような ISA 階層を示している。Person 実体に関する型スキーマを示す:

entity	types
Beethoven	Leader, Worker, QualifiedPerson, CommitteeMember
Brahms	Leader, Worker, QualifiedPerson, CommitteeMember
Bruckner	Worker, QualifiedPerson, CommitteeMember
Chopin	Worker
Liszt	Worker
Puccini	Leader, Worker, QualifiedPerson, CommitteeMember
Schumann	Worker
Schubert	Worker
Sibelius	CommitteeMember
Verdi	Leader, Worker, QualifiedPerson, CommitteeMember
Wagner	CommitteeMember, QualifiedPerson, Checker,Worker

□

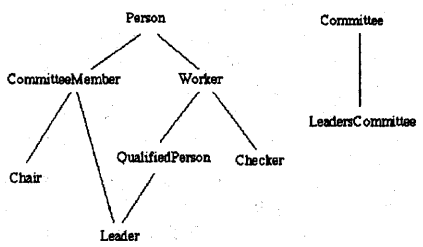

図 2: ISA Hierarchies in Project Activity Database

2.2 複合型と型スキーマ

$\mathcal{D}_U$ および $\mathcal{V}_U$ を、それぞれデータ構造とその実現値集合とする。以下でデータ構造を考えるときは、有限個の構成子 $\mathcal{C}$ を仮定し、$\mathcal{C}$ に関する複合型の集合を $\mathcal{D}(\mathcal{C})$ または $\mathcal{D}$ と表す。構成子 η は $\mathcal{D}_U^n$ から $\mathcal{D}_U$ への写像と見れる (n を η の位数という)。例えば setof, recordof(n), arrayof(n) は、集合、レコード、配列といったデータ構造を '作り出す' 構成子である。以下では id (自己写像) を $\mathcal{C}$ に含まないとする。構造構成子 $\eta, \eta_1, .., \eta_n \in \mathcal{C}$ が $\eta : \mathcal{D}_U^n \to \mathcal{D}_U$、$\eta_i : \mathcal{D}_U^{k_i} \to \mathcal{D}_U, i = 1, .., n$, となるとき、これらの合成 $\eta(\eta_1 ... \eta_n)$ は $\mathcal{D}_U^{k_1+..+k_n}$ to $\mathcal{D}_U$, となる新たな構造構成子と考えることができる。これを構造構成と呼ぶ。

データ構造実現値を論じるとき、$\mathcal{V}_U$ 上のデータ構造化操作 ζ によってデータ構造の意味を定義する必要がある。例えばレコード (*records*) は $[v_1, .., v_n]$ の形の値の集まり (n は共通)、配列 (*arrays*) は $(v_1, .., v_n)$ の形の値の集まり (n は共通)、集合 (*sets*) は $\{v_1, .., v_n\}$ の形の値の集まり ($n \geq 0$ は各値に依存) と定義される。データ構造化操作の集まりを $\mathcal{C}'$ と表し、自己写像 *identity* は特別な操作とする。

$\mathcal{D}_U$ 上の構造構成子の有限集合 $\mathcal{C} = \{\eta_1, ..., \eta_k\}$ に対して、複合型集合 $\mathcal{D}(\mathcal{C})$ は形式的に次のように定義される:
(1) どの型 $t \in \mathcal{T}$ も $\mathcal{D}(\mathcal{C})$ の要素である
(2) $\eta \in \mathcal{C}$ (位数 n) と $t_1, .., t_n \in \mathcal{D}(\mathcal{C})$ に対して、$\eta(t_1 ... t_n)$ は $\mathcal{D}(\mathcal{C})$ に含まれる
(3) 各 η_i, η_j に対して $\eta_i(t_1 ... t_{n_i}) = \eta_i(s_1 ... s_{n_j})$, ならば $i = j$ かつ $t_k = s_k$ $(k = 1, .., n)$ となる。即ち、どの複合型も一意な方法で構成できる。

$\mathcal{T}$ と $\mathcal{D}(\mathcal{C})$ を区別するため、$t \in \mathcal{T}$ を基本型とよび、(1),(2) で定義される型を複合型という。(3) はどの構造構成も自由な解釈を許すことを意味する。

同様に、データ構造化操作の集合 $\mathcal{C}' = \{\zeta_1, ..., \zeta_n\}$ (ζ_j は位数 n_j) によって得られる複合値の集まり $\mathcal{V}(\mathcal{C}')$ を次のように定義する:
(1) $\mathcal{E} \subseteq \mathcal{V}(\mathcal{C}')$
(2) $\zeta \in \mathcal{C}'$ (位数 n) と $V_i \subseteq \mathcal{V}(\mathcal{C}')$, $i = 1, .., n$ に対して、$\zeta(V_1 ... V_n) \subseteq \mathcal{V}(\mathcal{C}')$
(3) 構造構成 ζ_i, ζ_j で *identity* を含まないならば、$\zeta_i(V_1 ... V_{n_i})$ and $\zeta_j(U_1 ... U_{n_j})$. に共通の値は存在しない。
(1) で導入される実体は基本値、(1),(2) の双方で構成される値は複合値と呼ばれる。定義 (2) において、$v \in \zeta(V_1 ... V_n)$ は $v_i \in V_i$ または $v_i \subseteq V_i$ $(i = 1, .., n)$ から構成される、つまり v は関数ではないが ζ を用いた表現という意味で、$v(v_1, .., v_n)$ と表される (ζ-表現ともいう)。性質 (3) からこの表現は一意である。v_i は $v(v_1 .. v_n)$ に生じる。例えば、集合値 $v \in 2^V$ で $a_1, .., a_m \in V$ からなる場合、$v_1 = \{a_1, .., a_m\}$ は $v(v_1)$ と表現できる。同様に、$v \in V_1 \cap V_2$ となる値 v は v 自身で表現される。*identity* を用いた記述は自明な記述と呼ばれる。

