統合知識ベース管理システム

— FGCS プロジェクトにおけるデータベース・知識ベースの研究開発の概要 —

横田 一正　安川 秀樹

第 3 研究室

(財) 新世代コンピュータ技術開発機構 (ICOT)

東京都港区三田 1-4-28 三田国際ビル 21F

Tel: 03-3456-3069　Fax: 03-3456-1618

{kyokota,yasukawa}@icot.or.jp

概要

知識表現言語と知識ベースは、知識情報処理システムで重要な役割を果たしている。これらのシステムをサポートするために、知識表現言語 *Quixote*、そのデータベースエンジンとしての並列データベース管理システム Kappa、*Quixote* と Kappa 上のいくつかの応用システム、そしてより柔軟な制御機構のための 2 つの実験的なシステムを開発してきた。

このシステム全体は、データベースの観点から、演繹オブジェクト指向データベース (DOOD) の枠組のもとにあると考えることができる。一方、データベースと自然言語処理との間にある多くの類似性を考えれば、状況理論における状況推論をサポートしていると見なすことができる。したがってわれわれの応用はこれらの両方の特徴を持っている。DOOD の応用としては分子生物学データベースと法的推論システム TRIAL があり、状況推論の応用としては時制推論システムがある。

効率的で柔軟な制御機構のために開発した 2 つのシステムは、unfold/fold 制約変換に基づく cu-Prolog、そして制約ネットワーク上の力学に基づく力学プログラミング DP である。

本論文では、統合知識ベース管理システムを目的とした FGCS プロジェクトにおけるデータベースと知識ベースの研究開発活動の概要を述べる。

1　はじめに

1982 年の第 5 世代コンピュータ・システム (FGCS)・プロジェクトの発足以来、論理と並列を枠組とする研究開発の一環として、多くの知識情報処理システムが設計・開発されてきた。これらのシステムには、電子化辞書データベース、数学データベース、分子生物学データベース、法的判例データベースなどさまざまなデータと知識があり、知識ベース管理システムによって効率的な処理が期待されている[1]。応用システムの大規模なデータと知識を表現し、処理し、管理することは、FGCS プロジェクトの主要な課題であり、われわれのデータベースと知識ベースに関する研究開発活動は、論理パラダイムの下で、このような環境のデータと知識に向けられてきた。

70 年代後半から、関係モデルの、非効率的な表現や不十分な問合せ能力などの欠点を克服するために、関係モデルを拡張した多くのデータモデルが提唱されてきた。これらの拡張の中で、演繹データベースは、その論理的基盤と強力な推論能力によって、論理プログラミング分野と人工知能分野の多くの研究者を魅き付けた。演繹データベースの多くの成果は、データベースの理論的側面を定義し、強力な問合せ処理能力を示した。しかしながら、応用の観点からは、そのデータをモデル化するという能力にはむしろ貧弱であった。これは主に、一階述語を基にした表現という、関係モデルの欠点を継承しているためである。一方、オブジェクト指向データベースが CAX データベースやマルチメディア・データベースなどの“新しい”応用に対処するために、関係モデルの拡張の中で多くの支持を獲得し始めている。オブジェクト指向概念による柔軟性や適応性は、たとえ、オブジェクト指向データベースがその形式化や意味論に多くの欠点を持っているとしても、演繹データベースの文脈の中においても検討されるべきである。

これらの利点を統合することは、高度応用にとって妥当なものなので、われわれは演繹 (•) オブジェクト指向データベース (DOOD) を提唱した [Yokota and Nishio 1989][2]。

---

[1] “データベース”と“知識ベース”の境界ははっきりしていない。それらは文脈に依存して使用される。大多数のデータベース研究者は、たとえルールや、演繹や発見のような推論能力を持っていたものであっても、データベースという用語を好んで使用する。たとえば、演繹データベースやエキスパート・データベース、自己組織型データベースなどがある。本論文では、この使用法に従って、データベースという用語を使うことにする。本表題中の知識ベースという語は、「真の」知識ベースを実現するには、データベースの拡張に基づいたアプローチが、エキスパート・システムで使われている伝統的な知識ベースに基づくアプローチより有望だというわれわれの見解を示している。

それは、関係モデル（あるいは演繹データベースとオブジェクト指向データベース）を、論理、データモデル、計算モデルの3つの方向で拡張するものである。DOODは、こういった拡張による枠組である。一方、DOODと自然言語処理の多くの類似性を考えれば、この枠組が自然言語処理の状況推論にとっても妥当なものとなっている。このような観察は、データベースと知識ベース・知識表現言語の統合による知識ベース管理システムへとわれわれをはっきり導いてくれる。

FGCSプロジェクトにおいては、上記の観察に基づき、知識ベース管理システムの目標としてDOODに焦点を合わせ、知識ベース（あるいは知識表現）言語としての *Quixote*、そしてその下層のデータベース・エンジンである Kappa、そしてそれらの応用システムを開発してきた。*Quixote* は DOOD 言語であり、それに基づいて DOOD システムを実装した。これらの特徴については、2節でその特徴を述べる。DOOD システムで大量のデータを効率的に処理するために、その下層にデータベース・エンジンがなくてはならない。このエンジンは Kappa と呼ばれ、そのデータモデルは DOOD のサブクラスである非正規関係モデルである。より効率的な処理のために、並列データベース管理システム Kappa-P が、並列推論マシン上に実装されている。このデータモデルとシステムについては 3 節で述べる。この DOOD システムの上に、法的推論システム (TRIAL) や分子生物学データベース、自然言語処理の時制推論システムのようないくつかの応用も開発している。これらの概要については、4 節で触れる。上記の研究と共に、論理プログラミングのより柔軟な制御のために、制約変換と力学プログラミングの研究開発を行っており、その成果を *Quixote* に埋め込むことを予定している。これらについては5 節で説明する。これらの関連性について図1で示す。最後に、知識ベース管理システムの拡張に関連して、関連研究と今後の予定について述べる。

2 知識表現言語 (*Quixote*)

知識ベースに対するわれわれのアプローチは、すでに述べたように、演繹オブジェクト指向データベース (DOOD) に従っている。その中核となる *Quixote* と呼ばれる言語は多様な特徴を持っている[3]。つまり図1 中に示したものの他に、制約論理プログラミング言語、状況プログラミング言語、オブジェクト指向データベースプログラミング言語、そして DOOD 言語としての特徴を持っている。

[2]そのような統合に向けた国際会議が京都とミュンヘンで開催された [Kim *et al.* 1990, Delobel *et al.* 1991]。

[3]詳細については [Yasukawa *et al.* 1992] を参照されたい。

2.1 基本概念

Quixote の基本的な概念を理解するために、図2の文献データの例[4]を考えてみよう。この図において、項の3番

```
文献 (参照 ('Patterson et al (1981)'),
      1991/4/24,
      [種類 (paper),
       著者 (['D. Patterson',
              'S. Graw',
              'C. Jones'
             ]),
       標題 ('Demonstration by somatic cell genetics
             of coordinate regulation of genes for two
             enzymes of puring synthesis assigned to
             human chromosome 21'),
       誌名 ('Proc. Natl. Acad. Sci. USA'),
       巻 (78),
       ページ (405-409),
       発行年 (1981)
      ]).
```

図 2: Prolog のレコード (項)

目の引数が特異である。つまり、それはラベル (関数子) と値の対から構成されている（リストの形式の）組である。"著者" のラベルは集合値をリスト形式で持っており、そしていくつかの値はより複雑な値 (他の組) となるかもしれない。ユーザは、このような構造とこれらの項の単一化に対して責任を持ってプログラムを書かなければならない。このような構造が必要な理由はレコード型 (スキーマ) を前もって決定することができないからである。すなわち、オブジェクトそれ自体は一般的には不安定なものなので、オブジェクトについては部分情報しか得ることができないのである。この特徴は、関係モデルの伝統的な正規化の手法を、このようなデータベースの設計には必ずしも適用できないことを示している。オブジェクト識別性の概念を導入することにより、このレコードは識別子を含む2項関係の集合という形式で表現することができる。しかしながら、これは余りに非効率的な表現である。

Quixote は、複合オブジェクトの構成子に基づいたオブジェクト識別子 (oid) とプロパティの概念を導入している。図2 の例は、図3 のように *Quixote* で表現することができる。この図の "/" の左側が oid (*Quixote* ではオブジェクト項 (o-項) と呼ぶ) で、右側が関連するプロパティである。オブジェクトは、ひとつの oid とそのプロパティから構成され、*Quixote* では、同一の oid を持った属性項 (a-項) の集合として書くこともできる。たとえば

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

のように分解・合成することができる。このような記述は部分情報の処理に効果的である。o-項の中の属性は、そ

[4][Yoshida 1991] の例を修正している。

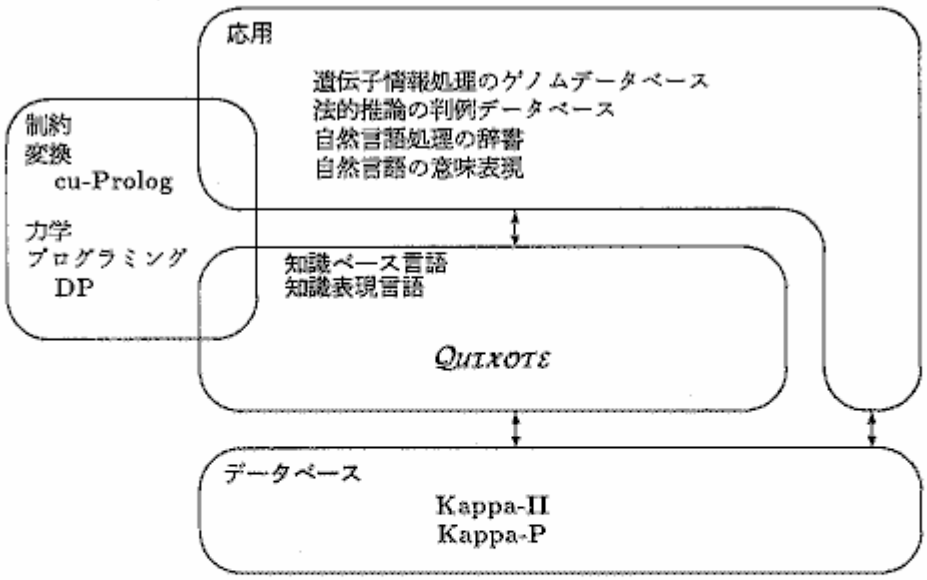


図 1: FGCS プロジェクトの知識ベース (DOOD) 管理システムの枠組

```
文献[参照 ='Patterson et al (1981)']/
    [日付 =1991/4/24,
     種類 =paper,
     著者 ={'D. Patterson',
            'S. Graw',
            'C. Jones'
            },
     標題 ='Demonstration by somatic cell genetics
           of coordinate regulation of genes for two
           enzymes of puring synthesis assigned to
           human chromosome 21',
     誌名 ='Proc. Natl. Acad. Sci. USA',
     巻 =78,
     ページ =405-409,
     発行年 =1981
    ].
```

図 3: *QUIXOTE* のオブジェクト

のオブジェクトに固有のものである。

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

この "/" の右側の "[···]" は、$o[l=c]$ への属性付加と呼ばれる。o-項中の属性は、固有 (もしくは不変) 属性と呼ばれ属性付加中にのみ現れる属性を、非固有 (もしくは可変) 属性と呼ぶ[5]。

もうひとつの問題は、oid とプロパティの表現力である。まず、論理プログラミングの方式に従って、oid はルールの集合によって下記のように内包的に定義することができる。

パス [起点 $=X$, 終点 $=Y$] $\Leftarrow$ アーク [起点 $=X$, 終点 $=Y$].
パス [起点 $=X$, 終点 $=Y$] $\Leftarrow$ アーク [起点 $=X$, 終点 $=Z$],
　　パス [起点 $=Z$, 終点 $=Y$].

