ICOT Technical Memorandum: TM-1272

TM-1272

演繹オブジェクト指向データベースシステム Quixote の特徴と実現

高橋　千恵（JIPDEC）、森田　幸伯（沖）、
西岡　利博（MRI）、津田　宏

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5

Institute for New Generation Computer Technology

演繹オブジェクト指向データベースシステム *Quixote* の特徴と実現

高橋千恵
(財) 日本情報処理開発協会

森田 幸伯
沖電気工業 (株)

西岡利博
(株) 三菱総合研究所

津田 宏
(財) 新世代コンピュータ技術開発機構

自然言語処理や、法的推論、遺伝子情報処理などを扱う知識情報処理では、高い表現能力と豊富な問い合わせ能力が要求される。このような、知識情報処理の応用の要求に答えるために、ICOT では、演繹オブジェクト指向データベース (DOOD) 言語として、また、制約表現が可能な制約論理型言語として、*Quixote* を設計し、試作システムを開発してきた。本稿では、*Quixote* システムの機能のうち、属性継承、モジュール、拡張された問い合わせの 3 点を中心にその特徴と実現方式について述べる。

Features and Implementation of a Deductive Object-Oricnted Database System *Quixote*

Chie Takahashi
Japan Information Processing
Development Center(JIPDEC)

Yukihiro Morita
Oki Electric Industry Co. Ltd.

Toshihiro Nishioka
Mitsubishi Research Institute, Inc.

Hiroshi Tsuda
Institute for New Generation Computer
Technology (ICOT)

Knowledge information processing systems that treat applications such as natural language processing, legal reasoning, and genetic information processing, require many features for knowledge-bases and knowledge representation. In order to support such applications, *Quixote* was designed as a Deductive Object Oriented Database (DOOD) language and as a constraint logic programming language and its experimental system has been developed at ICOT. This paper describes some features and their implementations. We focus on the property inheritance, modules, and extended queries of *Quixote*.

1 はじめに

電子化辞書データ、分子生物学データ、法的判例データなどを扱う知識情報処理では、高い表現能力と豊富な問い合わせ能力が要求される。そこで、知識情報処理で重要な役割を果たす知識表現言語として、論理的基盤、強力な推論能力と柔軟なモデル化能力を持つ演繹オブジェクト指向データベース (Deductive Object-Oriented Database:DOOD) 言語が注目されている。

又、知識情報処理で扱うデータは、今までのデータベースが扱っていたものとは異なる性質を持つことが知られている。例えば、遺伝子情報は、まだほんの一部が解明されたにすぎず、たくさんのノイズを伴った知識とデータが部分的に集められ始めた段階である。しかも、以前のものとは一見矛盾したデータが集められたり、重複があったり、確定値ではなく制約のみしか分からないデータもある。法的推論では、同じ判例データから、弁護士や裁判官など立場の違いによって、異なる判決を推論するのが普通である。そこでは、単純な演繹ではなく、仮説推論 / 生成、帰納推論、類推などの高度な推論が実験的に行なえる環境が求められている。

このような状況をふまえて、ICOT では、DOOD 言語として、また制約表現を可能にした制約論理プログラム (Constraint Logic Programming:CLP) 言語として知識表現言語 *QUIXOTE* の設計、システムの試作を 1990 年から開始した。*QUIXOTE* システムは、ICOT で開発された並列推論マシン PIM 上の並列論理プログラム KL1 で実装された。このシステムは、ICOT Free Software として 1992 年 9 月に 2 版が、1993 年 6 月に 3 版がリリースされた。

知識表現言語 *QUIXOTE* は多くの特徴を持っている [4, 5, 7, 8]。ここでは、以下の 3 点に焦点をあて、その特長的な機能や、実現方式などについて言及する。

- 属性継承
- モジュール
- 拡張された問い合わせ

本稿では、2章で DOOD 言語 *QUIXOTE* の基本要素を述べ、3章で DOOD 言語 *QUIXOTE* の特徴とその実装を示し、4章で DOOD システム *QUIXOTE* の全体構成を明らかにし、5章で、まとめと今後の研究について述べる。

2 DOOD 言語 *QUIXOTE* の基本要素

言語 *QUIXOTE* の基本となる項は、オブジェクト項と呼ばれる拡張項で、

- 基本オブジェクト項
- 複合オブジェクト項
- 変数

からなる。*QUIXOTE* では、このオブジェクト項をオブジェクト識別子 (oid) として用いている。

基本オブジェクト項は、apple, plant, dog, japan などのアトムである。

複合オブジェクト項は、

$$o[l_1 = v_1, l_2 = v_2, \cdots]$$
$o, \forall i\ l_i$ は 基本オブジェクト項,
v_i は オブジェクト項