$\mathcal{D}(\mathcal{C})$ の解釈を定義する。構成子集合 $\mathcal{C}$ と操作集合 $\mathcal{C}'$ が与えられたとき、各 $\eta \in \mathcal{C}$ に $\zeta \in \mathcal{C}'$ を対応させ、これを η の意味と呼ぶ。$t \in \mathcal{D}(\mathcal{C})$ の解釈 $\Gamma(t)$ を次のように定義する:
(1) 基本型 $t \in \mathcal{D}(\mathcal{C})$ の解釈 $\Gamma(t)$ を実体型の解釈とする、さらに (2) 複合型 $t = \eta(t_1 ... t_n)$ の解釈 $\Gamma(t)$ を $\zeta(\Gamma(t_1), .., \Gamma(t_n))$ とする。

ζ 表現について言えば、$v \in \Gamma(\eta(t_1 ... t_n))$ は $v(v_1, .., v_n)$ と表される、ここで $v_i \in \Gamma(t_i)$ または $v_i \subseteq \Gamma(t_i), i = 1, .., n$ である。

本稿では、上述の枠組みでモデル化できる様々な構造構成子を論じる。例えば、setof 構成子は t を t の集合型へ写像し、setof(t) の解釈を $\Gamma(t)$ の部分集合族とする。listof も同様。recordof(2) 構成子は t_1, t_2 を $\Gamma(t_1) \times \Gamma(t_2)$ の値の組を解釈とするレコード型に対応させる。

構成子	表現/解釈
`unionof`(t_1, t_2)	*trivial*
	$\Gamma(t_1) \cup \Gamma(t_2)$
`intersectionof`(t_1, t_2)	*trivial*
	$\Gamma(t_1) \cap \Gamma(t_2)$
`differenceof`(t_1, t_2)	*trivial*
	$\Gamma(t_1) - \Gamma(t_2)$
`setof`(t)	$\{v_1, .., v_n\}$
	$2^{\Gamma(t)}$
`listof`(t)	$(v_1, .., v_n)$
	$\Gamma(t)$ 上のリスト
`arrayof(n)`(t)	$(v_1, .., v_n)$
	$\Gamma(t) \times .... \times \Gamma(t)$ (n times)
`recordof(n)`$(t_1, .., t_n)$	$[v_1, ..., v_n]$
	$\Gamma(t_1) \times .... \times \Gamma(t_n)$
`treeof(n)`$(t, t_1, .., t_n)$	$[v; v_1, ..., v_n]$
	$\Gamma(t) \times \Gamma($`listof`$(t_1)) \times ....$
	$\times \Gamma($`listof`$(t_n))$
	unique on $\Gamma(t)$

複合型 $t \in \mathcal{D}$ には複合値集合 $\Gamma(t)$ が対応する: $\Gamma(t) = \{v \in \mathcal{V}(\mathcal{C}') \mid v$ of a type $t\}$. 複合値 v は複数の複合型 $\tau(v)$ をもつことがある: $\tau(v) = \{t \in \mathcal{D}(\mathcal{C}) \mid v \in \Gamma(t)\}$ これを v の複合型スキーマ (*complex type scheme*) という。

基本型、構成子、実体集合のいずれもが有限であっても、$\Gamma(t)$ および $\tau(v)$ が無限となることがある。例えば `listof` 構成子は有限個の実体から無数の複合値を生成する。

[EXAMPLE 3] 例 1 では, 詳細を表す複合型が定義されている。複合オブジェクト型 `Committee` は、`Committee` と `setof (CommitteeMember)` との組として定義され、委員会の組織を記述している。これは `Committee ≡ setof (CommitteeMember)` と表され、$e : \{e_1, .., e_n\}$ は委員会 e が委員会メンバ $e_1, .., e_n$ からなることを示す。
同様に複合オブジェクトが与えられている:

```
OperatorGroup ≡ setof (Worker)
Authority ≡ setof (Checker)
LeadersCommittee ≡ setof (Leader)
                ExpertList ≡ listof( recordof(2)
(QualifiedPerson, Experience) )
```