このプログラムでは、"アーク [起点$=a$, 終点$=b$]" のようなファクトの集合から "パス [起点$=X$, 終点$=Y$]" が推移的に定義され、さらに oid は、プログラムの実行の結果として、X と Y を具象化することによって生成される。これによって、ひとつのオブジェクトが、たとえ異なった環境で生成されたとしても、一意的な oid を持つことが保証される。さらにまた、タグを導入し循環的なオブジェクトを定義したり、集合構成子と組構成子を持ついわゆる複合オブジェクトを表現することもできる。

$$X@o[l=X] \iff X|\{X=o[l=X]\}$$
$$o[l=\{a,\cdots,b\}] \iff o[l=a]\wedge\cdots\wedge o[l=b]$$

最初の例は、変数 X は制約 $X=o[l=X]$ を持つ oid であることを示している。2番目は、o-項中の集合が、集合構成子を持たない o-項の連言に分解可能だということを示している。

[5] ときたま属性とプロパティという用語を誤用することがあるかも知れない。属性とプロパティのどちらも通常はラベルと (複合) 値の対であるが、属性という用語はレコード構造の文脈で使用されることが多く、プロパティという用語はオブジェクト構造の文脈で使用されることが多い。*QUIXOTE* では、ラベルと値の対 (あるいはラベルと関係子と値の三組) を属性と呼ぶ。しかし、継承の文脈では伝統的にプロパティ継承という用語を使用する。後述するように、*QUIXOTE* では非固有属性のみが継承されるので、非固有属性を単にプロパティと呼ぶ。さらに補足すれば、属性-値対というように、属性という用語は単にラベルだけを意味することもあるが、その意味は、通常は文脈から明かであろう。

一方、プロパティはその値が確定していなくても制約の形式で表現できる。すなわち、ラベルと値(の集合)の間に包摂関係に対応する演算子を導入し、それらを、ドット記法を導入することによって制約の集合と考える。

$$
\begin{aligned}
o/[l=a] &\iff o/|\{o.l \cong a\} \\
o/[l\to a] &\iff o/|\{o.l \sqsubseteq a\} \\
o/[l\leftarrow a] &\iff o/|\{o.l \sqsupseteq a\} \\
o/[l=\{a,\cdots,b\}] &\iff o/|\{o.l \cong_H \{a,\cdots,b\}\} \\
o/[l\to\{a,\cdots,b\}] &\iff o/|\{o.l \sqsubseteq_H \{a,\cdots,b\}\} \\
o/[l\leftarrow\{a,\cdots,b\}] &\iff o/|\{o.l \sqsupseteq_H \{a,\cdots,b\}\}
\end{aligned}
$$

"/|" の右側はプロパティに関する制約の集合である。ただし、$\sqsubseteq_H$ と $\sqsupseteq_H$ はそれぞれ、$\sqsubseteq$ と $\sqsupseteq$ によって定義される Hoare 順序によって生成される半順序であり、$\cong_H$ はその順序に関する同値関係である。もしオブジェクト o がラベル l に関して属性が指定されていない場合には、o の l-属性は何らの制約も持たないものとみなされる。

oid の意味論は、超集合 (hyperset) [Aczel 1988] のサブクラスであるラベル付きグラフの集合の上で定義される。oid はラベル付きグラフに写像され、属性はラベル付きグラフの集合上の関数に対応している。この意味で属性は、F-logic [Kifer and Lausen 1989] のように、オブジェクトのメソッドと考えることができる。

超集合論のサブクラスを意味論の定義域として採用した理由は、無限データ構造を扱うためである。この詳細については [Yasukawa *et al.* 1992] を参照されたい。

2.2 包摂関係とプロパティ継承

基本 (非構造) オブジェクトの集合に半順序関係が与えられると、具象オブジェクト項の集合から束を構成することができる。その順序は包摂関係 $\sqsubseteq$ と呼ばれる。これは既に、プロパティを制約として表現するのに用いられている。この順序関係に従って、プロパティはオブジェクト間を上方 (かつ / あるいは) 下方へと継承される。プロパティ継承の一般規則は、下記のようになる。

$$o_1 \sqsubseteq o_2 \supset o_1.l \sqsubseteq o_2.l$$

ただし、固有属性は継承の範囲外である。この規則に従って、下記の結果が得られる。

$$
\begin{aligned}
o_1 \sqsubseteq o_2,\ o_2/|\{o_2.l \sqsubseteq a\} &\implies o_1/|\{o_1.l \sqsubseteq a\} \\
o_1 \sqsubseteq o_2,\ o_1/|\{o_1.l \sqsupseteq a\} &\implies o_2/|\{o_2.l \sqsupseteq a\} \\
o_1 \sqsubseteq o_2 \sqsubseteq o_3,\ o_2/|\{o_2.l \cong a\} &\implies o_1/|\{o_1.l \sqsubseteq a\}, \\
&\qquad\quad o_3/|\{o_3.l \sqsupseteq a\}
\end{aligned}
$$

$o_2/|\{o_2.l \sqsubseteq a\}$ は $o_2/[l\to a]$ のことであるので、プロパティ継承は制約の継承と考えることができる。したがって、多重継承は制約のマージとして、上方と下方の両方で定義できる。

$$
\begin{aligned}
o_1 \sqsubseteq o_2,\ o_1 \sqsubseteq o_3,\ o_2/[l\to a],\ o_3/[l\to b] \\
\qquad\qquad \implies o/[l\to \mathrm{meet}(a,b)] \\
o_1 \sqsupseteq o_2,\ o_1 \sqsupseteq o_3,\ o_2/[l\leftarrow a],\ o_3/[l\leftarrow b] \\
\qquad\qquad \implies o/[l\leftarrow \mathrm{join}(a,b)]
\end{aligned}
$$

制約の集合は、制約解消系によって簡約される。複合 o-項においては、固有属性はプロパティ継承の例外を生じさせる。

$$o[l=a] \sqsubseteq o,\ o/[l\to b] \implies o[l=a]/[l=a]$$

2.3 プログラムとデータベース

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。m をモジュール識別子 (mid) (構文的には o-項に同じ)、a を a-項とすると、$m:a$ は「m は a をサポートする」という命題である。mid m、a-項 a、命題 $p_1,\cdots,p_n$ によってルールは下記のように定義される。

$$m :: a \Leftarrow p_1,\cdots,p_n.$$

これは、「mid m を持つモジュールは '$p_1,\cdots,p_n$ が成り立つならば、a は mid m を持つモジュールで成り立つ' というルールを持つ」ことを意味している。もし、p_i において mid が省略されていたら既定値として m が取られ、もし、m が省略されたらそのルールはすべてのモジュールで成り立つことを意味している。a はヘッド、$p_1,\cdots,p_2$ はボディと呼ばれる。a-項は、o-項と制約の集合とに分割することができるので、このルールは下記のように書き換えることができる。

$$m :: o/|C_H \Leftarrow m_1:o_1,\cdots,m_n:o_n||C_B.$$

ただし $a \doteq o/|C_H$, $p_i \doteq m_i:o_i|C_i$ かつ $C_B = C_1 \cup \cdots \cup C_n$ である。C_H はヘッド制約、C_B はボディ制約と呼ばれる。これらの定義域はラベル付きグラフの集合である。ボディの a-項による制約は、C_B に包めることができることに注目されたい。従来の制約論理プログラミングに比べると、ヘッド制約を分離しているという点で拡張されている。

モジュールは、同一の mid を持つルールの集合として定義される。モジュール間の非巡回関係を**サブモジュール関係**として定義する。これは下記のように、**ルール継承**のために利用される。

$$
\begin{aligned}
m_1 &\sqsupseteq_S m_2 \\
m_3 &\sqsupseteq_S m_4 \cup (m_5 \setminus m_6)
\end{aligned}
$$

ここでは、m_2 のルールの集合を m_1 が継承し、$m_4 \cup (m_5 \setminus m_6)$ のような集合演算によって定義されるルールの集合を m_3 が継承している。共通集合や差のような集合演算は構文的に評価される。たとえモジュールがパラメータを持っていても、すなわち mid が変数を伴う o-項であっても、サブモジュール関係が定義できる。ルール継承の例外を扱うために、それぞれのルールは局所性と上書き

といった性質を持っている。つまり、局所ルールは他のモジュールに継承されず、上書きルールは、同一のヘッドを持つルールの他のモジュールからの継承を阻害する。

プログラムあるいはデータベースは、包摂関係とサブモジュール関係の定義を伴ったルールの集合として定義される。ルールの集合はまたモジュールの集合と見なすこともできる。もし2つのモジュール間にサブモジュール関係が、たとえ推移的にでも、定義されていなければ、同じ oid を持つオブジェクトは、それぞれのモジュールの中で異なった (あるいは矛盾した) プロパティを持つことができる。プログラムの意味論は、mid と o-項に対応したラベル付きグラフの対の定義域で定義される。この枠組において、矛盾をも含むことのできる、大規模知識ベースの分類が可能となり、*Quixote* データベースの中に格納することができる。

2.4 更新と永続性

Quixote は入れ子トランザクションの概念を持っており、2種類のデータベースの更新を許している。

1) 問合せ時のデータベースの漸増的な追加、および

2) 問合せ処理中の、o-項と a-項の動的な挿入と削除

1) は、既に存在するデータベースに、問合せ時に新しいデータベースを付加できることを意味している。たとえば、下記のデータベース DB への問合せ列を考えてみよう。

DB に対する問合せ列		同等な問合せ
?- begin_transaction.		
?- Q_1 with DB_1.	$\Longleftrightarrow$	?- Q_1 to $DB \cup DB_1$
?- begin_transaction.		
?- Q_2 with DB_2.	$\Longleftrightarrow$	?- Q_2 to $DB \cup DB_1 \cup DB_2$
?- abort_transaction.		
?- Q_3 with DB_3.	$\Longleftrightarrow$	?- Q_1 to $DB \cup DB_1 \cup DB_3$
?- Q_4.	$\Longleftrightarrow$	?- Q_1 to $DB \cup DB_1 \cup DB_3$
?- end_transaction.		

上記の問合せ列の実行が成功すると、DB は $DB \cup DB_1 \cup DB_3$ に変化している。それぞれの DB_i は、ルールの集合の他に、包摂関係もしくはサブモジュール関係の定義を持つことができ、それは既に存在するデータベースの定義へマージされ、必要ならば、包摂もしくはサブモジュールの階層は再構築される。トランザクションのロールバックによって、このような問合せはまた、仮説推論にも使用することができる。

2) によって、問合せ処理中の o-項や、その (可変) プロパティの更新が可能となる。ここでトランザクションは、1) のトランザクションのサブトランザクションとして位置づけられる。更新の意味論を保証するために、いわゆる AND-並列と OR-並列の実行は禁止される。たとえば、下記の例は従業員の給料を更新するための単純なルールである。

賃上げ [年度 = 1992, 所属 = X]/[昇給率 = Y]
⇐begin_transaction;
従業員 [社員番号 = Z]/[所属 = X, 給与 = W];
− 従業員 [社員番号 = Z]/[給与 = W];
+ 従業員 [社員番号 = Z]/[給与 = New];
end_transaction
||{$New = W * Y$}.

ここで、";" は、AND-並列の実行を禁止するために逐次的な実行を指定するもので、"+" は挿入を、"−" は削除を意味している。

問合せ処理中に削除されたりロールバックされる一時的なオブジェクトを除き、*Quixote* のプログラムの中のすべての (外延あるいは内包) オブジェクトは永続性が保証されている。このような永続オブジェクトは、下層のデータベース管理システム (次節で説明) か、ファイル・システムに格納される。

2.5 問合せ処理

Quixote は、基本的には、オブジェクト識別性、複合オブジェクト、カプセル化、型階層、メソッドなどのオブジェクト指向の特徴を持つ制約論理プログラミング言語である。しかし、この問合せ処理は、ヘッド制約や oid がある点で従来の問合せ処理とは異なっている。たとえば、下記のプログラムを考えてみよう。

くじ[数字 = X]/[賞品$_1$ → a] ⇐ $X \sqsubseteq 2n$.
くじ[数字 = X]/[賞品$_2$ → b] ⇐ $X \sqsubseteq 3n$.
くじ[数字 = X]/[賞品$_1$ → c] ⇐ $X \sqsubseteq 5n$.

ここで $2n$ は2の倍数の型である。

?-賞品[数字 = 30]/[賞品$_1$ = X, 賞品$_2$ = Y]

という問合せが与えられると、その答えは $X \sqsubseteq \mathrm{meet}(a, c)$ と $Y \sqsubseteq b$、すなわち、

くじ[数字 = 30]/[賞品$_1$ → meet(a, c), 賞品$_2$ → b].

である。まず最初に、oid が存在するために、同一の oid を持つかも知れないすべてのルールは評価されなければならず、必要ならばマージされる。したがって、*Quixote* における問合せは常に全解を得るために処理される。第2に、*Quixote* のルールは2種類の制約 (ヘッド制約とボディ制約) を持っており、ボディ制約は仮定として働くためである。たとえば

くじ/[賞品 = 旅行] ⇐ ||{くじ. 数字 = 1234}.