という形の項で表現される。各ラベル l_i は、固有属性と呼ばれ、その値とともに、その複合オブジェクトを識別する、本質的な性質を示す属性である。例えば、以下が青りんごに相当する、複合オブジェクト項である。

$$apple[color = green]$$

QUIXOTE では、ユーザが基本オブジェクト項間の包摂関係 $\sqsubseteq$ をプログラムで定義することができる。$a \sqsubseteq b$ は、「a は b の下位概念である」または「a は b である」ことを意味する。ユーザの定義した包摂関係は、半順序であるが、極大値 $\top$、極小値 $\bot$ を加えた (基本オブジェクト項, $\sqsubseteq, \top, \bot$) から、完備束が構成できることが知られている [1]。又、基本オブジェクト項の包摂関係は、基底複合オブジェクト項間に自然に拡張され、完備束も構成される [4]。

QUIXOTE では、ルールの集合をモジュールと呼び、知識の分類を可能にするモジュール概念が導入されている。各モジュールは、モジュール識別子と呼ばれるオブジェクト項により識別される。又、ユーザは二つのモジュールのモジュール間関係 $\sqsupseteq_S$ をプログラムで定義することができる。モジュール間関係はルールの継承の方向を規定する。例えば、

$$m_1 \sqsupseteq_S m_2$$
$$m_3 \sqsupseteq_S m_4 \cup (m_5 \backslash m_6)$$

の場合、原則としてモジュール m_2 内の全てのルールは、モジュール m_1 に継承され、$m_4 \cup (m_5 \backslash m_6)$ のような集合演算によって定義されるルールの集合がモジュール m_3 に継承される。共通集合や差のような集合演算は構文的に評価される。又、モジュール識別子の無いルールは、全てのモジュールに継承される。

Quixote には、オブジェクト項の他に、オブジェクト項に属性の指定を加えた記述方法として、属性項と呼ばれるものがある。

$o/[l_1 op_1 v_1, l_2 op_2 v_2, \cdots]$
$o, \forall i\ l_i, v_i$ は オブジェクト項,
$op_i \in \{=, \leftarrow, \rightarrow\}$

という形をしている。各ラベル l_i は、付帯属性と呼ばれる。

Quixote 上では、属性項の記法は一種のマクロであり、以下の規則によりオブジェクト項と制約の集合に変換される。(属性項のオブジェクト項と制約の集合に変換される性質)

$$o/[l = X] \Leftrightarrow o|\{o.l \cong X\}$$
$$o/[l \rightarrow X] \Leftrightarrow o|\{o.l \sqsubseteq X\}$$
$$o/[l \leftarrow X] \Leftrightarrow o|\{o.l \sqsupseteq X\}$$

$o|C$はオブジェクト項oとその制約Cを表す。
$o.l$の形の項は、ドット項と呼ばれ、
オブジェクト項oの属性lの値を示す。

上記のことから、属性項は、以下のような分解、合成の性質を持つ。(属性項の合成の性質)

$$o/[l_1 = a, l_2 = b] \Longleftrightarrow o/[l_1 = a], o/[l_2 = b]$$

このような性質は、部分情報の処理に効果的である。この時、属性のラベルは(モジュールと)オブジェクトを与えると、属性値を返す関数と解釈しているため、(同一モジュール内では)属性値はユニークでなければならない。

Quixote の制約は、オブジェクト項、或は、ドット項間の包摂関係による制約式の集合である。各制約式は、

$$< term_1 >< op >< term_2 >$$
$< term_1 >, < term_2 >$ はオブジェクト項、或はドット項,
$$< op > \in \{\cong, \sqsubseteq, \sqsupseteq\}$$

という形をしている。

Quixote のルールは、以下の形の節で構成される。

$$\overbrace{m_0 :: H}^{\text{頭部}}\ \overbrace{|HC}^{\text{頭部制約}}\ \Leftarrow \overbrace{m_1 : B_1, \ldots, m_n : B_n}^{\text{本体}}\ \overbrace{||BC}^{\text{本体制約}}\ ;;$$

H, B_iはオブジェクト項または属性項、
HC, BCは制約、m_iはモジュールを示す。

但し、属性項は、オブジェクト項と制約の集合に変換できるので、H, B_i は、オブジェクト項と仮定できる。なお、本体、制約、m_i は省略可能である。

上のルールは、直観的には、

> 「もし各 B_i がモジュール m_i において制約 BC の元で成り立つならば、H と制約 HC がモジュール m_0 において成り立つ」

という m_0 における知識を意味する。

上記の基本要素を使って、*Quixote* のプログラム(データベース)は、

- S: 包摂関係 $\sqsupseteq$ の定義
- M: モジュール間関係 $\sqsupseteq_S$ の定義
- R: ルールの定義

の3つ組 (S, M, R) と表せる。

3 DOOD 言語 *Quixote* の特徴と実現

3.1 属性継承

3.1.1 基本概念

Quixote には、拡張項に基づいたオブジェクト識別子(oid)と属性の概念が導入されており、oid 間に定義されている包摂関係 $\sqsubseteq$ に従って属性継承が行なわれる。