図 3 は、上記の複合オブジェクトの状況を表すダイアグラムとなっている。矢印で ≡ を伴うものは複合オブジェクト定義を意味し、(属性以外の) 矢印は構成方法を表している。

```
complex value
  complex types
{ Beethoven,Brahms }
  setof(Leader), setof(QualifiedPerson),
  setof(Worker), setof(CommitteeMember)
{ Puccini, Verdi }
  setof(Leader), setof(QualifiedPerson),
  setof(Worker), setof(CommitteeMember)
{ Sibelius, Brahms }
  setof(CommitteeMember)
{ Wagner }
  setof(CommitteeMember),setof(QualifiedPerson),
  setof(Checker), setof(Worker)
{ Chopin, Liszt }
  setof(Worker)
{ Schumann,Schubert }
  setof(Worker)
{ Brahms,Bruckner }
  setof(CommitteeMember), setof(QualifiedPerson)
```

□

動的な複合値とは、予め与えられた生成規則を用いて宣言的に生成されるデータ構造値であって、演繹閉包と呼ばれる。以下ではこの生成方法をデータ構造の構成子 `closureof` とみなす[6]。演繹閉包の基本的なアイデアは演繹集合である。これはホーン節の形で与えられた生成規則の最小モデルとして定義される[4]。定義しようとする複合値集合 $\mathcal{V}_U \supseteq \mathcal{E}$、空ではない有限集合 $X \subseteq \mathcal{E}$ および有限生成規則集合 F で各規則が $\mathcal{V}_U$ で閉じているものを考える: $f^n : \mathcal{V}_U^n \to \mathcal{V}_U$。このとき、$Y \subseteq \mathcal{V}_U$ が F による X の演繹領域であるとは、(1) $X \subseteq Y$ であり (2) 各 $f^n \in F$ と $x_1, \cdots x_n$ in Y に対して $f^n(x_1 \cdots x_n) \in Y$ となるときを言う。この様な Y すべての共通部を F による X の演繹閉包と言う。型 $t \in \mathcal{T}$ と生成規則集合 F に対して、`closureof(F)(t)` を $\Gamma(t)^*_F$ を解釈とする型とする。データ構造は固有の操作が必要であるが、もしどの要素も生成の方法が一意であれば、操作関数を宣言的に定義することができるという性質がある。

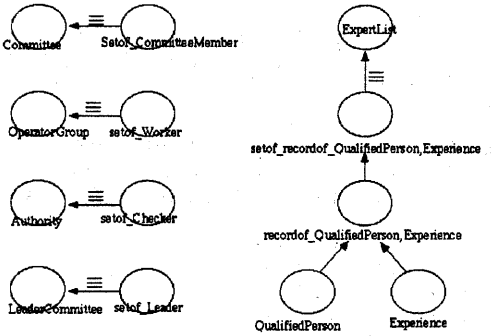

図 3: Complex Objects in Project Activity Database

[EXAMPLE 4] $\Sigma = \{a, b, c\}$ とし、a, b からなる文字列領域 S を考える。生成規則集合 F を $F = \{con_a : S \to S, con_a(x) = xa; con_b : S \to S, con_b(x) = xb\}$、$t$ を $\Gamma(t) = \{a, b\}$ とする型とすれば、`closureof`$(F)(t)$ は $\{a, b\} \cup \{aa, bb, ab, ba\} \cup \{aaa, aab, \cdots\} \cup \cdots$ を解釈にもつ。

操作関数として文字列の長さを得る関数 *length* を定義しよう。$\Gamma(t)$ 上で $length(x) = 1$ とし、`closureof`$(F)(t)$ の要素 $\alpha \cdot a$ について、$length(\alpha \cdot a)$ を $length(\alpha) + 1$ と定義すれば、この関数は一意にしか存在しないことが示せる。□

3 複合型の ISA 階層

以下ではデータベーススキーマにある意味順序 $\leq$ を仮定し、しかも DAG の形に現されているとする。複合型で幾つかの順序を定義し、型スキーマのコンパクトな表現を得ることを考える。複合型の場合も型整合性を論じることができることに注意したい: 各複合型 v に対して, $t \in \tau(v)$ および $t \leq s$ が成り立てば $s \in \tau(v)$ も成立する。

はじめに `ISA` をデータ構造値集合の包含関係を用いて論じる。`ISA` は集合の包含関係を用いて定義したが、同様の方法によって `ISA` 順序を素直に拡張してみよう。複合型 t_1, t_2 in $\mathcal{D}(\mathcal{C})$ に対して、$\mathcal{D}(\mathcal{C})$ 上の `ISA` 制約 (または汎化) とは t_1 `ISA` t_2 の形の式を言う。データベースにおいて t_1 `ISA` t_2 が成り立つとは $\Gamma(t_1) \subseteq \Gamma(t_2)$ を充たすときとする。複合型とその解釈を構成的に定義したため、包含関係に基づく `ISA` は順序となる。

基本型と同様に、複合型 $\mathcal{D}_U$ においても `ISA` を演繹できる。例えば t_1 `ISA` t_2 が成り立つとき、定義より $\Gamma(t_1) \subseteq \Gamma(t_2)$ だから $2^{\Gamma(t_1)} \subseteq 2^{\Gamma(t_2)}$ となり、`setof`(t_1) `ISA setof`(t_2) を得る。しかし、`differenceof`(t_1, t_0) `ISA differenceof`(t_2, t_0) は成り立つが `differenceof`(t_0, t_1) `ISA differenceof`(t_0, t_2) はそうでない。つまり、このような性質は t_1, t_2 ではなくて