というプログラムに、問合せ ?- くじ /[賞品 =X] を出すと、くじ. 数字 =1234 ならば X= 旅行、という回答が返される。したがって、G_i をサブゴールの集合、C_i を関連する変数の制約集合としたときの、従来の制約論理プログラミングの導出過程

$$(G_0, \emptyset) \rightarrow \cdots \rightarrow (G_i, C_i) \rightarrow \cdots \rightarrow (\emptyset, C_n)$$

とはと異なっている。G をサブゴールの集合、A をドット項のボディ制約で構成される仮定の集合、C をヘッド制約と変数環境とからなる制約集合としての結論の集合とすると、*Quixote* の導出列の各ノードは (G, A, C) である。正確に言えば、*Quixote* では、導出は列ではなく、有向非巡回グラフである。なぜならば、仮定と制約の間の包摂関係によって、2つの列がマージされることがあるからである。たとえば、(G, A, C) と (G, A, C') がマージされて $(G, A, C \cup C')$ となる。図4にそのような導出が示されている。ただし2つの列をマージすることが可能な環

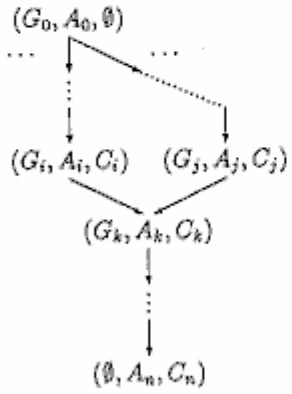


図 4: *Quixote* における導出

境は限定されている。つまり、ルールのヘッドの包摂関係により継承されるルールも含んだ、いわゆる OR- 並列による結果の時だけで、それはもっとも内側からマージされる。*Quixote* の問合せ処理の現在の実装では、全解を得るために OLDT のようなタブラー法を基礎にしている。横方情報伝達 (sideways information passing) についても、束縛情報だけでなく、プロパティ継承をも考慮することによって、実装されている。

2.6 *Quixote* システム

Quixote のシステム環境は図5の通りである。この節では *Quixote* システムのいくつかの特徴を列挙する。

- *Quixote* プログラムは、制御情報、包摂関係、サブモジュール関係、そしてルール集合の4つの部分から構成されており、それはソース・コードとオブジェクト・コードの2つの形式で永続記憶に格納される。永続性は永続マネジャによって制御され、それはプログラムをどこに格納すべきか切替える。オブジェクト・コードの中のルールの集合は、従来の演繹データベースのように、外延および内包データベースに分割できるように最適化されている。

- ユーザが *Quixote* で巨大データベースを構築する時には、モジュールの概念とは別に、巨大データベースを小さなデータベースの集合として書くことができる。それらは1つのデータベースの中に集められる。すなわち、データベースは他のデータベースの中で再使用することができる。

- ユーザが *Quixote* で書いたデータと知識を利用するときには、*Quixote* サーバを通して、多数のデータベースへ同時にアクセスすることができる。ただし *Quixote* の同時実行制御は現在単純な実装しかなされていない。

- ユーザは *Quixote* データベースを2種類のインタフェースで利用できる。ESP [Chikayama 1984] や KL1[Ueda and Chikayama 1990] で書いた応用プログラムを経由する場合と、Qmacs と呼ばれる固有のウィンドウ・インタフェースを通す場合である。

Quixote の一版は 1991 年 12 月に、第2版は 1992 年の4月にリリースされた。両方の版とも KL1 で書かれていて、並列推論マシン (PIM) [Goto *et al.* 1988] とその上の PIMOS [Chikayama *et al.* 1988] の呼ばれるオペレーティング・システム上で稼働している。

3 高機能データベース管理システム (Kappa)

Quixote で書かれた大規模データベースを効率的に処理するために、Kappa と呼ばれるデータベース・エンジンを開発した[6]。この節ではその特徴を説明する。

3.1 非正規関係と *Quixote*

豊富な表現能力と効率的な処理能力とはトレードオフの関係にあるので、*Quixote* のどの部分をデータベース・エンジンによってサポートするかが問題となる。またこのデータベース・エンジンは、実用的なデータベース管理システム (DBMS) の役割を果たすことを意図している。われわれの知識情報処理環境中での多様なデータと知識を考慮し、*Quixote* の無限構造を持たないオブジェクト項のクラスに対応する拡張非正規関係モデルを採用した。"拡張"という用語の意味は、それが Prolog の項のような新しいデータ型をサポートし、多様な応用のための拡張可能性を提供するからである。非正規関係モデルを採用した理由は、当然ながら、効率的表現と効率的処理を達成するためである。

非正規関係は、直観的には、定義域あるいは他の非正規関係のデカルト積の部分集合として定義される。D をアトムの値の集合とすると

$$NR \subseteq E_1 \times \cdots \times E_n$$
$$E_i ::= D \mid 2^{NR}$$

すなわち、この関係は階層構造と、値として他の関係の集合を持つことができる。これは、組構成子と集合構成子の

[6]詳細は [Kawamura *et al.* 1992] を参照されたい。

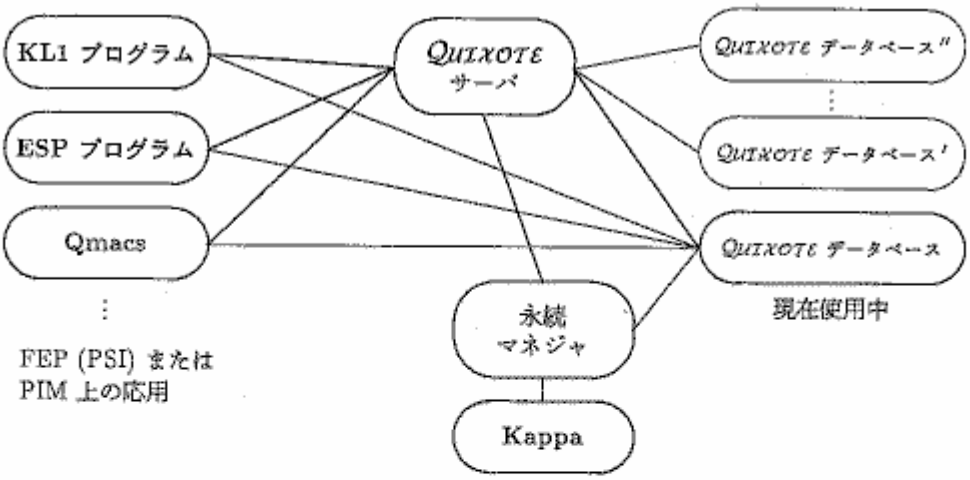


図 5: QUIXOTE の環境

導入に対応している。構文的および意味的な制限の観点から見ると、そこには多様なサブクラスがある。拡張関係代数は、そのそれぞれに定義されうる。

Kappa の非正規関係では、集合構成子は下記のように正規関係の集合の省略形としてのみ使用される。

$$\{r[l_1 = a, l_2 = \{b_1, \cdots, b_n\}]\}$$
$$\Longleftrightarrow \quad \{r[l_1 = a, l_2 = b_1], \cdots, r[l_1 = a, l_2 = b_n]\}$$

"⇒" の操作はアンネスト (unnest) に対応しているが、逆向きの操作 (⇐) はネスト (nest) あるいはグループ・バイ (group-by) に対応している。⇐ はその操作の適用順序によって必ずしも合流性はないが、Kappa での非正規関係の意味論は、集合構成子を持たない (正規) 関係と同じであるので、意味は変わらない。このような意味論をとる理由は、効率的な処理のための一階の意味論を保持するためと、広く使われている関係データベースとの両立性を存続させるためである。非正規組 nt を与えられたとき、集合構成子をもたない対応する組の集合を $\underline{nt}$ とする。非正規関係 NR を

$$NR = \{nt_1, \cdots, nt_n\}$$
$$\text{ただし } \underline{nt_i} = \{t_{i1}, \cdots, t_{ik}\} \text{ for } i = 1, \cdots, n,$$

とする。そうすると、NR の意味論は

$$\bigcup_{i=1}^{n} \underline{nt_i} = \{t_{11}, \cdots, t_{1k}, \cdots, t_{n1}, \cdots, t_{nk}\}.$$

となる。この非正規関係データベースに関する拡張関係代数が Kappa で定義されている。この代数は、上記の意味論に従ってた結果を生み出すが、それは、ラベル階層の処理を除き、対応する関係データベースに対する結果と同じであることを保証している。

問合せは、一階言語として定式化することができ、一般的には、非正規組によって構成されたルールの形式をしていると考えることができる。データベースのファクト間の関係は、証明論的な見地からすれば連言であるので、ルールの意味論は上記の意味論によって明らかである。たとえば、下記のルール

$$r[l_1 = X, l_2 = \{a, b, c\}]$$
$$\Leftarrow B, r'[l_2 = Y, l_3 = \{d, e\}, l_3 = Z], B'.$$

は、集合構成子のない次のルールの集合に変換できる。

$$r[l_1 = X, l_2 = a]$$
$$\Leftarrow B, r'[l_2 = Y, l_3 = d, l_3 = Z], r'[l_2 = Y, l_3 = e, l_3 = Z], B'.$$
$$r[l_1 = X, l_2 = b]$$
$$\Leftarrow B, r'[l_2 = Y, l_3 = d, l_3 = Z], r'[l_2 = Y, l_3 = e, l_3 = Z], B'.$$
$$r[l_1 = X, l_2 = c]$$
$$\Leftarrow B, r'[l_2 = Y, l_3 = d, l_3 = Z], r'[l_2 = Y, l_3 = e, l_3 = Z], B'.$$

すなわち、それぞれのルールもまたアンネストすることができる。Kappa の関係を効率的に処理するポイントは、ネストとアンネストの演算数を減少することである。すなわち、可能な限り集合を直接的に処理することである。

この意味論では、非正規関係に対する問合せ処理は、論理プログラミングの従来の手順とは異なっている。たとえば、ただひとつの組で構成されている単純なデータベースを考えてみよう。

$$\{r[l_1 = \{a, b\}, l_2 = \{b, c\}]\}$$

問合せ $?\text{-}r[l_1 = X, l_2 = X]$ に対して、$X = \{b\}$ の答えを得ることができる。すなわち、それは $\{a, b\}$ と $\{b, c\}$ の共通集合である。つまり、このためには単一化の概念が拡張されなければならない。このような手順を一般化するために、2 つの概念を手続き的意味論に導入しなければならない [Yokota 1988]。

1) 剰余ゴール
下記のプログラムと問合せを考えてみよう。

$$r[l=S'] \Leftarrow B.$$
$$?\text{-}r[l=S].$$

もし $S \cap S'$ が、$r[l=S]$ と $r[l=S']$ の単一化で空集合とならなければ、新しいサブゴールは、$r[l=S \setminus S'], B$ となる。すなわち、もし、$S_1 \setminus S_2$ が空集合でなければ、剰余サブゴール $r[l=S \setminus S']$ が生成され、さもなければ単一化は失敗する。ここで注意すべきことは、もし複数の属性に集合値があれば、複数の剰余サブゴールが生成されるかもしれないことである。

2) 制約としての束縛
次のデータベースと問合せを考えてみよう。

$$r_1[l_1=S_1].$$
$$r_2[l_2=S_2].$$
$$?\text{-}r_1[l_1=X], r_2[l_2=X].$$

$r_1[l_1=X]$ と $r_1[l_1=S_1]$ の単一化によって $X=S_1$ と、新しいサブゴール $r_2[l_2=S_1]$ を得ることができるが、次の単一化の結果として $r_2[l_2=S_1 \cap S_2]$ と剰余サブゴール $r_2[l_2=S_1 \setminus S_2]$ を得る。$X=S_1 \cap S_2$ という答を得るべきであるので、このような手順は間違いである。この状況を回避するために、束縛情報は暫定的で、保持すべき制約としての役割を持たねばならない。

$$r_1[l_1=X], r_2[l_2=X]$$
$$\Longrightarrow r_2[l_2=X] \| \{X \subset S_1\}$$
$$\Longrightarrow \| \{X \subset S_1 \cap S_2\}.$$

ここで残っているひとつの問題は、非正規関係の一意的な表現は、既に言及したように、Kappa モデルでは必ずしも決定されないことである。一意的な表現を決定するために、それぞれの非正規関係は Kappa ではネストするためのラベルの順序を持っている。

上記の概念によって定義された Kappa での拡張関係代数の手続き的意味論によって、Kappa データベースは原理的には、非正規関係モデルの意味においても、必ずしも正規化されている必要はない。すなわち、ユーザは行ネスト構造を意識する必要性はない。

さらに、非正規関係モデルは、多値従属性の場合に関係の数を減らすことができることがよく知られている。したがって、Kappa モデルは関係モデルよりも、関係と組の数を減らすことによる効率的な処理と、複合構造によるよる効率的な表現を保証している。

3.2 Kappa システムの特徴

Kappa の非正規関係モデルはすでに実装されており、これには、逐次 DBMS Kaapa-II と並列 DBMS Kappa-P の 2 種類がある。Kappa-II は ESP で書かれており、逐次推論マシン (PSI) とそのオペレーティング・システム (SIMPOS) 上で稼働している。一方、Kappa-P は、KL1 で書かれており、並列推論マシン (PIM) とそのオペレーティング・システム (PIMOS) 上で稼働している。これらは環境が異なっているので必ずしも同じ設計ではないが、この節ではこれらに共通の特徴について説明する。

- データ型

Kappa は、知識情報処理環境の DBMS を目指し、新しいデータ型として項が加えられた。それは、多様なデータと知識はしばしば項の形式で表現されるからである。これらの操作のために、単一化とマッチングが付け加えられた。論理プログラミングの導出は、単一化ベースの関係代数によってエミュレートできるが、この代数はそれほど効率的ではないので、Kappa ではこの機能をサポートしていない。さらにまた、Kappa ではデータ型によって、2 バイト文字 (JIS) と 1 バイト文字 (ASCII) を識別する。これは、遺伝子配列データのような膨大なデータを圧縮する助けとなる。

- コマンド・インタフェース

Kappa では、2 種類のコマンド・インタフェースを用意している。下位レベル・インタフェースの基本コマンドと、上位レベル・インタフェースの拡張関係代数である。多くの応用では、(コストのかかる) 拡張関係代数のレベルは必ずしも必要ではない。このような応用では、ユーザは基本コマンドを使用することによって処理コストを減らすことができる。

DBMS とユーザ・プログラムの間の、通信コストを減らすために Kappa では、ユーザ定義コマンドを用意している。それは、Kappa-II では Kappa のカーネルと同じプロセスで、Kappa-P ではそれぞれのローカル DBMS と同じノードで実行される (これについては 3.3節で述べる)。

ユーザ定義コマンド機能によって、ユーザはそれぞれの応用システムに適したコマンド・インタフェースを設計したり、そのプログラムを効率的に走らせたりすることができる。Kappa の拡張関係代数は、組込みインタフェースではあるが、ユーザ定義コマンドの一部として実装されている。