2章で示したように、属性項はオブジェクト項と制約の集合に変換される。*Quixote* の属性継承の一般規則は、以下のように制約により表現されている。

$$o \sqsubseteq p \Rightarrow \forall\, l\ \ o.l \sqsubseteq p.l$$

従って、*Quixote* では、属性継承は制約の継承と考えてよく、多重継承は複数の制約の和として定義できる。

例えば、$(a \downarrow b)$, $(a \uparrow b)$ をそれぞれ包摂関係に基づく束上の meet 演算、join 演算とすると、

$o_1 \sqsubseteq o_2, o_1 \sqsubseteq o_3, o_2/[l \rightarrow a], o_3/[l \rightarrow b]$
$\Longleftrightarrow \quad o_1 \sqsubseteq o_2, o_1 \sqsubseteq o_3, o_2|\{o_2.l \sqsubseteq a\}, o_3|\{o_3.l \sqsubseteq b\}$

$$
\begin{aligned}
&\Longrightarrow \quad o_1|\{o_1.l \sqsubseteq a\}, o_1|\{o_1.l \sqsubseteq b\} \\
&\Longleftrightarrow \quad o_1|\{o_1.l \sqsubseteq a, o_1.l \sqsubseteq b\} \\
&\Longleftrightarrow \quad o_1/[l \rightarrow (a \downarrow b)]
\end{aligned}
$$

このように、通常の継承では、上位概念 (o_2, o_3) の属性値 (の制約) が、下位概念 (o_1) の属性値の制約として継承される。また $QUIXOTE$ では、次のように継承の規則により下位概念 (o_4, o_5) の属性値 (の制約) により、上位概念 (o_1) の属性値が継承を受ける場合もある。

$$
\begin{aligned}
&o_1 \sqsupseteq o_4, o_1 \sqsupseteq o_5, o_4/[l \leftarrow a], o_5/[l \leftarrow b] \\
&\Longleftrightarrow \quad o_1 \sqsupseteq o_4, o_1 \sqsupseteq o_5, o_4|\{o_4.l \sqsupseteq a\}, o_5|\{o_5.l \sqsupseteq b\} \\
&\Longrightarrow \quad o_1|\{o_1.l \sqsupseteq a\}, o_1|\{o_1.l \sqsupseteq b\} \\
&\Longleftrightarrow \quad o_1|\{o_1.l \sqsupseteq a, o_1.l \sqsupseteq b\} \\
&\Longleftrightarrow \quad o_1/[l \leftarrow (a \uparrow b)]
\end{aligned}
$$

但し、固有属性は、その意味付けから明らかなように、継承の範囲外である。つまり、固有属性により属性継承の例外を表現することができる。

例えば、$apple.color \cong red$ の場合、$apple \sqsupseteq mcintosh$ ならば、$color$ 属性が $mcintosh$ に継承されて、$red \sqsupseteq mcintosh.color$ となる。しかし、$apple[color = green] \sqsubseteq apple$ は常に成り立つが、$color$ は固有属性でもあるので、$apple[color = green].color \cong green$ となり、属性間には包摂関係は成り立たない。

oid の概念を導入したことによって、$QUIXOTE$ のルール評価は、Prolog 等の論理プログラムとは違う処理を行なわなくてはならないことを以下に説明する。

ルールのうち本体が空のものは、ファクトと呼ばれ、$QUIXOTE$ では属性項により表現される。同じオブジェクト項を持つ２つの属性項は、属性項の合成の性質より、一つの属性項にまとめられ、まとめられた制約が解消され、属性項のマージが起こる。

例えば

$$
\begin{aligned}
&apple/[taste \rightarrow sour] \wedge \\
&fruit/[taste \rightarrow sweet] \wedge apple \sqsubseteq fruit \\
&\Longrightarrow apple/[taste \rightarrow sour \downarrow sweet]
\end{aligned}
$$

である。

このことは、問い合わせ評価中に、本体と本体制約が満たされたルールの頭部のオブジェクト項についても当てはまる。例えば、

$$
\begin{aligned}
&o/[l \rightarrow a] \Leftarrow q || \{q \sqsubseteq b\}; ; \\
&p/[l \rightarrow c] \Leftarrow q || \{q \sqsubseteq d\}; ; \\
&q; ; \\
&where\ p \sqsubseteq o \wedge q \sqsubseteq b \wedge q \sqsubseteq d
\end{aligned}
$$