構成子に基づくため、構成子の一般的な性質の分析が必要である。

setof や **recordof** 等の構成子は $\mathcal{D}_U^n$ から $\mathcal{D}_U$ への写像と見ることができる。構成子 $\eta : \mathcal{D}_U^n \rightarrow \mathcal{D}_U$ が単調であるとは、η が順序関係を保存する、つまり複合型 $t_1,...,t_n \in \mathcal{D}_U$, $s_1,...,s_n \in \mathcal{D}_U$ が $\Gamma(t_i) \subseteq \Gamma(s_i)$, $i = 1,...,n$ を充たすならば、$\Gamma(\eta(t_1,...,t_n)) \subseteq \Gamma(\eta(s_1,...,s_n))$ を充たすときを言う。

[THEOREM 1] **setof**, **recordof(n)**, **arrayof(n)**, **listof**, **treeof(n)**, **unionof**, **intersectionof** は単調である。

上で示したように、**differenceof** は単調ではない。

[THEOREM 2] 任意の $t \in \mathcal{D}_U$ に対して、t **ISA** t_F^* が成り立つ。また 構成子 **closureof**(F) は単調である。

[THEOREM 3] 構成子 $\eta : \mathcal{D}^n \rightarrow \mathcal{D}$, $\eta_i : \mathcal{D}^{k_i} \rightarrow \mathcal{D}$, $i = 1,..,n$ がすべて単調ならば、構造構成 $\eta(\eta_1,..,\eta_n)$ も単調である。

[THEOREM 4] 構成子 $\eta : \mathcal{D}_U^n \rightarrow \mathcal{D}_U$ が単調ならば、η は **ISA** を保存する。つまり、$t_1,...,t_n \in \mathcal{D}_U$, $s_1,...,s_n \in \mathcal{D}_U$ が t_i **ISA** s_i, $i = 1,...,n$ を充たすならば、$\eta(t_1,...,t_n)$ **ISA** $\eta(s_1,...,s_n)$ が成り立つ。

この性質は t_1 **ISA** t_2 から別の規則 **setof**(t_1) **ISA** **setof**(t_2) を導けることを意味する。これは更に、**setof(setof**(t_1)**)** **ISA** **setof(setof**(t_2)**)** をも意味するため、有限個の **ISA** 規則と基本型から無数の (複合型上の) 規則を生成できることになる。複合型に関する **ISA** の一般的な含意問題は有限時間で扱えない。

[EXAMPLE 5] 例 2 では **setof**, **listof** **recordof** が生じるが、これらの構成子はすべて単調であり、後述するように逆単調でもある。例 2 から **ISA** hierarchies を生成すると、図 4 になる。例 3 のスキーマはこの階層の型整合性を充たしている。□

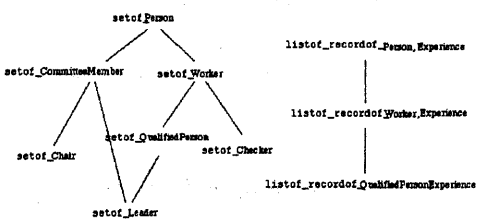

図 4: ISA Hierarchies in Project Activity Database

たとえ $\eta(t_1,...,t_n)$ **ISA** $\eta(s_1,...,s_n)$ が成り立っても、t_i **ISA** s_i は必ずしも成立しない。簡単な反例は $\eta(t) = t \cup t_0$ である: $\eta(t_0)$ **ISA** $\eta(t)$ だが t_0 **ISA** t は必ずしも成り立たない。. 構成子 η が逆単調であるとは、$\Gamma(\eta(t_1,...,t_n)) \subseteq \Gamma(\eta(s_1,...,s_n))$ ならば t_i, s_i, $i = 1,...,n$ に対して$\Gamma(t_i) \subseteq \Gamma(s_i)$ となるときをいう。

[THEOREM 5] **recordof(n)**, **setof**, **listof**, **arrayof(n)** は逆単調である。他方、**unionof**, **intersectionof** は逆単調ではない。

単調性と逆単調性は構成子の性質に依存している。このふたつが同時に成り立てば、複合型の **ISA** 階層は基本型階層と 1 対 1 に対応する。これらは型整合性に新たな制限を設ける事になる。η が単調でその解釈を ζ とするとき、$t_i \in \tau(v_i)$ かつ t_i **ISA** s_i, $i = 1,..,n$ ならば、$\tau(\zeta(v_1...v_n))$ は$\eta(t_1...t_n)$ および $\eta(s_1...s_n)$ を含む。同様に、η が逆単調で、$\tau(\zeta(v_1...v_n))$ が$\eta(t_1...t_n)$ および $\eta(s_1...s_n)$ を含み$\eta(t_1...t_n)$ **ISA** $\eta(s_1...s_n)$ ならば、$\tau(v_i)$ は t_i かつ s_i を含む $(i = 1,..,n)$。

4 複合型の間の対応

本来 **ISA** 概念は 2 つの値を同一視することを基にしているが、これを拡張して 2 つの複合値の間の単射対応を用いることを考える。一般に複合型 t_1 が t_2 に対応する、または t_1 **ISC** t_2 とは、(全域的) 単射関数 $f : \Gamma(t_1) \rightarrow \Gamma(t_2)$ が存在するときをいう。定義より t_1 **ISA** t_2 ならばt_1 **ISC** t_2 である。