- 実用性

既に述べたように、Kappa は、*Quixote* のデータベース・エンジンとしてだけではなく、実用的な DBMS として *Quixote* とは独立に稼働することも目指している。この目的を達成するために、いくつかの拡張機能がある。まず第一に、上記で言及したデータ型の他に、その応用の環境を格納するために導入された、新しいデータ型がある。それは、リスト、バッグ、そしてプールである。しかしな

がら、それらは、意味論上の困難さによって、拡張関係代数においては十分にはサポートされていない。

Kappa は、このようなデータ型のために SIMPOS や PIMOS が持っているのと同一のインタフェースをサポートしている。

ウィンドウ・システムから Kappa データベースを使うために、Kappa はスプレッド・シート風のユーザ指向のインタフェースを用意している。それには、更新も含んだアドホックな問合せ機能、様々な出力フォーマットでのブラウジング機能、そしてカスタマイズ機能などがある。

- 主記憶データベース

頻繁にアクセスするデータは、主記憶データベースとして主記憶にロードし保持しておくことができる。この主記憶データベース機能によって二次記憶データベースに余分の負荷を負わせたくないので、現在の実装では、一時関係の効率的な処理のためにのみ設計されている。したがって、遅延更新や同期などの機能はサポートしていない。Kappa-P では、主記憶データベースは 2 次記憶データベースよりも、少なくとも 3 倍は効率的である。2 次記憶データベースと同期を取った主記憶データベースは、Kappa-P では、複製データベースの一種として実装されている。

実装の面では、Kappa の効率的な処理には、いくつかの指摘すべきポイントがある。これらのうち、2 点について述べる。

- ID 構造と集合演算

ネストした (非正規) 組は、関係の中で一意に決められる組識別子 (ntid) を持つ。それは、明示的に処理されるべきひとつの“オブジェクト”として扱われる。抽象的に言うと、非正規組、ntid、ntid の集合、そして非正規組の集合 (関係) の 4 つの種類の“オブジェクト”がある。基本的には図 6 のように示される、変換のためのコマンドがサポートされている。ただし、Kappa-P では集合はストリームとして扱われている。大部分の演算は、ntid か集合の形

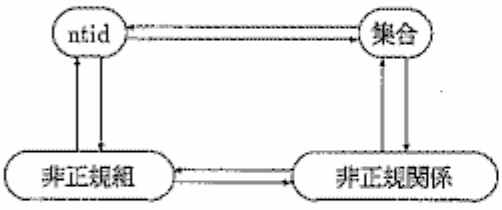


図 6: Kappa における“オブジェクト”とその基本操作

式で処理される。

選択結果の処理のために、非正規組のそれぞれの部分組はまた、仮想的に sub-ntid を持つ。(アンネストとネスト演算を含む) 集合演算は、対応する組を読むことなく、主に、(sub-)ntid もしくは集合の形式で処理される。

- 格納構造

意味的にはアンネストされた組で構成される非正規組はまた、同時にアクセスされるアンネストされた組の集合としてもみなされる。それゆえ、非正規組は分解されずに、圧縮され、原則的には 2 次記憶の同じページに格納される。遺伝子配列のような膨大な組には、連続したページが使われる。効率的に組にアクセスするためには、2 つのことを考えねばならない。まず、必要な組の位置を効率的に決定することと、組の中から必要な属性を能率良く取り出すことである。組の位置決定のためには、図 7 のように、ntid と論理ページ (lp) のアドレス変換テーブル、そして論理ページと物理ページ (pp) のアドレス変換テーブルを備えている。後者のテーブルは、下層のファイル・システムによって使用される。組から属性を取り出すために、非正規

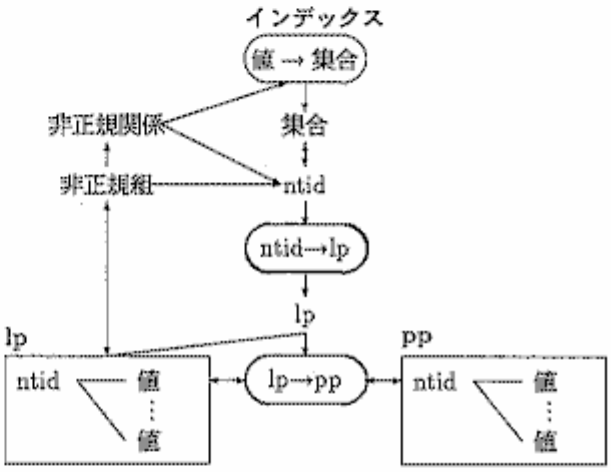


図 7: 二次記憶 DBMS のアクセス・ネットワーク

組のそれぞれのノードは、圧縮された組の中に局所ポインタとカウンタを持っている。ただしこれは更新負荷を増大させるので、更新処理の効率とのバランスが重要となる。

インデックスのそれぞれのエントリは、ネスト構造を反映し、sub-ntid を含んでいることもある。そのエントリの値は、部分文字列や結合などの文字列演算の結果であってもよく、またユーザのプログラムによって抽出された結果であってもよい。

3.3 並列データベース管理システム (Kappa-P)

Kappa-P は、並列 DBMS として多くの特徴を持っている。この節では、それらの概観を簡潔に述べる。

図 8 に、Kappa-P の全体的な構成を示している。インタフェース (I/F)・プロセス、サーバ DBMS、そしてローカル DBMS の、3 つの構成要素がある。I/F プロ

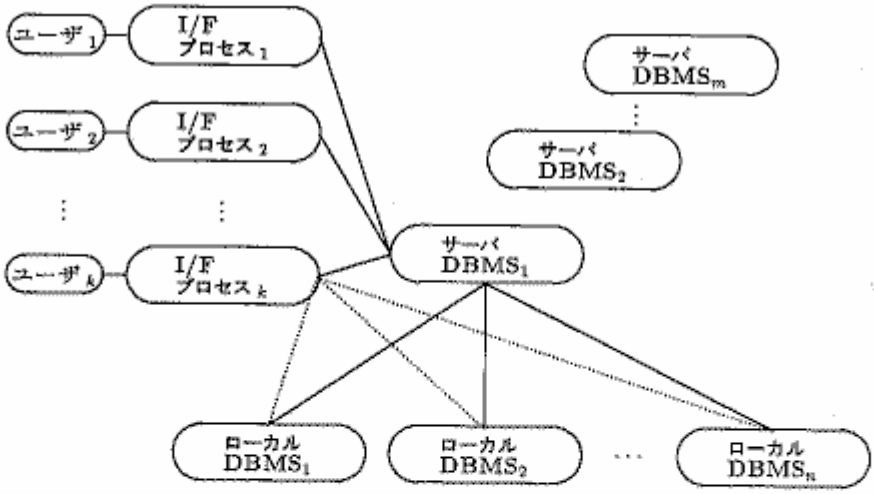


図 8: Kappa-P の構成

セスは、ユーザのプログラムによって動的に作り出され、ストリームによって、ユーザ・プログラムと（サーバもしくはローカル）DBMS との仲介をする。サーバ DBMS は、ローカル DBMS の所在のグローバル・マップを持っており、ユーザのストリームを該当するいくつかのローカル DBMS に直接つなげる働きをする。通信のボトルネックを回避するために、複数のサーバ DBMS を作ることが可能で、それらはすべてグローバル・マップの複製をもっている。ローカル DBMS 自体は、Kappa-II に対応する、単一の非正規関係 DBMS とみなすことができ、ユーザのデータはここに格納される。このデータは、複数のローカル DBMS に、水平分割を含んだ形で分散配置するか、複製することができる。もし、それぞれのローカル DBMS が共有メモリ（密結合）並列プロセッサである PIM のクラスターに置かれていれば、個々のローカル DBMS 自体が並列に動作する。ローカル DBMS 群は、分散メモリ（疎結合）並列マシンの各ノードに配置され、全体でひとつの分散 DBMS のように動く。

拡張関係代数を用いたユーザの手続きは、I/F プロセスにより、KL1 と似た構文の中間言語で書かれた手続きに変換される。その変換処理で、I/F プロセスは、必要ならば、どのローカル DBMS が処理のコーディネータとなるべきかを決定する。それぞれの手続きは対応するローカル DBMS に送られ、そこで処理される。結果は、いったんコーディネータに集められ、それから処理される。

Kappa-P が、他の並列 DBMS と異なっている点は、ユーザの応用プログラムと DBMS が同一マシンで動作する点である。もし、Kappa-P が従来の分散 DBMS のように、各ローカル DBMS で作成された結果を、（そのようなまとめ上げが不必要な場合でさえ）ひとつの結果にまとめ上げたとすると、並列処理の利点は減少する。このような状況を回避するために、ユーザの応用プログラムの一部は、図 9 のように、関連するローカル DBMS と同一のノードに置くことができる。この機能は、より高い並列

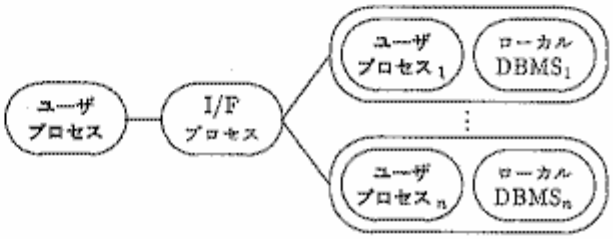


図 9: Kappa ノードのユーザ・プロセス

効果を引き出し処理効率を向上させることに寄与しているだけでなく、ユーザが応用に適したコマンドを設計しコマンド・インタフェースをカスタマイズすることにも寄与している。

4 応用

Quixote と Kappa 上に、3 つの応用システムを開発している。この節では、それぞれの研究のテーマについての概要を述べる。

4.1 分子生物学データベース

遺伝子情報処理システムは、科学的および工学的な観点のみならず、「ヒトゲノム・プロジェクト」に見られるように、社会的な観点からも大変重要なものである。また、ICOT においても知識情報処理という観点からこのようなシステムに携わってきた。この節において、*Quixote* と Kappa 上の分子生物学データベースに主に焦点を合わせ、その活動について説明する[7]。

[7]詳細は [Tanaka 1992] を参照されたい。

4.1.1 分子生物学データベースの必要条件

遺伝子情報処理の最も重要な目的は、目標とする蛋白質を設計しそれを生産することにあるが、現在のところあまりに多くの技術的な難題が残されている。蛋白質全体を考えれば、まだほんの一部しか解明できておらず、たくさんのノイズを伴った知識とデータを集め始めた段階に過ぎないといっても過言ではない。

このようなデータと知識の中には、配列、構造、そして遺伝子と蛋白質の機能といった多様なものが存在し、それぞれが相互に関連している。2重螺旋の形をしている遺伝子配列 (DNA) の中のひとつの遺伝子は、メッセンジャー RNA (mRNA) に複製され、アミノ酸の配列に翻訳され、そして蛋白質の一部 (もしくは全体) になる。このような過程は、生物学では「セントラル・ドグマ」と呼ばれる。同じ蛋白質の機能を持った、異なったアミノ酸配列が存在するかも知れない。遺伝子配列データのひとつの単位の大きさは、2～3個の文字 (塩基) のものから20万を越すものまでにわたり、そして、遺伝子データがしだいに一層解析されてくると、より長いものになってゆくだろう。たとえばヒト遺伝子配列はおおよそ30億個の塩基からなっており、これが単純な配列の上限である。蛋白質について解明されているのはほんの数パーセントに過ぎない。その解明のために配列データは基本的なデータであり、モチーフと呼ばれるパターンによる相同性の検索や、既知の蛋白質の機能から未知の機能の予想をするための配列間のマルチプル・アラインメントのために使用される。

分子生物学データベースのために考察すべきいくつかの問題がある。

- いかにこのような膨大な長さの配列値を格納するか、そしてそれらをいかに効率的に処理するか
- どのように配列データを表現し、それらにどのような演算を適用すべきか
- 化学反応のような蛋白質の機能をいかに表現するか、そして
- それらの関連性をいかに表現し、それらをいかに結合するか

である。データベースの観点からは、上記のデータと知識に関していくつかの点を考察しなければならない。

- 図2のような複合データの表現
- 変わりやすいデータの部分情報や雑音の多い情報の処理
- 上の3番目の問題点のような機能を表現する推論規則と推論機構、そして
- 生物学の概念と分子進化のような階層の表現

などである。

上記のような問題を考慮した結果、われわれは DOOD (*Quixote* + Kappa) 上にそのようなデータベースを構築することにした。大量の単純なデータは、効率的な処理のために Kappa-P の中に格納し、最適化したウィンドウ・インタフェースを通して、直接処理する。この分野には、生物学者の協力が絶対に必要な (不可欠な) ため、彼らをサポートするための環境も実装した。現在の実装の全体的な構成を図10に示す。