の時、p の属性は、$p/[l \rightarrow (a \downarrow c)]$ となる。

3.1.2 実現

$QUIXOTE$ では、属性継承を問い合わせ評価中に処理している。属性継承を問い合わせ処理の前処理で行なう選択もあるが、包摂関係は一般に数段ではないと考えられ、ルールに書かれているオブジェクト項も含めて、全てのオブジェクト項に属性継承を飽和させることは、メモリー上無理があると筆者らは判断した。

$QUIXOTE$ では、抽象的には、以下のようなルール変換を行なって、属性継承を実現している。

サブゴール p があり、ルール本体の制約に p をそのオブジェクト項とするドット項 $p.l_1, \cdots, p.l_k$ がある場合、包摂関係 $\sqsupseteq$ で、p の

- $\{o_1, \cdots, o_m\}$: 上位関係にあるオブジェクト項の集合
- $\{q_1, \cdots, q_n\}$: 下位関係にあるオブジェクト項の集合

を求め、

1) p を頭部のオブジェクト項とするルール

2) $o_1, \cdots, o_m, q_1, \cdots, q_n$ を頭部のオブジェクト項とするルール

を変換する。

1) の変換は、$o_1, \cdots, o_m, q_1, \cdots, q_n$ を特殊なサブゴールとして付加する。特殊なサブゴールの処理は、

> 特殊なサブゴールが失敗しても、そのサブゴールを無かったように扱い、ルール評価の処理は続行する。

とする。

2) の変換は、

- o_i を頭部のオブジェクト項とするルールには、$p.l_j \sqsubseteq o_i.l_j$ を頭部制約に加える。但し、$1 \leq j \leq k$
- q_i を頭部のオブジェクト項とするルールには、$q_i.l_j \sqsubseteq p.l_j$ を頭部制約に加える。但し、$1 \leq j \leq k$

である。例えば、

$$o/[l \rightarrow a]::$$
$$p::$$
$$q/[l \leftarrow b]::$$
$$where\ q \sqsubseteq p, p \sqsubseteq o, b \sqsubseteq a$$

というプログラムに対して、問い合わせ $?-p/[l=X]$ が与えられたとする。サブゴール p には $p.l$ があるので、包摂関係 $\sqsupseteq$ において、p の上位のオブジェクト項 o、下位のオブジェクト項 q を使って、変換すると、1) の変換は、

$$p \Leftarrow o.q$$

2) の変換は、

$$o/[l \rightarrow a]|\{p.l \sqsubseteq o.l\}$$
$$q/[l \leftarrow b]|\{q.l \sqsubseteq p.l\}$$

となる。上記の 3 つのルールを評価すると、

$$\{b \sqsubseteq q.l, q.l \sqsubseteq p.l, p.l \sqsubseteq o.l, o.l \sqsubseteq a\}$$

が得られ、制約解消を行なうと、問い合わせに含まれていた変数 $X = p.l$ の制約は、$\{b \sqsubseteq X, X \sqsubseteq a\}$ となる。

Quixote では、実際の 2) の変換は、ルール頭部のオブジェクト項の構造を抽象化したテンプレートを用いたある種の索引構造 (DAG 構造) を作成し、継承処理に必要なルールを高速に検索できるようにしている。索引作成処理は、問い合わせの前処理として行なっている。

3.2 モジュール

3.2.1 基本概念

Quixote は、知識を分類し、(局所的) 矛盾を取り扱うためにモジュール概念が導入されている。

Quixote のプログラムはモジュールという単位で分割できる。各モジュールは、ルールの集合からなり、モジュール識別子と呼ばれるオブジェクト項により識別される。

Quixote では、ルールの継承の例外も扱うことができる。各ルールに、継承モードとして o、l、ol を付加することができる。o(*override*) の付いたルールは、上位のモジュールから継承してきた同一頭部オブジェクト項を持つルールを上書きする。l(*local*) の付いたルールは、下位のモジュールに継承されない。モード ol は、o と l の組合せである。

$$england \sqsupseteq_S europe, france \sqsupseteq_S europe, \cdots$$
$$europe :: car/[pass = right];;$$
$$england :: (o)car/[pass = left];;$$

は、「ヨーロッパでは一般に車は右側通行、但し、イギリスでは、左側通行。」という知識を表している。

2 章で説明したように、属性のラベルはモジュールとオブジェクトを与えると、属性値を返す関数と解釈される。

つまり、一つのモジュール内では同一のオブジェクト項は同一の属性値を持たなければならないが、異なるモジュールでは、同一のオブジェクト項でも異なった属性値を持つことができる。

このモジュールの機能によって、大規模知識ベースにおける知識の分類が可能となり、矛盾をも含んでいるデータを *Quixote* システムの中に格納することができる。

3.2.2 実現

Quixote では、ルールの継承を問い合わせ評価の前に処理している。基本オブジェクト項、複合オブジェクト項から構成される完備束に比べ、一般に、モジュール間関係は少なく、ルールの継承を飽和させることが可能だと判断したからである。