数学的な観点から見れば、t_1 **ISC** t_2 の意味することは $|\ \Gamma(t_1)\ | \leq |\ \Gamma(t_2)\ |$ であるが、**ISC** の定義では計算可能 (*computable*) な単射 f を必要とする。つまり、t_1 **ISC** t_2 ならば $|\ \Gamma(t_1)\ | \leq |\ \Gamma(t_2)\ |$ を満たし、かつその対応を実際に f を用いて操作できることが必要である。したがって、単射関数の性質を論じる必要が有り、実際の **ISC** の意味は構造構成子に応じて異なる。

ISC は決して包含関係による順序とはならない。実際、t_1 が t_2 に準等価 (*pseudo-equal*) $t_1 \cong t_2$ とは、$\Gamma(t_1)$ と$\Gamma(t_2)$ の間の双射を言うが、これによって **ISC** は順序となる。

[EXAMPLE 6] どのリーダもただ一つの **OperatorGroup** を管理するとしよう。このことから、**Leader** から **OperatorGroup** への単射が存在する。つまり、**Leader ISC OperatorGroup** となる。□

構造構成子 $\eta : \mathcal{D}_U^n \rightarrow \mathcal{D}_U$ が単射であるとは、η が **ISC** を保存するときをいう。つまり、t_i **ISC** s_i が $i = 1,..,n$ について成り立つならば、$\eta(t_1...t_n)$ **ISC** $\eta(s_1...s_n)$ となる。逆単調性と同様に、逆単射性を定義できる。**ISA** と同様、単射性や逆単射性は型整合性にあらたな制約を加えることになる。

[THEOREM 6] $\eta : \mathcal{D}^n \rightarrow \mathcal{D}$ と$\eta_i : \mathcal{D}^{k_i} \rightarrow \mathcal{D}$ が与えられ、しかも η, η_i がすべて単射とする、ただし $i = 1,..,n$ とする。このとき結合構造 $\eta(\eta_1,..,\eta_n)$ も単射となる。

[THEOREM 7] $t_1, t_2,.., s_1, s_2,..$ を $\mathcal{D}(\mathcal{C})$ の複合型とする。
(1) t_1 **ISC** t_2 かつ t_2 **ISC** t_3 ならば、t_1 **ISC** t_3 。
(2) **recordof** は単射かつ逆単射な構成子である。
(3) **setof** は単射かつ逆単射な構成子である。
(4) **listof** は単射かつ逆単射な構成子である。
(5) **arrayof(n)** は単射かつ逆単射な構成子である。
(6) **treeof(n)** は単射かつ逆単射な構成子である。

残念ながら他の構成子に関しては大部分が単射にならない。例えば **unionof** の場合、t_i **ISC** s_i (単射関数を f_i とする、$i = 1,2$) であっても、**unionof**(t_1,t_2) **ISC** **unionof**(s_1,s_2) が常に成り立つとは言えない、というのは $|\Gamma(\textbf{unionof}(t_1,t_2))| \leq |\Gamma(\textbf{unionof}(s_1,s_2))|$ が成り立たないから。逆単調性についても同様である。

[THEOREM 8] **closureof**(F) を F 上の自由生成演繹集合の構成子とする。F のすべての生成規則がとその解釈が単射ならば、**closureof**(F) も単射となる。

5 埋め込み可能な複合型

単純な **ISA** 拡張は構成子の間の関連性を考慮していないため、重要な性質を見失うことがある。例えば、ふたつの複合型 **recordof(2)**(t_1,t_2) と **recordof(3)**(t_1,t_2,t_3) において、構成子が異なるため両者の間に **ISA** 関連は成立し得ない。しかし、$\Gamma(t_3)$ に特別な値 $\perp$ を導入し複合値 (v_1,v_2) を $(v_1,v_2,\perp)$ と見なせば、$(v_1,v_2) \in \Gamma($**recordof(2)**$(t_1,t_2))$ を$\Gamma($**recordof(3)**$(t_1,t_2,t_3))$ の要素と対応付けることがで

きる。前者の値は後者の一部として生じているため、recordof(2)(t_1,t_2) を recordof(3)(t_1,t_2,t_3) に埋め込んだと考えて良い。このように構成子を変換する技法を埋め込みという。

$\Gamma(\eta_1(t_1..t_n))$ から $\Gamma(\eta_2(t_1..t_n s_{n+1}..s_m))$ への関数 α が η_1 埋め込みであるとは、α が全域的単射で、各 t_i 値 v_i が η_1 表現 $v(v_1..v_n)$ に生じているならば $\alpha(v)$ にも表れているときとをいう。型 $\eta_1(t_1...t_n)$ が別の型 $\eta_2(t_1...t_n s_{n+1}...s_m)$ に η_1 埋め込み可能 (これを $\eta_1(t_1...t_n) \hookrightarrow \eta_2(t_1...t_n s_{n+1}...s_m)$ と表す) とは、$\Gamma(\eta_1(t_1...t_n))$ から$\Gamma(\eta_2(t_1...t_n s_{n+1}...s_m))$ へのη_1 埋め込みが存在するときを言う。例えば、recordof(2)(t_1,t_2) $\hookrightarrow$ recordof(3)(t_1,t_2,t_3) である。直観的に、型 t が他の型 s に η 埋め込み可能であるとき、単射的に η 部に書き換え s 上の値を得ることができる。そのあと $v \in \Gamma(t)$ と $\alpha(v) \in \Gamma(s)$ を同一視し、$\Gamma(t)$ を構成する各値は (αによって) $\Gamma(s)$ の値となる。