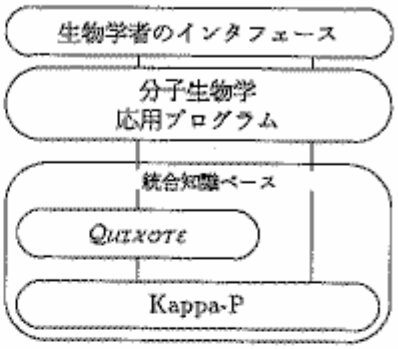


図 10: *Quixote* と Kappa 上の統合システム

4.1.2 *Quixote* と Kappa 上の分子生物学情報

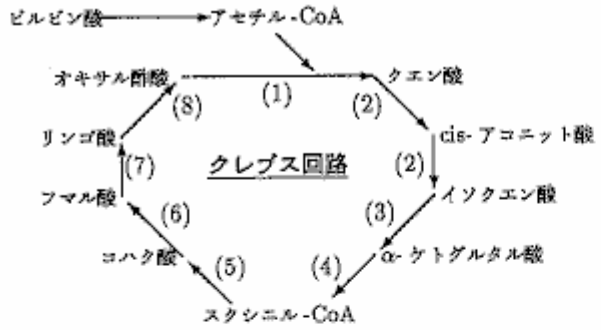
ここで、例として2種類のデータを考えてみたい。配列データと蛋白質の機能のデータである。

最初に、DNA 配列を考えてみる。このようなデータには推論規則は必要ではないが、相同性検索のための強力な能力が必要である。われわれのシステムにおいては、このようなデータは Kappa の中に直接に格納する。Kappa は大量のデータの格納をサポートし、高速の検索のためにユーザのプログラムによる元の配列を加工したインデックスを作ることを可能にする。情報検索のための配列指向のコマンドはそのようなインデックスを使い、ユーザ定義コマンドとして、Kappa の中に埋め込むことができる。さらにまた、図3に見られるように複合レコードは非正規関係として処理できるので、その表現もまた効率的である。Kappa は実用的な DBMS として、その有効性を示している。

次に、酵素と補酵素の化学反応について考えてみる。その図式を下記に示す。

$$\text{原料} + \text{補酵素} \overset{\text{酵素}}{\underset{\text{環境}}{\Longrightarrow}} \text{産物}$$

代謝反応の例として、図11のクレブス回路を考える。クレブス回路の化学反応は、図12のように *Quixote* におけるファクトの集合として書くことができる。この図では、$o_1 \sqsubseteq o_2/[\cdots]$ は $o_1/[\cdots]$ と $o_1 \sqsubseteq o_2$ を意味する。上記のファクトから反応鎖を得るためのルールは、以下のように *Quixote* で書くことができる。



酵素

(1) クエン酸合成酵素
(2) アコニット酸
(3) イソクエン酸脱水素酵素
(4) α-ケトグルタル酸脱水素酵素複合体
(5) スクシニル-CoA 合成酵素
(6) コハク酸脱水素酵素
(7) フマル酸
(8) リンゴ酸脱水素酵素

図 11: 代謝反応のクレブス回路

$$
\begin{aligned}
&\text{反応}\,[from = X, to = Y]\\
&\quad \Leftarrow W \sqsubseteq \text{反応}\,/[\text{原料}^+ \leftarrow X,\\
&\qquad\qquad \text{産物}^+ \leftarrow Z],\\
&\qquad \text{反応}\,[from = X, to = Y]\\
&\qquad \|\{\{X, Y, Z\} \sqsubseteq \text{反応}\,\}.\\
&\text{反応}\,[from = X, to = X]\\
&\quad \Leftarrow \|\{X \sqsubseteq \text{反応}\,\}.
\end{aligned}
$$

このような機能の表現には多くの難しさがあるにもかかわらず、*Quixote* ではそれらを容易に書くことを可能にした。

このほかの問題は、*Quixote* データベースと Kappa データベースをいかに統合するかである。Kappa のインタフェースを *Quixote* の中に埋め込むのがもっとも簡単なひとつの方法ではあるが、それはよりコストがかかり、*Quixote* の統一的な表現を破壊させるかもしれない。より良い方法は、Kappa、*Quixote* 両方に共通のオブジェクト識別子を管理させることで、そして共通のオブジェクトを保証することだろう。しかしながら、Kappa のそのような機能はまだ実装していない。現在の実装では図 10 のようにユーザにその一貫性の責任を負わせている。

4.2 法的推論システム (TRIAL)

最近、法的推論が人工知能研究者から、人工知能の大きな応用になるかもしれないとの高い期待を込めて、多くの注目を集めてきている。いくつかのプロトタイプ・システムが既に開発されている。われわれもまた、DOOD システムの応用のひとつとして法的推論システムを開発した[8]。

```
クレブス回路 :: {{
    クレブス 1⊑反応 /
                [原料+ ←{ アセチル coa, オキサロ酢酸 },
                 産物+ ←{ クエン酸, coa},
                 酵素 ← クエン酸合成酵素,
                 エネルギー = −7.7].
    クレブス 2⊑ 反応 /
                [原料+ ← クエン酸,
                 産物+ ←{ イソクエン酸, 水 },
                 酵素 ← アコニット酸].
     ⋮
    クレブス 8⊑ 反応 /
                [原料+ ← リンゴ酸,
                 産物+ ← オキサロ酢酸,
                 酵素 ← リンゴ酸脱水素酵素,
                 エネルギー = 7.1].
                }}
```

図 12: *Quixote* で書いたクレブス回路のファクト

4.2.1 法的推論の必要項目と TRIAL

最初に、法的推論の特徴について説明する。分析的法的推論の過程は 3 つの段階で構成されていると考えられる。それは、事実認定、法令解釈、そして法令適用である。

事実認定は、出発点として大変重要なものであるが、それは現在の技術ではとても難かし過ぎる。したがって、われわれのシステムでは、新規の事案は適切な形式であらかじめ表現されていると仮定している。法令解釈は、人工知能の観点からは最も興味深いテーマのひとつである。法的推論システム TRIAL は、法令適用のみならず法令解釈に焦点を合わせた。

法令解釈のためには多くのアプローチがあるが、われわれは下記の段階を採用している。

- 類比検出
  新規の事案が与えられると、既に存在する判例データベースからその事案に対して類似の判例を検索する。
- ルール変換
  類比検出によって抽出した判例 (解釈ルール) を、新規の事案がそれらに適用できるようになるまで抽象化する。
- 演繹推論
  ルール変換によって変換した抽象的解釈ルールに、新規の事案を演繹的方法で適用する。この段階は、同じ方法が使用される法令適用を含んでいてもよい。

これらの段階の中で、類比検出のための戦略はより良い判例のより効率的な検索のために法的推論に欠くことのでき

[8] 詳細は [Yamamoto 1990] を参照されたい。ただしこの節での内容はそれより新しい版に対応している。

ないものであり、それは、法的推論の結果の質を決定する。現在の TRIAL の主要な目的は、この分野での *Quixote* の可能性を調査することと、プロトタイプ・システムを開発することなので、われわれは、小さなターゲットにのみ焦点を絞った。すなわち、どの範囲まで、ひとつの新しい事案のために解釈ルールが抽象化されるべきかということである。それはもっともらしい説明を伴った回答を得るためであって、一般的な抽象化機構を得るためではない。

4.2.2 判例データベース上の TRIAL

TRIAL のすべてのデータと知識は、*Quixote* で記述されている。このシステムは、KL1 で書かれ、*Quixote* 上に構築されている。図 13 に、全体的な構成を示す。こ

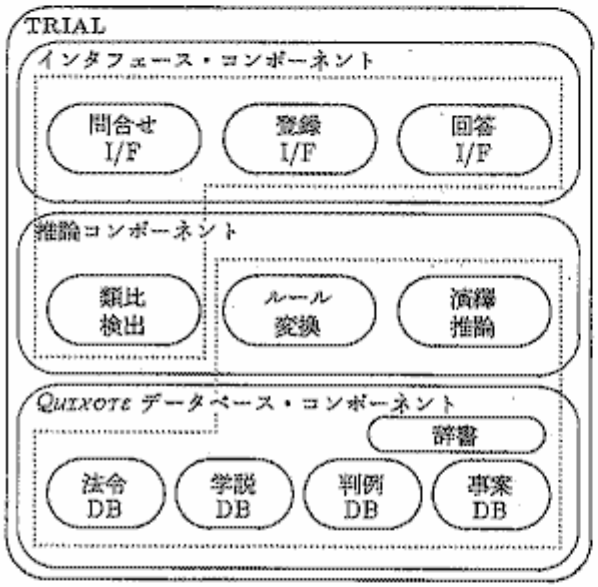


図 13: TRIAL のシステム構成

の図の中で、*Quixote* がサポートしているのは、データベース・コンポーネントの他に、基本的な機能としてルール変換と演繹推論の機能である。一方、TRIAL は、インタフェース・コンポーネントに加え、類比検出の機能をサポートしている。

類比検出について論じるために、"過労死" に関連する、以下の簡素化した例 (事案) を考える。

> 運転手のメアリは、"S" という会社に雇用されていた。仕事の合間に仮眠をとっていた際、心臓麻痺により死亡した。この事件に労働法が適用されうるか?

これは、下記のように *Quixote* の、モジュール "事案" として表現される。

```
事案::{{事案 /[対象者=メアリ,
              状況=仮眠,
              結果=心臓麻痺];;
       関係 [形態=雇用, 従業員=メアリ]
            /[所属=組織 [名前="S"],
              職種→運転手]}}
```

ここで ";;" はルール間の区切り文字である。このモジュールは、事案データベースに格納される。ここでは、業務遂行性と業務起因性の 2 つの抽象化された判例があると仮定する[9]。

```
事件1 :: 判決[事件=X]/[判決→業務起因性]
          ⇐関係 [状態=Y, 従業員=Z]/[原因=X]
           ||{X ⊑parm. 事件,
              Y ⊑parm. 状態,
              Z ⊑parm. 従業員 };;
事件2 :: 判決 [事件=X]/[判決→業務遂行性]
          ⇐ X/[状況 = Y, 結果 = Z],
            Y ⊑職種
           ||{X ⊑parm. 事件,
              Y ⊑parm. 状況,
              Z ⊑parm. 結果 }.
```

両方のルールの中の、変数 X, Y そして Z がオブジェクト parm のプロパティによって制限されていることに注目されたい。すなわち、これらは既に parm によって抽象化されていて、これらの抽象化の程度は、parm のプロパティによって制御されている。このような判例が、類比検出によって判例データベースから検索され、またルール変換によって抽象化されている。下記のような、(法令データベースの中の) 労働法と (学説データベースの中の) 学説を考察しなければならない。

```
労働法:: 組織[名前=X]/[責任→保証 [対象=Y,
                                金銭=給与分]]
          ⇐判決 [事件→事件]
               /[対象者=Y,
                 結果→病気,
                 判決→保険],
            関係 [状態=Z, 従業員=Y]
                 /[所属=組織 [名前=X]].

学説:: 判決 [事件=X]/[判決→保険]
        ⇐判決 [事件=X]/[判決→業務起因性],
          判決 [事件=X]/[判決→業務遂行性]
          ||{X ⊑ 事件 }.
```

さらにまた、下記の parm オブジェクトを定義しなければならない。

```
parm :: parm/[事件 = 事件,
              状態 = 関係,
              状況 = 職種,
              結果 = 病気,
              従業員 = 人].
```

[9]本稿では、ルール変換の段階は省略し、抽象的解釈ルールは与えられているものと仮定している。

事件$_1$ と事件$_2$ に parm を使うために、サブモジュール関係を下記のように定義する。

$$parm \sqsupseteq_S \text{事件}_1 \cup \text{事件}_2.$$

この情報は、ルール変換の際に動的に定義される。その上に、包摂関係を定義しなければならない。

事件	$\sqsupseteq$	事案
関係	$\sqsupseteq$	雇用
病気	$\sqsupseteq$	心臓麻痺
職種	$\sqsupseteq$	仮眠
人	$\sqsupseteq$	メアリ
業務起因性	$\sqsupseteq$	保険
業務遂行性	$\sqsupseteq$	保険

このような定義は、辞書の中に前もって格納される。

それから、上記のデータベースに対して仮説を伴う質問をすることができる。

1) もし“事案”が“*parm*”と“学説”を継承したら、どのような種類の判決を受けとることができるか？

?-事案: 判決 [事件＝事案]/[判決＝X]
if 事案 $\sqsupseteq_S$ *parm*∪ 学説.

3つの回答が得られる。

- X＝業務遂行性
- もし“事案:判決 [事件＝事案]”が、“判決$\sqsubseteq$業務起因性”というプロパティを持っていたら、“$X \sqsubseteq$保険”
- もし“事案:関係 [状態＝雇用, 従業員＝メアリ]”が、“原因＝事案”というプロパティを持っていたら、“$X \sqsubseteq$保険”

これらのうち2つは仮定を含む回答である。

2) もし“事案”が“労働法”と“*parm*”を継承すれば、メアリが所属している組織はどのような責任を負うべきか?

?-事案: 組織 [名前＝“S”]/[責任＝X]
if 事案 $\sqsupseteq_S$ *parm*∪ 労働法.