抽象的には、以下のような手順で、ルールの継承を実現している。

1) モジュール間関係で定義されている共通集合や差の集合演算を構文的に評価する。

2) モジュール間関係とルールの継承の例外情報に従って、ルールをコピーし、ルールの継承関係を飽和させる。

3) ファクトを整理する。

Quixote では、手順 1) は、構文としてルールに識別子を振り、その識別子を使って、共通集合や差の集合演算を評価する。又、この時作られるルールのデータにおいて、ルールの本体のサブゴールで、モジュール識別子がないものは、変数 $Self$ をモジュール識別子とする。変数 $Self$ は、ユーザがルール中に書いたどの変数とも違うものとする。

手順 2) では、あるルール r が、m_1 から m_2 に継承されるとすると、r をコピーし、ルール頭部のモジュール識別子を m_1 から、m_2 に変更し、$Self \cong m_2$ という制約をルールの本体制約に追加する。

手順 3) では、3.1.1 で説明したように、属性項のマージを行なう。

例えば、

$$
\begin{aligned}
&m_1 :: \ (l)h_1 \Leftarrow b_1||C_1;;\\
&m_1 :: \ o/[l \rightarrow a];;\\
&m_2 :: \ h_2 \Leftarrow b_2||C_2;;\\
&m_2 :: \ o/[l \rightarrow b];;\\
&m_3 :: \ h_3 \Leftarrow b_3||C_3;;\\
&\{m_3, m_4\} :: h_4 \Leftarrow b_4||C_4;;\\
&m_5 :: \ (o)h_2 \Leftarrow b_5||C_5;;\\
&m_6 :: \ h_6 \Leftarrow b_6||C_6;;
\end{aligned}
$$

但し、$m_5 \sqsupseteq_S m_1 \cup m_2$, $m_6 \sqsupseteq_S m_3 \backslash m_4$ とする。手順 1) は、以下のようにルール識別子 rID を振り、

rID	ルール	
r_1	$m_1 ::$	$(l)h_1 \Leftarrow Self : b_1 \| C_1;;$
f_1	$m_1 ::$	$o/[l \rightarrow a];;$
r_2	$m_2 ::$	$h_2 \Leftarrow Self : b_2 \| C_2;;$
f_2	$m_2 ::$	$o/[l \rightarrow b];;$
r_3	$m_3 ::$	$h_3 \Leftarrow Self : b_3 \| C_3;;$
r_4	$\{m_3, m_4\} ::$	$h_4 \Leftarrow Self : b_4 \| C_4;;$
r_5	$m_5 ::$	$(o)h_2 \Leftarrow Self : b_5 \| C_5;;$
r_6	$m_6 ::$	$h_6 \Leftarrow Self : b_6 \| C_6;;$

とし、$m_1 \cup m_2$ と $m_3 \backslash m_4$ を構文的に評価すると、それぞれ、$\{r_1, r_2, f_1, f_2\}$ と $\{r_3\}$ となる。手順 2) で、$m_5 \sqsupseteq_S m_1 \cup m_2$, $m_6 \sqsupseteq_S m_3 \backslash m_4$ とモード o,l に従って、ルール継承関係の飽和を行なうと、m_5 と m_6 は、

mID	rID	rule	
m_5	r_{51}	$m_5 ::$	$h_2 \Leftarrow Self : b_5 \| C_5 \cap \{Self \cong m_5\};;$
	f_{51}	$m_5 ::$	$o/[l \rightarrow a];;$
	f_{52}	$m_5 ::$	$o/[l \rightarrow b];;$
m_6	r_{61}	$m_6 ::$	$h_6 \Leftarrow Self : b_6 \| C_6 \cap \{Self \cong m_6\};;$
	r_{62}	$m_6 ::$	$h_3 \Leftarrow Self : b_3 \| C_3 \cap \{Self \cong m_6\};;$

となる。手順 3) で、m_5 のファクトの整理を行なうと、

m_5	r_{51}	$m_5 ::$	$h_2 \Leftarrow Self : b_5 \| C_5 \cap \{Self \cong m_5\};;$
	f_{511}	$m_5 ::$	$o/[l \rightarrow (a \downarrow b)];;$

となる。

3.3 拡張された問い合わせ

3.3.1 基本概念

1 章で述べたように、知識情報処理におけるデータベースは、最初から完全な情報が与えられることはなく、部分情報しか与えられない。このような環境において、どのような問い合わせ処理が必要であろうか。筆者らは、以下の 2 点がまず、重要だと考えている。

1. 対象のデータベースに情報を追加したら、同じ問い合わせの解がどうなるか。(仮説推論)
2. ある問い合わせに対して、対象のデータベースには、どんな情報が足りないのか。(仮説生成)

上記の 2 点は、*Quixoteε* における、問い合わせと解の拡張に対応している。

(1) 仮説推論

Quixoteε の問い合わせは、

- $? - m : G_1, \ldots ||C.$
- $? - m : G_1, \ldots ||C;;$ &*program*;; プログラム;; &*end*.