η_1 、η_2 を意数 n,m の構造構成とする ($n \leq m$)。η_1 が η_2 に埋め込み可能 ($\eta_1 \hookrightarrow \eta_2$ と表す) とは、$\eta_1(t_1...t_n)$ が別の型 $\eta_2(t_1...t_n s_{n+1}...s_m)$ に η_1 埋め込み可能のときとする。ここで、$t_1,..,t_n$ は任意の型であり、特別の値が $s_{n+1},..,s_m$ で用意されているとする。

2 つの型 $t=\eta_1(t_1...t_n)$、$s=\eta_2(t_1...t_n s_{n+1}...s_m)$ に対して、$t \hookrightarrow s$ は単射による意味を用いた特殊な種類の ISA に見える。なぜなら、$\alpha(\Gamma(t)) \subseteq \Gamma(s)$ が成り立ち、任意の t_i 値が対応する s 値に単射的に生じているからである。

[THEOREM 9] 構成子 $\eta_i, \xi_i \in C, i=1,..,k$ が$\eta_i \hookrightarrow \xi_i$ を満たすなら、任意の位数 k の構成 η に対して、$\eta(\eta_1..\eta_k) \hookrightarrow \eta(\xi_1..\xi_k)$ となる。

[THEOREM 10] (1) t ISA s ならば $t \hookrightarrow s$ (id による)
(2) $\eta_1 \hookrightarrow \eta_2$ かつ$\eta_2 \hookrightarrow \eta_3$ ならば、$\eta_1 \hookrightarrow \eta_3$.
(3) t_1 ISA t_2 かつ $t_2 \hookrightarrow t_3$ ならば、$t_1 \hookrightarrow t_3$ が成り立つ。
(4) (3) の性質は演繹閉包の場合も成り立つ。

この $\hookrightarrow$ は構成子上の順序と考えることができる。実際、$\eta_1 \hookrightarrow \eta_2$ かつ$\eta_2 \hookrightarrow \eta_1$ を準等号 (*pseudo-equality*, $\doteq$ と表す) とすれば良い。

[THEOREM 11] 構成子 id は setof, listof, recordof(n), arrayof(n), treeof(n), unionof, closureof のいずれかを XXX とすると $id \hookrightarrow XXX$ である。

[THEOREM 12] 次の性質が成り立つ。
(1) setof $\hookrightarrow$ listof.
(2) listof $\hookrightarrow$ treeof(1).
(3) recordof(n) $\hookrightarrow$ recordof(m), arrayof(n) $\hookrightarrow$ arrayof(m), treeof(n) $\hookrightarrow$ treeof(m) ただし $n \leq m$.
(4) recordof(n) $\hookrightarrow$ arrayof(n), arrayof(n) $\hookrightarrow$ recordof(n). 言い替えると、arrayof(n) $\doteq$ recordof(n).
(5) recordof(n) $\hookrightarrow$ treeof(n).
(6) arrayof(n) $\hookrightarrow$ listof.
(7) arrayof(n) $\hookrightarrow$ treeof(n).
(8) intersectionof $\hookrightarrow id$ and differenceof $\hookrightarrow id$, しかし逆が成り立たない。
(9) listof $\hookrightarrow$ closureof(F). arrayof(n) $\hookrightarrow$ closureof(F).

木の埋め込み (*tree embedding*) は重要な例である。$\alpha([e,\langle a_1..\rangle,...,\langle b_1...\rangle]) = [e,\langle a_1..\rangle,...,\langle b_1...\rangle,\langle\rangle,,,\langle\rangle]$ を埋め込みとすれば、treeof(n)$(t,t_1,...,t_n)$ $\hookrightarrow$ treeof(m)$(t,t_1,...,t_n,...,t_{n+m})$ を得る。

構成子を組み合わせただけでは得られない構造構成の間の $\hookrightarrow$ も定義できる。例えば setof (unionof (t_1,t_2)) と recordof(2) (setof(t_1),setof(t_2)) が挙げられる。実際、$\{v_1,v_2,..,v_n\}$ の要素のすべての t_1 値をまとめて $\{a_1,...,a_k\}$ とし、すべての t_2 値を $\{b_1,..,b_m\}$ とすれば、$\{v_1,v_2,..,v_n\}$ を $[\{a_1,...,a_k\},\{b_1,..,b_m\}]$ とみなすことができる。こう言った性質は t_1,t_2 には依存せず、むしろ setof(unionof) や recordof(2) (setof, setof) といった構造構成による。