次に2つの回答を得る。

- もし、“事案:判決 [事件＝事案]”が“判決$\sqsubseteq$業務起因性”というプロパティを持っていたら、“$X \sqsubseteq$保証 [対象＝メアリ, 金銭＝給与分]”
- もし“事案:関係 [状態＝雇用, 従業員＝メアリ]”が“原因＝事案”というプロパティを持っていたら、“$X \sqsubseteq$保証 [対象＝メアリ, 金銭＝給与分]”

類比検出のために、parm オブジェクトは、

- “事件$_1$”と“事件$_2$”にあるように、いかにルールを抽象化するか、
- どのプロパティを parm の中で抽象化するか、そして、
- parm のプロパティの中にどんな値をセットするか

を決定する上で本質的な役割を果たしている。TRIAL において、われわれはそのような抽象化を実験することができた。つまり、*Quixote* での類比検出である。

TRIAL のユーザ・インタフェースのために、*Quixote* は、もし必要ならば、回答と共に説明 (導出グラフ) を返している。TRIAL のインタフェースは、ユーザの要求に従ってこれをグラフとして表示する。仮定と対応する説明の妥当性から回答を判断することによって、ユーザはデータベースの更新や抽象化の戦略を変更することができる。

4.3 自然言語の時制推論

時間的な情報は、自然言語処理の中で重要な役割を果たす。しかしながら、自然言語の時間軸は、自然科学のように均質ではなく、心的な事象と関連しており、一部が縮小したり伸長したりしている。さらに、その相対性は観察者の視点に依存して異なっている。この研究の目的は、それぞれの語彙的な項目から抽出された時間情報を結合し、語が持っている時間的曖昧性を解決するための推論機構の枠組を示すことである[10]。

4.3.1 自然言語における時間情報

同じ現実の状況に対して異なった表現をすることがしばしばある。たとえば

ドン・キホーテは風車を攻撃する。
ドン・キホーテは風車を攻撃した。
ドン・キホーテは風車を攻撃している。
⋮

このような異なった表現は、時制と様相に関連している。これらの間の関係を、どのように記述すべきだろうか？

状況理論に従って、状況 s とインフォン (情報子) σ の間のサポート関係を下記のように書く。

$$s \models \sigma.$$

たとえば、もし上記の例のひとつが状況 s でサポートされていれば、それは下記のように書く。

$$s \models \ll \text{攻撃する}, \text{ドン・キホーテ}, \text{風車} \gg,$$

ここでは、“攻撃する”は関係で、“ドン・キホーテ”と“風車”はパラメータである。しかし厳密に言えば、このような関係は視点 $\mathcal{P}$ から切りとられたもので、下記のように書くべきだろう。

$$s \models \sigma \iff \mathcal{P}(s' \models \sigma').$$

[10]詳細は [Tojo and Yasukawa 1992] を参照されたい。

このような関係の視点を入れ子にすることもあるかもしれないが、ここではある自己反映的な性質を仮定する。

$$\mathcal{P}(s' \models \sigma') \implies \mathcal{P}(s')\mathcal{P}(\models)\mathcal{P}(\sigma').$$

時間の観点から、$\mathcal{P}(s')$ と $\mathcal{P}(\sigma)$ をどのように表現するかを考察するために、時点の集合間に半順序関係を導入する。時点の集合が $\preceq$ によって半順序関係にあると仮定すると、集合 T_1 と T_2 の間に下記のように $\preceq_t$ と $\subseteq$ を定義することができる。

$$T_1 \preceq_t T_2 \stackrel{def}{=} \forall t_1 \in T_1, \forall t_2 \in T_2.\ t_1 \preceq t_2.$$
$$T_1 \subseteq T_2 \stackrel{def}{=} \forall t_1 \in T_1.\ t_1 \in T_2.$$

混乱がなければ、添え字の t は省略する。

時制と様相をより明確にするために、下記の概念を導入する。

1) 発話状況 u と記述状況 s の区別。
2) 状況とインフォンの持続 (始点と終点によって決定される線形の時点の集合)。 T の持続を $\| T \|_t$ と書く。

発話状況、記述状況、インフォンの3つの持続の間の関連性を図14に見ることができる。ここでは、$\| s_1 \|_t \preceq \| s_2 \|_t$

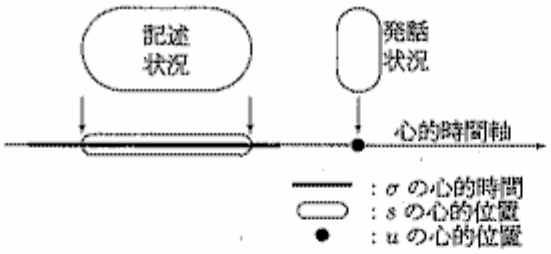


図 14: 3つの持続の関係

の代わりに $s_1 \preceq s_2$、$\| s_1 \|_t \subseteq \| s_2 \|_t$ の代わりに $s_1 \subseteq s_2$ という簡単な記法を使うことにする。

上記の定義によって、$s \models \sigma$ のとき時制と様相を下記のように定義できる ($\mathcal{P}(\models)$ は $\models$ と書く)。

$$s[s \preceq u] \models \ll 過去, \sigma \gg .$$
$$s[s \supseteq u] \models \ll 現在, \sigma \gg .$$
$$s[s \subseteq u] \models \ll 進行, \sigma \gg .$$
$$s[\sigma \preceq u] \models \ll 完了, \sigma \gg .$$

s は記述状況、u は発話状況、そして σ はインフォンである。$s[C]$ の C は制約で、視点を意味する。上記の規則は、*Quixote* で書かれた時制推論のためのルール (もしくは公理) として組み込まれている。

4.3.2 *Quixote* の時制推論

状況推論のためのルールを下記のように定義する。

$$s \models \sigma \Leftarrow s_1 \models \sigma_1, \cdots, s_n \models \sigma_n.$$

ここで、$s, s_1, \cdots, s_n$ は視点を伴う状況である。このルールは、$s_1 \models \sigma_1$、$\cdots$、かつ $s_n \models \sigma_n$ ならば、$s \models \sigma$ であることを意味している。このようなルールは、視点を伴った状況をモジュールへ、インフォンを o-項へ、そして持続間の半順序は包摂関係へと関連させることによって、*Quixote* のサブクラスへ容易に変換することができる。しかしここにはひとつの制限がある。つまりルールのヘッドの中の制約は、o-項の間の包摂関係を含むことはできない。なぜならば、このような関係は包摂の束を破壊するかも知れないからである。

言語化された (情報を表す) インフォンは、下記のようにo-項として表現される[11]。

$$inf[\text{動詞関係} = [\text{動詞} = R, \text{分類} = CLS, \text{時制状況} = P], \text{引数} = Args]$$

たとえば、"John is running" は下記のように書かれる。

$$inf[\text{動詞関係} = [\text{動詞} = run, \text{分類} = act_2, \text{時制状況} = [fov = ip, pov = pres]], \text{引数} = [\text{動作主} = john]].$$

すなわち、動作主は $john$、動詞は run で、それは、act_2 (進行状態もしくは結果状態) に分類され、その視点は (図14 の丸い枠の) 視野としての進行状態に、(図14 の黒丸の) 観点としての現在である。

そのような言語化したインフォンをサポートする談話状況は下記のように表現される。

$$\text{談話状況}[fov = ip, pov = pres, src = U],$$

ここで、最初の2つの引数は、上記のインフォンの "時制状況" と同じで、3番目の引数は発話状況である。

この書き換えにしたがって、表現の中の時間情報の曖昧性を減少することを可能にする、小さな例を示すことができる。

1) 表現 E が与えられると、それぞれの形態素は時間情報を調べるために処理される。

$$mi[u = U, \text{表現} = [\,], e = D, infon = Infon].$$
$$mi[u = U, \text{表現} = [Exp|R], e = D, infon = Infon] \Leftarrow d\_cont[\text{表現} = Exp, sit = D, infon = Infon], mi[u = U, \text{表現} = R, e = D, infon = Infon].$$

[11] o-項 $\top[l_1 = o_1, \cdots, l_n = o_n]$ は $[l_1 = o_1, \cdots, l_n = o_n]$ と略記する。

それぞれの形態素の時間情報は D で共通部分が取られる。すなわち、曖昧性は徐々に減少する。

2) 談話状況と言語化したインフォンの組の時間情報は、下記のルールによって定義される。

d_cont[表現 $= Exp$,
$sit =$ 談話状況 $[fov = Fov, pov = Pov, src = U]$
$infon = inf$[動詞関係 $= V\_rel$, 引数 $= Args$]]
⇐辞書 : v[分類 $= CLS$, 動詞 $= R$, 日本語 $= Exp$]
||{V_rel =[動詞 $= R$, 分類 $= CLS$, 時制状況 $= P$]}

d_cont[表現 $= Exp$,
$sit =$ 談話状況 $[fov = Fov, pov = Pov, src = U]$
$infon = inf$[動詞関係 $= V\_rel$, 引数 $= Args$]]
⇐辞書: 助動詞 [$asp = ASP$, 日本語 $= Exp$],
写像 [分類 $= CLS$, $asp = ASP$, $fov = Fov$]
||{V_rel =[動詞 $= \_$, 分類 $= CLS$, 時制状況 $= P$],
$P = [fov = Fov, pov = \_\}$;;

d_cont[表現 $= Exp$,
$sit =$ 談話状況 $[fov = Fov, pov = Pov, src = U]$
$infon = inf$[動詞関係 $= V\_rel$, 引数 $= Args$]]
⇐辞書: 接尾辞 [$pov = Pov$, 日本語 $= ru$]
||{V_rel =[動詞 $= \_$, 分類 $= \_$, 時制状況 $= P$],
$P = [fov = \_, pov = Pov]$}

3) 下記のように語彙情報を定義しているモジュール“辞書”がある。

辞書::{{
v[分類 $= act_1$, 動詞 $= put\_on$, 日本語 $=$ き];;
v[分類 $= act_2$, 動詞 $= run$, 日本語 $=$ はし];;
v[分類 $= act_3$, 動詞 $= understand$, 日本語 $=$ わか];;
助動詞 [$asp = state$, 日本語 $=$ てい];;
接尾辞 [$pov = pres$, 日本語 $=$ る];;
接尾辞 [$pov = past$, 日本語 $=$ る]}}

さらに、視野の写像もまた下記のように (大城的) ファクトの集合として定義される。

写像 [分類 $= act1, asp = state, fov = \{ip, tar, res\}$].
写像 [分類 $= act2, asp = state, fov = \{ip, res\}$].
写像 [分類 $= act3, asp = state, fov = \{tar, res\}$].

もし問合せの中にいくつかの日本語表現が与えられると、対応する時間情報が上記のプログラムによって返される。

5 より柔軟なシステムを目指して

DOOD システムを拡張するために、より柔軟な実行制御のための別のアプローチをとっている。その例として、主に自然言語の応用に焦点を絞っている。

5.1 制約変換

自然言語の文法理論は多く存在する。たとえば、GB のような制約ベースの変形文法、GPSG と LFG のようなルールベースの単一化文法、そして、HPSG と JPSG のような制約ベースの単一化文法などである。文法のより一般的な枠組を考えると、形態素、構文、意味論、そして、語用論が制約として統一的に扱えるという点で、HPSG と JPSG がより良いものと考えられる。このような観点から、新しい制約論理プログラミング (CLP) 言語 cu-Prolog を開発し、それを使って JPSG (Japanese Phrase Structure Grammar: 日本語句構造文法) パーザを実装した[12]。

5.1.1 単一化文法における制約

まず最初に、制約ベースの文法における様々な制約について考察してみよう。

- 基本的な情報構造として用いられる選言的素性構造は、非正規関係の組や複合オブジェクトのように、下記のように定義される。
  1) 素性構造は、ラベルと値の対で構成される組 $[l_1 = v_1, \cdots, l_n = v_n]$ である。
  2) 値はアトムか素性構造、もしくは素性構造の集合 $\{f_1, \cdots, f_n\}$ である。
- JPSG においては、文法規則は図 15 のように、二分木の形式で記述され、それぞれのノードは素性構造になっている。この中で、特定の素性 (属性) によって、

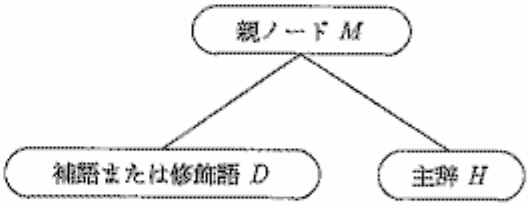


図 15: JPSG の句構造

D が補語なのか修飾語なのかが決定される。構造原理と呼ばれるそれぞれの文法は、局所的句構造木の中の、M と D と H の 3 つの素性間の制約として表現されることに注意されたい。

上記の定義に見られるように、素性構造は DOOD のデータ構造にとてもよく類似している[13]。cu-Prolog を始めとする自然言語処理の経験を通して、DOOD システム上に自然言語処理を構築するための要求を見出しアイデアを得たので、今後それらを、DOOD システム上の応用システムに反映させることができるだろう。