のいずれかの形で与えられる。ここで, m はモジュール識別子、G_i はオブジェクト項または属性項、C は制約である。また &*program* と &*end* の間にはプログラム (データベース) を記述することができる。後者の形式の質問は、プログラムを付け加えた後に質問の処理を行なうことにあたり、*Quixoteε* では、一種の仮説推論と見なしている。

2章で述べたように、*Quixoteε* のデータベースは、3 つ組 (S, M, R) で表すことができる。データベース (S, M, R) に、(S_H, M_H, R_H) なるデータベース付き問い合わせ、$? - q;;(S_H, M_H, R_H)$ を行なうことは、データベース $(S \cup S_H, M \cup M_H, R \cup R_H)$ に対して、問い合わせ $? - q$ を行なうことと同じである。

データベース付き問い合わせが複数実行された場合、*Quixoteε* では、問い合わせに付加されているデータベースを、順番に問い合わせ対象のデータベースに追加していく。例えば、データベース DB への問い合わせ列は、以下のようになる。

このような仮説推論では、仮説として逐次的に複数追加したデータベースを追加する前の状態に戻したくなる。そのため、*Quixoteε* では、入れ子トランザクションの機構

DB に対する

問い合わせ列		同等な問い合わせ
$?-q_1;;DB_1$	$\Leftrightarrow$	$?-q_1;;DB \cup DB_1$
$?-q_2;;DB_2$	$\Leftrightarrow$	$?-q_2;;DB \cup DB_1 \cup DB_2$

を使って、上記のようなデータベース付き問い合わせを管理している。

Quixote では、ユーザが、問い合わせの前後に *begin_transaction*, *end_transaction*, *abort_transaction* を自由に宣言することができる。まず、*begin_transaction* にてデータベースのある状態を設定する。いくつかの問い合わせを行なった後、*end_transaction* とした場合、問い合わせ時に追加されたデータは保持される。*abort_transaction* と宣言すると、先ほど設定した *begin_transaction* の状態に戻る(ロールバックする)まで、追加されたデータベースが削除される。例えば、トランザクションも用いた、データベース DB への問い合わせ列は、以下のようになる。

DB に対する

問い合わせ列		同等な問い合わせ
$?-$ *begin_transaction*		
$?-q_1;;DB_1$	$\Leftrightarrow$	$?-q_1;;DB \cup DB_1$
$?-$ *begin_transaction*		
$?-q_2;;DB_2$	$\Leftrightarrow$	$?-q_2;;DB \cup DB_1 \cup DB_2$
$?-$ *abort_transaction*		
$?-q_3;;DB_3$	$\Leftrightarrow$	$?-q_3;;DB \cup DB_1 \cup DB_3$
$?-q_4$	$\Leftrightarrow$	$?-q_4;;DB \cup DB_1 \cup DB_3$
$?-$ *end_transaction*		

上記のような仕組みや仮説推論は、知識ベースの構築や思考実験、Trial and Error タイプの問い合わせ処理に適している。

(2) 仮説生成

Quixote の解は、一般に

- if *assumption* then *answer* because *explanation*
- NO

のいずれかの形で与えられる。ここで、*answer* は、解であり、問い合わせ中の変数の代入である。*assumption* は、ドット項に対するの制約の集合であり、解が成立するための仮説 (assumption) と呼ばれる。*explanation* は、解が導かれた理由を示す情報であり、導出木の情報である。

仮説とは、導出過程で充足されなかった制約の集合であり、プログラム自体に欠けていた情報と考えられる。*Quixote* では、この仮説付きの解を求めることを、一種の仮説生成と見なしている。

例えば、以下のようなデータベースを考える。

相撲:: 関脇/[上位 = 大関, 下位 = 小結];;
相撲:: 花田勝/[生年月日 = "S46.1.20"];;
現在:: 花田勝/[四股名 = 若ノ花, 番付 = 関脇];;
次場所::X/[番付 = Z] $\Leftarrow$ 現在 : X/[番付 = Y, 成績 = 優勝],
Y/[上位 = Z]|||
{$X \sqsubseteq$ 力士, $Y \sqsubseteq$ 番付};;

ここでは、相撲、現在、次場所の 3 つのモジュールがある。但し、現在 $\sqsupseteq_S$ 相撲, 次場所 $\sqsupseteq_S$ 相撲 とする。すなわち、現在および次場所モジュールは相撲モジュールの知識を継承する。次場所モジュールにあるルールの意味は、「今場所優勝した人は、次の場所では番付が一つ上がる」である。ここで、「次場所の若ノ花の番付は何ですか?」という質問は、*Quixote* では、以下のように行うことができる。

?- 次場所: 花田勝 /[番付 =X].