[THEOREM 13] 次の性質が成り立つ。
(1) recordof(n+m) $\doteq$ recordof(2)(recordof(n),recordof(m)).
(2) setof(unionof) $\hookrightarrow$ recordof(2)(setof, setof)

[EXAMPLE 7] 例 5 において次が得られる:
setof(Worker) $\hookrightarrow$ listof(Worker)
listof(Worker) $\hookrightarrow$ listof(recordof(2)(Worker,Experience))
setof(CommitteeMember) $\hookrightarrow$ listof(Person)
listof(Person) $\hookrightarrow$ listof(recordof(2)(Person,Experience))
recordof(2)(QualifiedPerson,Experience) $\hookrightarrow$
listof(recordof(2)(QualifiedPerson,Experience))

ISA 階層と組み合わせる事によって、図 5 は新たな関連を含む。ただし、例 3 の型スキーマは無限集合になる。例えば、集合値 { Beethoven, Brahms } はこの拡張によって、setof(Worker), setof(setof(Worker)), ..., listof(Worker), listof(setof(Worker)), listof(listof(setof(Worker))), ..., listof(listof(Worker)), ..., listof (recordof(2) (Worker,Experience)), listof(listof (recordof(2) (Worker,Experience))), ..., listof (recordof(2) (Person,Experience)), listof (listof (recordof(2) (Person,Experience))), ... を含んでいる。□

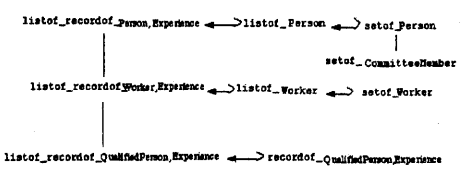

図 5: Embeddability in ISA Hierarchies

$\hookrightarrow$ は構造構成を書き換えるのに有用である。すべての構造構成の規則が機械的に得られるわけではない。このため、書き換え規則の形で明示的に指定されていると仮定する。

6 近汎型集合の生成

近汎複合型スキーマを得るため、これまで 3 種類の順序 ISA, ISC, $\hookrightarrow$ を導入した。しかし本質的な問題を残したままになっている、つまり複合型スキーマが無限集合となる状況の回避と記述の簡単化手法である。ここでは順序を (主とし ISA と考えて) $\leq$ と表現する。

以下では次のような性質を充たす構成子集合を考える:
(1) 埋め込みの検査 ($\eta_1 \hookrightarrow \eta_2$) は有限時間で停止する。
(2) 自明な構成子 id は他のすべての構成子に埋め込み可能である。
(3) 構造構成 η_1 で自明な表現を解釈とするならば、これとは異なる構成 η_2 で$\eta_2 \hookrightarrow \eta_1$ を充たすものが存在する。

最初の仮定の意味することは、異なる複合型の統合化のために構造構成を変更できることにある。第 2 の仮定は、どの値も適当に補う事によって他の型の値と見なすことができ、構造構成が複雑になるほどその解釈が広がることを意味する。これより、例えば intersectionof は除外される。最後の仮定の例は unionof である。これは自明な表現を解釈とし、$t \hookrightarrow \texttt{unionof}(t,s)$ である。一方 intersectionof はそうではない。

6.1 多相的な空値

集合型 setof(t) は、解釈の定義から $\Gamma(\texttt{setof}(t))$ に空集合 ($empty$) ϕ を要素として含む。どのような型 t についても成り立つから、ϕ はすべての集合型を含み、$\tau(\phi)$ は無数を要素を持つ。ここで Φ を解釈が ϕ だけをもつような型とする。定義より、任意の集合型 s に対して Φ ISA s である。しかも $\Gamma(\Phi)$ は集合値を含むから、Φ は setof(t_0) の形であらわされるに違いない、ここで t_0 は要素を表す型で $LG(\phi) = \{\Phi\}$ のはずである。既に見たように setof は逆単調であるから、任意の型 t に対して t_0 ISA t が成り立つので、t_0 は (基本型も含めて)'最小の型' である (このため混乱がない限り $\perp$ と表す)。なお ϕ は $\{\phi\}$ と同一ではない。同様の考察が listof に付いても成り立ち、特別なリスト ρ を考えることができる。

treeof(n) の場合には、空木 $[\perp, \rho, .., \rho]$ を考えることがある。ここで、$\perp$ は値がないこと、つまり $\perp \in \Gamma(t_0)$ を示す。他の場合も同様の特別な値を考えることができる。$\perp$ 値を空 ($null$) と呼ぶが、空値はすべての型を持つ。ϕ は $\{\perp\}$ 型を持ち、ρ は $\langle\perp\rangle$ 型を持つ。

一般に、構成子 $\eta : \mathcal{D}_U^n \rightarrow \mathcal{D}_U$ に対して、すべての $\eta(t_1...t_n)$ の解釈は共通して一意な η 空値を含む。η 空値は構成子によって決まるもので、型 t_1,.. に依存しない。例えば、ϕ は setof 空値であり、ρ は listof 空値である。$\mathcal{C}$ は有限であるから、構成子に依存した空値は有限個しかない。但し、構造構成は無数にあり、これに応じて空値も無数個存在する。例えば、$\{\{\}\}$ は setof(setof) 構造構成による空値である。