[12]詳細は [Tsuda 1992] を参照されたい。

[13]このことが *QUIXOTE* の設計を決定した理由のひとつである。詳細は付録を参照されたい。

5.1.2 cu-Prolog

素性構造を効率的に処理するために、cu-Prolog と呼ばれる新しい CLP を開発した。ルールは、下記のように定義される[14]。

$$H \Leftarrow B_1, \cdots, B_n || \{C_1, \cdots, C_m\}.$$

ここで、$H, B_1, \cdots, B_n$ は原子式で、その引数は素性構造の形をしている。$C_1, \cdots, C_m$ は素性構造や変数、アトムの間の等式、あるいは他のルールの集合によって定義された原子式である。制約解消の合流性を保証するために、制約中の原子式にはモジュラーという制限がある。これは、静的にチェックすることができる。意味論の定義域は CIL と同じように部分的なタグ付き木の関係の集合であり、制約の定義域もまた CIL に同じである。

cu-Prolog の導出は、従来の CLP と同じように、制約の集合とサブゴールの集合の対の列である。これらの相違点を下記に示す。

- 述語中のすべての引数は素性構造となることができる。すなわち、素性構造間の単一化が必要である。
- 計算ルールは、制約解消の失敗したルールを選択しない。つまり、$(\{A\} \cup G, C)$、$A' \Leftarrow B||C'$、および $A\theta = A'\theta$ の場合には、制約 $C\theta \cup C'\theta$ が閉約されないならばそのルールは選択されない。
- 制約解消系は、unfold/fold 変換に基づいていて、制約部に新述語を動的に作り出す。

cu-Prolog の素性構造の「還言」は、ちょうど *Quixote* における o-項や Kappa (CRL) の非正規組のように、基本的に「連言」として取り扱われる。しかしながら、述語の存在のために、還言は新しい制約とファクトを導入することによって解消 (もしくはアンネスト) できる。

$$H \Leftarrow p([l=\{a,b\}]) \iff H \Leftarrow p([l=X])||\{new\_p(X)\}.$$
$$new\_p(a).$$
$$new\_p(b).$$

すなわち、cu-Prolog においては還言的素性構造は、CRL におけるような集合の単一化を回避するために、OR-並列で処理される。効率は全解を求めようとするか否かという点に依存している。

cu-Prolog のきわだった特徴のひとつは、問合せ処理中の動的な unfold/fold 変換であり、これは、問合せ処理の効率性を改善するために大いに貢献している。cu-Prolog で書いた JPSG パーザのいくつかの例は [Tsuda 1992] を参照されたい。述語を基にした表記法は本質的ではないので、cu-Prolog の言語の特徴は *Quixote* に書き換えることができ、またその制約解消系も、意味論を変えることなしに、*Quixote* の実装に埋め込むことができる。

[14]ここでは *Quixote* の構文に従っているので、以下の記法は cu-Prolog のものとは異なっている。

5.2 力学プログラミング

この研究の目的は、計算機科学から認知科学まですべてにわたって、制約の枠組を拡張することである[15]。ある意味では、この考えは、cu-Prolog の制約の取り扱いの中から生じたものであると言える。ここでは、制約処理の一般的な枠組としての力学プログラミングの概要と、自然言語処理における例を述べる。

5.2.1 記号システムの力学

すでに、2節で説明したように、部分情報は知識情報処理システムの中で本質的な役割を果たしている。したがって、部分性をどのように扱うかを知ることは今後の記号システムに欠かすことのできないことである。そのために、情報の流れから独立した制約システムを採用する。従来の論理よりも計算的により扱いやすいシステムを作るために、制約の力学を仮定し、システムの振舞をポテンシャル・エネルギーに基づいてとらえる。

節形式の下記のプログラムを考える。

$$p(X) \Leftarrow r(X,Y), p(Y).$$
$$r(X,Y) \Leftarrow q(X).$$

問合せ $?\text{-}p(A), q(B)$ が与えられたとき、演繹データベースでよく使われるルール・ゴール・グラフは、図16のようにトップ・ダウン計算を模擬する。このグラフはトップダウン計算やボトムアップ計算のような一定の情報の流れをあらかじめ仮定している。そこでもっと一般的に、図17の形式でこれを考える。図の中で、線は変数間の (部分的

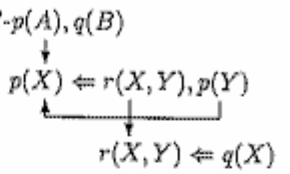


図 16: ルール・ゴール・グラフ

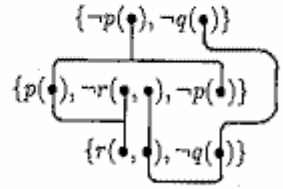


図 17: 制約ネットワーク

[15]詳細は [Hasida 1992] を参照されたい。

な) 等式を表現しており、変数間の相違は簡単のために書いていない。このようなグラフを制約ネットワークと呼ぶ。

この枠組においては、制約ネットワーク上でノード (変数もしくは要素制約) から他のノードへの制約伝播によって計算が行われる。このような計算を可能にするための制約の力学について、概略を下記に示す。

1) 活性値が、それぞれの要素制約 (原子式もしくは等式) に割当てられる。この値は、0 と 1 の間の実数で、制約の真理値とみなされる。

2) 活性値を基にして、それぞれの要素制約に対しては正規化エネルギーが、それぞれの節に対しては演繹エネルギーと発想エネルギーが、そして可能な単一化に対しては完備化エネルギーが定義される。ポテンシャルエネルギー U はこれらエネルギーの和である。

3) もし、制約の現在の状態がユークリッド空間の点 X で表現されるならば、U は点 X に関する力の場 F として定義される。$F \neq 0$ のときは、F は活性拡散を生じさせる。X の変化は制約ネットワークの隣接した部分へ伝播され、U が減少する。最終的には、活性値の割当ては、$F = 0$ を充足する安定平衡状態へ収束する。

記号計算もまた、同じ力学の下で制御される。この計算の枠組は、ホーン節には限定されない。

5.2.2 自然言語処理の統合アーキテクチャ

伝統的な自然言語処理システムは、典型的には、構文解析、意味解析、語用論解析、言語学外推論、文生成プランニング、表層文生成、という順序で行われる。しかし、構文解析は、意味的、語用論的理解に必ずしも先行するものではないし、文生成プランニングは表層文生成と深く関連している。統合アーキテクチャは、このような固定された情報の流れを改善するためのものである。われわれの制約の力学は、このようなアーキテクチャに適している。

以下の例を考えてみよう。

Tom took a telescope. He saw a girl with it.

he と it は、それぞれ Tom と telescope と照応するとみなすことができるが、with it には曖昧性がある。つまりトムが少女を見たとき、望遠鏡をトムが持っていたという解釈と、少女が持っていたと言う解釈がある。

以下のようなファクトの集合を考える。

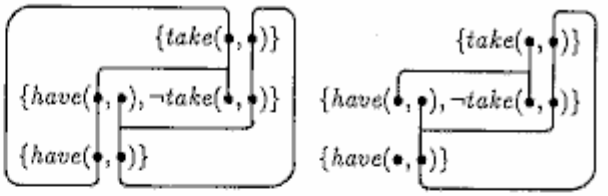
(1) $take(tom, telescope).$

(2) $have(tom, telescope).$

(3) $have(girl, telescope).$

推論規則として下記があるとする。

(4) $have(X, Y) \Leftarrow take(X, Y).$



(1),(2),(4) の制約ネットワーク　(1),(3),(4) の制約ネットワーク

図 18: 2 つの制約ネットワーク

図 18 にあるように、(1),(2),(4) と (1),(3),(4) の制約ネットワークを考えれば、右のネットワーク ((1),(3),(4)) の中には、ただひとつの巡回路 (girl) のみが存在するのに対して、左のネットワーク ((1),(2),(4)) の中には、(tom と telescope を含んだ) 2 つの巡回路があることがわかる。ポテンシャル・エネルギーの観点からは、左のネットワークは右のものよりより強力に活性化しやすい。言い換えれば、(2) は (3) よりももっともらしい。

自然言語処理では曖昧性の解消がキーポイントであるが、従来のアーキテクチャでは見込みがない。しかし、制約ネットワークの力学をベースにしたわれわれの統合アーキテクチャは、そのような応用のためばかりでなく、知識ベース管理システムにとっても、より多くの可能性を与えるものと思われる。

6 関連の研究

DOOD に基づいた、われわれの知識ベース管理システムは、他のシステムに比べて、概念、規模、そして多様性において、多くのきわだった特徴を持っている。このシステムが目指すのは、新しいパラダイムを提唱することだけではなく、データベースと知識ベースの機能を、実際に多くの知識情報処理システムに提供することである。

DOOD の概念に関連して、論理プログラミングの中にオブジェクト指向の概念を埋め込むための多くの研究がある。その中でも F-logic [Kifer and Lausen 1989] は最も豊富な概念を持っているが、オブジェクト識別のための id-項は、述語に基づいた記法であり、プロパティの表現は制約の観点からは不充分なものである。さらにまた、更新の機能やモジュールの概念が欠如している。*Quixote* は、F-logic に比べ、より多くの機能を持っている。ある意味では *Quixote* は機能を持ち過ぎている言語であるかもしれないが、ユーザは *Quixote* の任意のサブクラスを選択することが可能である。たとえば、もしユーザがオブジェクト項のサブクラスだけを使用する時には、Prolog の単純な拡張として、そのサブ言語を意識するだけで良い。

非正規関係モデルに関しては、1977 年に提唱されて以来多くの研究があり、いくつかのプロトタイプ・システムがすでに実装されている。その中には Verso [Verso 1986]、DASDBS [Schek and Weikum 1986]、さらには AIM-P

[Dadam *et al.* 1986] がある。しかし、Kappa のモデルの意味論はこれらとは異なっている。DASDBS や AIM-P の (拡張) NF^2 モデルは集合に基づいた (高階の) 意味論をとっている。それはユーザの直観にはなじみ易いものだが、問合せ能力を効率的に拡張することが大へん困難である。また一方、Verso では URSA (universal relation schema assumption) を仮定しているので、効率的な手続き的意味論を保証している。しかしながら、その意味論はユーザの直観にはなじみにくいものである。たとえば、関係 T に対して $t \notin \sigma_1 T$ かつ $t \notin \sigma_2 T$ であったとしても、$t \in \sigma_1 T \cup \sigma_2 T$ ということが生じえる。これらと比較すると、Kappa では 3 節で述べたように簡単な意味論を採っている。この意味論は効率的な計算のために、*Quixote* の o-項と、cu-Prolog の選言的素性構造でも保持されている。

遺伝子情報処理については、論理プログラミングと演繹データベースの研究者が、有望な応用としてこの分野に焦点を合わせ始めている。しかしながら、これらの研究の大部分は推移閉包のような問合せ能力やプロトタイピング能力に充てられており、データと知識表現に焦点を合わせている研究は数少ない。その一方で、*Quixote* では上記の両方の達成を目指している。法的推論に関しては、論理プログラミングとその拡張を基にして多くの研究が行われている。われわれの研究はそれらの機能をそのまま考慮に入れたものではなく、データベースの観点から、とりわけモジュールの概念を導入することによって、これらの問題を再考している。

7 今後の計画とむすび

研究開発人員と期間の不足から積み残した機能がいくつか残っている。さらに、本稿で述べた研究開発の経験を通して今後の拡張機能も考えている。

まず *Quixote* に関しては、以下の改良と拡張を考えている。

- 横方情報伝達や部分評価のような問合せ変換は、現在の実装には十分に反映できていない。このような最適化技術は *Quixote* の中に埋め込まれるべきである。ただし、制約論理プログラミングでは従来の演繹データベースとは異なった方式を必要とする。さらにまた、より効率的な問合せ処理のために、cu-Prolog や力学プログラミングのような、柔軟な制御機構が埋め込まれるべきだろう。

- *Quixote* の記述の簡便性のためにメタ機能の導入を検討している。たとえば HiLog [Chen *et al.* 1989] では以下のような記述が可能である。

$$tc(R)(X,Y) :- R(X,Y)$$
$$tc(R)(X,Y) :- tc(R)(X,Z), tc(R)(Z,Y)$$

  このような機能を提供するために、基本オブジェクトを値域とする新しい変数を導入しなければならない。この考えはさらに、*Quixote* のプラットフォーム言語にまで拡張しうる。たとえば、集合概念を導入するために、集合間の (Hoare、Smyth、あるいは Egli-Milner のような) 順序関係を決めなければならないが、これをどれにするかは、応用に依存しているように見える。さらに多くの応用のためには、このような関係はプラットフォーム言語によって定義されることが最良であろう。現在の *Quixote* は、このようなプラットフォーム言語で定義される言語族の一員と考えられる。

- *Quixote* データベース間の通信は重要な役割を果たす。それは、分散知識ベースのためだけではなく、永続的なビュー、永続的な仮説、そして局所的もしくは個人的なデータベースをサポートするためでもある。さらにまた、*Quixote* で定義されるエージェント間の協調的問合せ処理を考えることもできる。ただし、それはオブジェクト識別子の存在論に深く依存している。

- 現在の実装では、*Quixote* のオブジェクトは KL1 で定義することもできる。周知のように、あらゆる現象を単一の言語で記述するのは困難であるので、どんな言語も他の言語とのインタフェースをサポートすべきである。したがって、*Quixote* においても、多言語システムが期待されるだろう。

- DOOD の枠組の中で、われわれは主にデータ・モデルの拡張に焦点を置いていたが、その方向は、論理的拡張と計算モデルの拡張とは必ずしも直交しない。たとえば、集合のグルーピングは失敗としての否定 (negation as failure) を含むと考えることができるし、また *Quixote* の手続き的意味論は、オブジェクト指向の枠組のもとに定義することができる。

  しかしながら、人工知能の観点からは、非単調推論やファジー論理がさらに埋め込まれるべきだろうし、設計論の観点からは、オブジェクト指向のような他の意味論もまた埋め込まれるべきだろう。

Kappa に関しては、以下の改良と拡張を考えている。

- ウィスコンシン・ベンチマークによって他の DBMS と比較すると、Kappa の性能はさらに改善できそうである。特に拡張関係代数の性能は、カーネル内の通信コストを減少させることによって、はっきり改良することができる。これは、機能拡張とは独立に追及すべきであろう。