これに対して、*Quixote* は、仮説生成により以下の答を返すことができる。

if 現在: 花田勝. 成績 = 優勝 then
X = 大関

但し、*explanation* は省略している。この答は、現在の若ノ花の成績が優勝であれば、次場所の番付は大関であることを示している。すなわち、データベースに情報が欠落していた、現在モジュールの若ノ花の成績を仮定して答を導きだしている。

逆に、仮説推論を用いて、若ノ花の現在の成績が優勝であるという仮説のもとに、次場所の若ノ花の番付の問い合わせは、

$?-$ 次場所 : 花田勝/[番付 = X];; &*program*;;
&*rule*;; 現在 :: 花田勝/[成績 = 優勝];; &*end*.

であり、この場合には、解は、

X = 大関

となる。

3.3.2 実現

問い合わせ処理は、基本的にトップダウン処理であり、解釈系により実現されている。解釈系は、大きくは、推論処理を行うインタプリタと制約解消や単一化など項の操作の操作を行う制約解消系からなり、必要なデータは、データ管理部に保持されている。データ管理部では、推論処理を効率化するために、モジュール継承など幾つかの処理を前処理として行っている。

仮説推論として、プログラムの追加を行う場合には、一般にその前処理は無効となる。例えば、同じオブジェクト項に関する属性情報をマージする属性項のマージ処理は、包節関係の追加により、基本オブジェクト項どうしの束演算の結果が変わるため異なった結果になる可能性がある。したがって、現在の実装では、プログラムの追加が行われた場合は、前処理のやり直しを行っている。

また、仮説として追加されたプログラムの情報は、トランザクションに識別子を付加して、トランザクションごとのデータを保持している。

Quixote には、属性に関する情報は、(同じモジュールの) 同じ oid でマージされるという性質があるため、解釈系は、サブゴール の展開において、展開可能なルールを全て展開し、同一のオブジェクト項をヘッドに持つルールの展開の結果をマージしている (解の OR マージ)。

ルールの展開に際し、その本体制約のうち、ドット項を含みかつ充足されることが示されなかった制約は、assumption として扱われる。ルールの展開の結果得られる解は、assumption とゴール節に含まれる変数に対する制約 (conclusion) の組 (a, c) で表すことができる。仮説生成を伴う場合、解の OR マージは、assumption を含むため複雑になる。2 つの解 $s_1 = (a_1, c_1), s_2 = (a_2, c_2)$ 解の OR マージの手順を以下に示す。

1. a_1 が真であれば必ず a_2 も真のとき
   s_2 は残し、 s_1, s_2 をマージ した解を作って[1] 、 s_1 は捨てる。

2. a_2 が真であれば必ず a_1 も真のとき
   s_1 は残し、 s_2, s_2 をマージ した解を作って、 s_2 は捨てる。

3. お互い、片方が真であれば必ずもう一方も真のとき
   s_1, s_2 を マージ した解を作成し、 s_1, s_2 ともに捨てる。

4. a_1 と a_2 が矛盾しているとき
   両方の仮定部が共に成り立つ場合というのはないので、 s_1, s_2 をそのまま用いる。

5. 上記のどれでもないとき
   s_1, s_2 を マージ した解を生成する。

ところで、二解 $s_1 = (a_1, c_1), s_2 = (a_2, c_2)$ が、

a_1 が真であれば必ず a_2 のとき かつ

c_1 が真であれば必ず c_2 のとき

の関係にあるとき、 s_2 は、 s_1 から自明であるので、別の解とするほどのことはない。そこで、1, 2, 5 項において、マージ する二解がこの関係にある場合、マージ した解を生成するのは無駄であるので、実際には生成しない。

Quixote では、知識情報処理の応用のために高度な推論を行なえるよう、さまざまな拡張を行なっているが、そのために処理が重たくなる傾向がある。応用によっては、使用しない機能もある。そこで *Quixote* では、実験的環境を充実させる一つの手段として、問い合わせにモードを幾つか設けている。主なものに以下の 3 種類がある。

- 解で、仮説生成を行う / 行わない。
- 解で、*explanation* を求める / 求めない
- 問い合わせ処理中に、属性継承を
  - 行なう
  - 上向き継承のみ行なう
  - 下向き継承のみ行なう
  - 行なわない

4 DOOD システム *Quixote* の全体構成

Quixote システムは、UNIX 上のクライアント (ユーザインタフェース) と PIMOS 上のサーバ (推論部) が TCP/IP を介して通信する構成になっている。

ユーザインタフェースは以下のような部分からなる。

[1] 二解 $s_1 = (a_1, c_1), s_2 = (a_2, c_2)$ のマージとは、a_1 と a_2、 c_1 と c_2 を、同一の dot 項を 単一化しながら合わせた a_3, c_3 からなる解 $s_3 = (a_3, c_3)$ を作ることである。