$\hookrightarrow$ 機構を用いれば、このような空値は同一になる。このとき、setof 空値は $\perp$ となる。従って $\perp = \{\perp\} = \{\{\perp\}\} = ...$ であるから、setof 空値は消滅する。一般に、$id \hookrightarrow con$ (ここで con は任意の構成子) であるから、どの構成子依存の空値も $\perp$ と同一になる。

6.2 近汎型集合

ここでは近汎型集合の有限性について論じる。このためには$\hookrightarrow$ を用いる必要がある。$\mathcal{T}$, $\mathcal{C}$, $\mathcal{E}$ や $\hookrightarrow$ 規則は有限であり、格納されている複合値も有限である。複合値 v とその型スキーマ $\tau(v)$ に対して、順序 $\leq$ に関する近汎集合 $LG(v)$ を次のように作る: 型 t_1, t_2 が $t_1 \in LG(v)$, $t_1 \leq t_2$ または $t_1 \hookrightarrow t_2$ で、しかも $t_1 \in LG(v)$ ならば、$t_2 \notin LG(v)$ である。型整合性より $\tau(v)$ と $LG(v)$ は同じだけの意味を有するが、(理論上) $\tau(v)$ は無限になることがある。もし $LG(v)$ が無限ならば我々は計算できず簡単化も行えない。空値問題を考えた理由はここにあった。

以下では $LG(v)$ が有限であることを示そう。v が ϕ の場合は前節の議論から $LG(v)$ は有限である。v が空値ではないとする。$v \in \Gamma(t)$ で $t_i \in \mathcal{T}$ $(i = 1, .., n)$、$t = \eta(t_1...t_n)$ を $LG(v)$ に含まれる複合型とする。v は一意な ζ 表現を持ち、これを $v = \zeta(v_1, .., v_m)$ とする。ここで、v_i は実体 (基本値) または実体の集まりを表す。T_i を v_i が持つすべての基本型の集合とすると、$\mathcal{T}$ が有限であるため、T_i は有限である。

表現が自明 (id) であるような η があれば、t に埋め込める型が存在するため矛盾。つまり、η は構成子並び η' に解釈され、その表現が ζ となる。ζ には id は含まれない。つまり、(T_i の有限性から) $T_1, .., T_n$ からの η による構造構成は有限である。

$\mathcal{C}$ および $\mathcal{C}'$ は有限であるから、有限個の構成子が ζ のデータ構造操作に解釈される。即ち、$LG(v)$ は有限である。

[EXAMPLE 8] 例 3 においても、ISA と埋め込み型の性質から、一般的には型スキーマは無限集合である。しかし上述の有限性から、同じ情報を表すのに次のような有限 LG 集合だけを保持すれば良い。

(格納) 複合値	LG 集合
{ Beethoven,Brahms }, { Puccini, Verdi }	setof(Leader)
{ Sibelius, Brahms }	setof(CommitteeMember)
{ Wagner }	setof(CommitteeMember), setof(QualifiedPerson), setof(Checker)
{ Chopin, Liszt }, { Schumann,Schubert }	setof(Worker)
{ Brahms,Bruckner }	setof(CommitteeMember), setof(QualifiedPerson)

各 LG 集合 T に対して、この要素を型にもつ値が必ず存在する。例えば、LG 集合 setof(Worker) は { Sibelius, Brahms } 以外の複合値が型に持つ。□

7 結論

本研究では複合オブジェクトの型スキーマのための理論的基礎を与え、関連する概念や一貫性制約のモデリングを拡張した。具体的には複合型階層を構成して順序を導入し、ISA, ISC および埋めこみを論じた。

本報告では ISA と埋め込み可能性を組み合わせて第 2 の ISA を考えた。つまり、$t_1 \hookrightarrow t_2$ は t_1 ISA t_2 よりも弱い結びつきを考えた。しかし、この順序を替えても問題は生じない。この問題は、データ構造の役割を決定する利用者の判断を必要とするため、別の考察が必要である。

参考文献

[1] Elmasri, R. and Navathe S.B.: Fundamentals of Database Systems, *Benjamin* (1994)

[2] Fikes, R. and Kehler, T.: The Role of Frame-Based Representation in Reasoning, *CACM* 28-9 (1985), 904-920

[3] Hull, R. and King, R.: Semantic Data Modeling, *ACM Comp. Surveys* 19-3 (1986)

[4] Lloyd, W.: Foundations of Logic Programming (2nd Eds), *Springer-verlag* (1987)

[5] Miura, T., Arisawa, H.: Logic Approach of Data Models - *Data Logic, Future Database Systems*, 1990, 228-259

[6] Miura, T.: A Logical Framework for Deductive Objects, *Information Systems* 17-5 (1992), 395-414

[7] Miura,T: Database Paradigms Towards Model Building, *Object Roll Modelling*, 1994, 228-258

[8] 三浦、塩谷: データベースにおける型スキーマの発見、情報処理学会論文誌 38-6 (1997)

[9] Miura, T. and Shioya, I.: Examining Complex Objects for Type Scheme Discovery, to appear in proc.*DEXA Conf.* , 1997

[10] Miura, T. and Shioya, I.: On Complex Type Hierarchy, to appear