- GNU-Emacs を利用した *Quixote* プログラムの実行できる Qmacs インタフェース
- API を用いる端末からのユーザインタフェースである Qshell
- X-Window を利用した、知識オブジェクトの束構造、モジュールの階層構造、質問に対する解の導出過程 (*explanation*) の表示を行なう、グラフィカルユーザインタフェース。

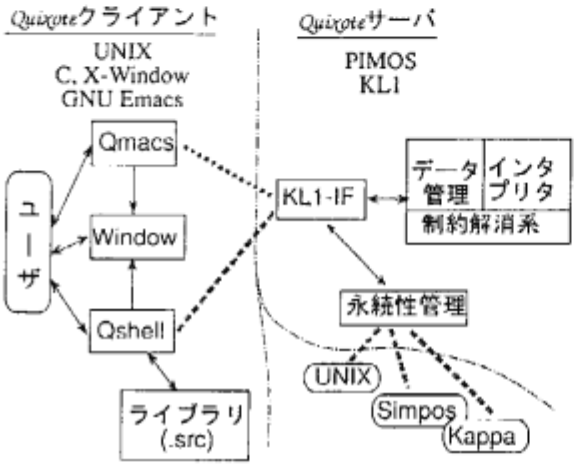

図 1: *QUIXOTE* 第 3 版の全体構成図

図 1はその全体構成図である。図の左方がユーザインタフェースであるクライアント、右方が推論を行なうサーバである。

これまでに、*Quixote* システム上に作られた応用プログラムには、法的推論、自然言語における状況推論、生物データベース等がある。

5 まとめと今後

本論文では、DOOD 言語 *Quixote* の特徴のうち、属性継承、モジュール、拡張された問い合わせを取り上げ、その実現を説明した。

Quixote システムをデータベースシステムとして見直した場合、問い合わせの拡張としての仮説推論の機能は、データベースの更新と考えることもできる。但し、通常のデータベースの更新処理では、データ共有などの点から、同時実行制御などの機能を持っている。*Quixote* も、演繹データベースの拡張として、更新規則によりファクトの更新を行なう機能を持っている。しかし、*Quixote* の更新規則や、仮説推論での更新は、同時実行制御などの機能が検討不足であり、今後の課題の一つである。

Quixote は ICOT Free Software として公開されているが、サーバは KL1 で記述されているため、その動作には ICOT の並列推論マシンが必要となる。より広範囲の応用に対して、*Quixote* の有効性を検証するためには、UNIX 環境で動作するものが望ましい。現在、FGCS の Follow-On project で、UNIX 環境だけで動く、機能縮小版の micro-*Quixote* 処理系も検討している。また、ICOT では、KL1 プログラムを C にコンバートして UNIX 環境上で動かす KLIC 処理系も開発中である。これを使えば、本来の *Quixote* システムが全て UNIX 環境で動く予定である。

さらに、大きな応用に耐えられるよう、機能／性能面での向上や、高度の知識処理のための拡張の方向性などについても検討を行なっている。

謝辞

貴重な意見や、コメントを下さった、*Quixote* グループに謝意を表する。

参考文献

[1] H.Aït-Kaci. "An Algebraic Semantics Approach to the Effective Resolution of Type Equations", *Theoretical Computer Science*, no.45, 1986.

[2] M.Kifer and G.Lausen. "F-Logic — A Higher Order Language for Reasoning about Objects, Inheritance, and Schema", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp.134-146, Portland, June, 1989.

[3] 津田宏 他. 「*Quixote* システム第 3 版」 ICOT Technical Memo (予定)

[4] H.Yasukawa, H.Tsuda, and K.Yokota. "Objects, Properties, and Modules in *Quixote*", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[5] K.Yokota and H.Yasukawa. "Towards an Integrated Knowledge-Base Management System", *Proc. Int.*

Conf. on Fifth Generation Computer Systems, ICOT, Tokyo, June 1-5, 1992.

[6] 近藤秀文, 平井千秋, 横田一正. 「束を用いた概念間の包摂関係に関する表現と利用方法」情報処理学会第46回(平成5年前期)全国大会, 1992

[7] K.Yokota H.Tsuda and Y.Morita. "Specific Features of a Deductive Object-Oriented Database Language *Quixote*", DOOD Workshop, 1993

[8] 津田宏, 横田一正. 「知識表現言語 *Quixote* による知識処理」第10回第五世代コンピュータ・シンポジウム 6月 1993

[9] 西岡利博, 小島毅, 津田宏, 横田一正. 「演繹オブジェクト指向データベース言語 *Quixote* の手続き的意味論」第94回データベースシステム研究会 7月 1993

[10] 横田一正, 柴崎真人. 「データベースに判決は予測できるか?」第94回データベースシステム研究会 7月 1993