- 逐次推論マシンや並列推論マシンから Kappa をアクセスするだけではく、汎用マシンやワークステーションからも Kappa をアクセスできるようにする計画がある。さらにまた、システムの移植性と開放システム環境への適応性についても考えなければならないだろ

う。その候補のひとつとして、Kappa-P はすでにある種の分散 DBMS であるが、さらにクライアント・サーバ・モデルを基にした異種分散 DBMS へのサーバ・システムへの拡張が考えられる。

- Kappa をより多くの応用に提供するためには、他の DBMS との互換性を増大させる他に、カスタマイズ機能とサービス・ユーティリティによって応用との親和性を強化しなければならないだろう。

Kappa と *Quixote* を統合してひとつの知識ベース管理システムにするためには、さらに拡張が必要となる。

- *Quixote* が入れ子トランザクション論理をとっているのに対し、Kappa ではフラットなトランザクション論理をとっている。その結果、*Quixote* が保証できるのはトップレベルのトランザクションの永続性に限られる。これらをよりしっかりと結合するためには、Kappa が入れ子トランザクション論理をサポートしなければならない。

- 効率的な処理の観点から、ユーザは *Quixote* を通して Kappa を直接に使用することはできない。しかし、これらを独立に使用すると、オブジェクト識別性についてユーザが責任を負うことになる。これを解決するためには、Kappa にオブジェクト識別性の概念を導入し、Kappa と *Quixote* が同じオブジェクト空間を共有できるようなことを許す機構も考慮されなければならないだろう。

- Kappa-P は並列 DBMS として素直なアーキテクチャを持っているが、現在の *Quixote* は、たとえそれが KL1 によって実装され並列に稼働するとしても、並列処理に必ずしも親和性をもっていない。より効率的な処理のために、Kappa と *Quixote* の並列処理を統一的に研究しなければならない。

われわれは、この論文で述べた応用よりもさらに大きなより多用な応用を開発しなければならない。そのためにはまた、従来のシステムとの互換性を高めなければならない。たとえば、Prolog から *Quixote* への、また正規関係モデルから Kappa の非正規関係モデルへの変換などが挙げられる。

われわれは DOOD の枠組を提案し、既に述べたように、この枠組でのデータベースと知識ベースのための様々な研究開発活動に携わってきた。それぞれのテーマが、必ずしもこの枠組から生じてきたものではないが、この方向性が多くの応用のために有望であることを、われわれの経験は指し示している。

謝辞

本稿で述べたそれぞれの研究開発テーマについて ICOT 第3研究室のすべてのメンバーの助力を受けた。とくに、*Quixote* と cu-Prolog は津田宏、Kappa は河村元夫と永沼和智、生物データベースには田中秀俊と阿比留幸展、TRIAL は山本慶一郎、時制推論は東条敏、DP は橋田浩一、の各氏に感謝する。また上記の研究開発に対して、刺激的で有益な意見を頂いた DOOD 関係、STS、および JPSG のワーキング・グループの委員の方々、およびシステムの実装に努力して頂いた関連プロジェクト (付録参照) のメンバーの方々に謝意を表する。いうまでもなく、これらの研究開発は、ICOT の渕一博研究所長と内田俊一研究部長の絶えざる励ましなくしては実現できなかったものであり、両氏に感謝したい。最後に、この日本語版の準備に多大な協力をして頂いた ICOT 第3研究室の谷關子さんに感謝の意を表したい。

参考文献

[Aczel 1988] P. Aczel, *Non-Well Founded Set Theory*, CSLI Lecture Notes No. 14, 1988.

[Chen *et al.* 1989] W. Chen, M. Kifer, and D.S. Warren, "HiLog as a Platform for Database Language", *Proc. the Second Int. Workshop on Database Programming Language*, pp.121-135, Gleneden Beach, Oregon, June, 1989.

[Chikayama 1984] T. Chikayama, "Unique Features of ESP", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, Nov.6-9, 1984.

[Chikayama *et al.* 1988] T. Chikayama, H. Sato, and T. Miyazaki,"Overview of the Parallel Inference Machine Operating Sistem (PIMOS)", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, Nov.28-Dec.2, 1988.

[Dadam *et al.* 1986] P. Dadam, *et al*, "A DBMS Prototype to Support Extended NF^2 Relations: An Integrated View on Flat Tables and Hierarchies", *ACM SIGMOD Int. Conf. on Management of Data*, 1986.

[Delobel *et al.* 1991] C. Delobel, M. Kifer, and Y. Masunaga (eds.), *Deductive and Object-Oriented Databases*, (*Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases (DOOD'91)*), LNCS 566, Springer, 1991.

[Goto *et al.* 1988] A. Goto *et al.*,"Overview of the Parallel Inference Machine Architecture (PIM)", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, Nov.28-Dec.2, 1988.

[Haniuda *et al.* 1991] H. Haniuda, Y. Abiru, and N. Miyazaki, "PHI: A Deductive Database System", *Proc.*

IEEE Pacific Rim Conf. on Communication, Computers, and Signal Processing, May, 1991.

[Hasida 1992] K. Hasida, "Dynamics of Symbol Systems — An Integrated Architecture of Cognition", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[Kawamura *et al.* 1992] M. Kawamura, H. Naganuma, H. Sato, and K. Yokota, "Parallel Database Management System *Kappa-P*", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[Kifer and Lausen 1989] 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.

[Kim *et al.* 1990] W. Kim, J.-M. Nicolas, and S. Nishio (eds.), *Deductive and Object-Oriented Databases*, (*Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases (DOOD89)*), North-Holland, 1990.

[Miyazaki *et al.* 1989] N. Miyazaki, H. Haniuda, K. Yokota, and H. Itoh, "A Framework for Query Transformation", *Journal of Information Processing*, Vol. 12, No.4, 1989.

[Mukai 1988] K. Mukai, "Partially Specified Term in Logic Programming for Linguistic Analysis", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, Nov.28-Dec.2, 1988.

[Schek and Weikum 1986] H.-J. Schek and G. Weikum, "DASDBS: Concepts and Architecture of a Database System for Advanced Applications", *Tech. Univ. of Darmstadt, Technical Report*, DVSI-1986-T1, 1986.

[Tanaka 1992] H. Tanaka, "Integrated System for Protein Information Processing", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[Tojo and Yasukawa 1992] S. Tojo and H. Yasukawa, "Situated Inference of Temporal Information", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[Tsuda 1992] H. Tsuda, "cu-Prolog for Constraint-Based Grammar", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[Ueda and Chikayama 1990] K. Ueda and T. Chikayama, "Design of the Kernel Language for the Parallel Ingerence Machine", *The Computer Journal*, Vol.33, No.6, 1990.

[Verso 1986] J. Verso, "VERSO: A Data Base Machine Based on Non 1NF Relations", *INRIA Technical Report*, 523, 1986.

[Yamamoto 1990] N. Yamamoto, "TRIAL: A Legal Reasoning System (Extended Abstract)", *Joint French-Japanese Workshop on Logic Programming*, Renne, France, July, 1991.

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

[Yokota 1988] K. Yokota, "Deductive Approach for Nested Relations", *Programming of Future Generation Computers II*, eds. by K. Fuchi and L. Kott, North-Holland, 1988.

[Yokota *et al.* 1988] K. Yokota, M. Kawamura, and A. Kanaegami, "Overview of the Knowledge Base Management System (KAPPA)", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, Nov.28-Dec.2, 1988.

[Yokota and Nishio 1989] K. Yokota and S. Nishio, "Towards Integration of Deductive Databases and Object-Oriented Databases — A Limited Survey", *Proc. Advanced Database System Symposium*, Kyoto, Dec., 1989.

[Yoshida 1991] K. Yoshida, "The Design Principle of the Human Chromosome 21 Mapping Knowledgebase (Version CSH91)", *Inetrnal Technical Report of Lawrence Berkley Laboratory*, May, 1991.

付録

データベースと知識ベースのプロジェクトに関する註記

この付録では、FGCS プロジェクトでのデータベース管理システムと知識ベース管理システムのプロジェクトの概要を説明しよう。簡単な歴史を図19に示す[16]。これらのプロジェクトの中で、三菱電機は Kappa-I、Kappa-II、Kappa-P、DO-ι、CIL、そして *Quixote* プロジェクトに、沖電気工業は PHI (DO-ϕ) と *Quixote* プロジェクトに、そして日立製作所は ETA (DO-η) と *Quixote* プロジェクトに携わってきた。

a. Kappa プロジェクト

知識情報処理システムにデータベース機能を提供するために、Kappa[17]プロジェクトは (FGCS プロジェクトの中期の始まりの数カ月後の) 1985年9月に始まった。最初のターゲットは、自然言語処理のための (概念辞書を含む) 電子化辞書のデータベースや、CAP-LA と呼ばれる証明検証システムの数学的知識のデータベースを構築することだった。前者は特に重要で、それぞれの辞書は数10万語のエントリを含み、それぞれのエントリは複雑なデータ構造をしていた。したがって、正規関係モデルではそのようなデータに対処できないと判断し、非正規関係モデルを採用することを決断した。さらに数学の知識を扱うために新しい型として項を追加することにした。その DBMS は ESP で記述され、PSI と SIMPOS の環境で稼働する必要があった。ESP で書いたシステムが効率的に動作するかどうか不安があったので、まず非正規関係の意味論を決め、Kappa-I と呼ばれるプロトタイプ・システムを作ることにした。ESP で6万行のこのシステムは1987年の春に完成し、大量の辞書データに対して効率的に動作することが確認された。このプロジェクトは、処理効率の必要な測定を行い、1987年8月に終了した。

PSI 上に効率的な DBMS を構築しうるという見通しを得た後、Kappa-II [Yokota *et al.* 1988] という次のプロジェクトを1987年4月に立ち上げた。これは非正規関係モデルに基づく実用的な DBMS を目指したものであった。Kappa-I に比べより良い処理効率という目的の他に、いくつかの改善を計画した。つまり、主記憶データベース機能、拡張関係代数、ユーザ定義コマンド、ユーザ指向のウィンドウ・インタフェースなどである。ESP で18万行のこのシステムは、PSI-I 上の Kappa-I に比べて、PSI-II 上の Kappa-II は10倍以上効率良く動いた。このプロジェクトは1989年3月に終了し、システムは国内のみならず海外の研究機関にも広くリースされ、それらは主として遺伝子情報処理のために使用された。

より大量のデータを効率的に処理するために、Kappa-P [Kawamura *et al.* 1992] と呼ぶ並列 DBMS のプロジェクトが1989年2月に始まった。このシステムは KL1 で書かれており、PIM と PIMOS の環境で稼働する。Kappa-P のそれぞれのローカル DBMS は単一プロセッサ上で、Kappa-II とほぼ同程度の処理効率を発揮するので、システム全体としては、環境は異なっているのだが、Kappa-II 以上の処理効率で動作することが期待されている。

b. 演繹データベース・プロジェクト

演繹データベースのために3つのプロジェクトがあった。

まず、Kappa の開発と並行して CRL (complex record language) [Yokota 1988] と呼ぶ演繹データベース・プロジェクトが始まった。CRL は非正規関係を取り扱うために新しく設計された論理プログラミング言語である。

CRL は、集合構成子と組構成子から構成される複合オブジェクトのサブクラスに基づいており、モジュール概念を持っている。DO-ι と呼ばれるこのプロジェクトは1988年の夏に始まり、1989年12月に終了した。このシステムは Kappa-II 上で稼働した。問合せ処理の戦略は、一般マジック集合法とセミナイーブ評価法に基づいている。そしてサブモジュール関係に基づいたモジュール間のルール継承が、動的に行われている。

次に、PHI[Haniuda *et al.* 1991] と呼ばれるプロジェクトが中期 (1985年4月) から始まった。これは、伝統的な演繹データベースにおけるより効率的な問合せ処理を目指したものであった。その戦略は HCT (Horn clause transformation) [Miyazaki *et al.* 1989] と呼ぶ3種類の問合せ変換に基づいている。つまり、部分評価あるいは unfolding に基づいた HCT/P、ルール変換しないで束縛伝播を行う HCT/S、そして、探索空間を狭めるためにルールの集合を変換し関連する新しいルールを追加する HCT/R である。HCT/R は一般マジック集合法に対応しているものである。これらの戦略を組合せることによって PHI はより効率的な問合せ処理を目指した。これに続くプロジェクトは DO-ϕ と呼ばれ、複合オブジェクトのための演繹機構の研究開発を目指した。

第3に、ETA と呼ぶプロジェクトが1988年4月に始まった。これは意味ネットワークのような知識表現に基づく知識ベースを目指した。一年後このプロジェクトは演繹データベースの拡張の方向に方針を転換し、DO-η と呼ばれた。

これらプロジェクトの "DO" は deductive and object-oriented databases の略であり、これらすべてが共通の枠組として DOOD [Yokota and Nishio 1989] を採用したことを示している。

---

[16] FGCS プロジェクトの前期においても、データベースと知識ベースのためのいくつかのプロジェクトがあった。たとえば Delta と Kaiser がある。しかし、それらは中期において別の目的のプロジェクトに変わった。

[17] Kappa は *knowledge application oriented advanced database management system* の略である。

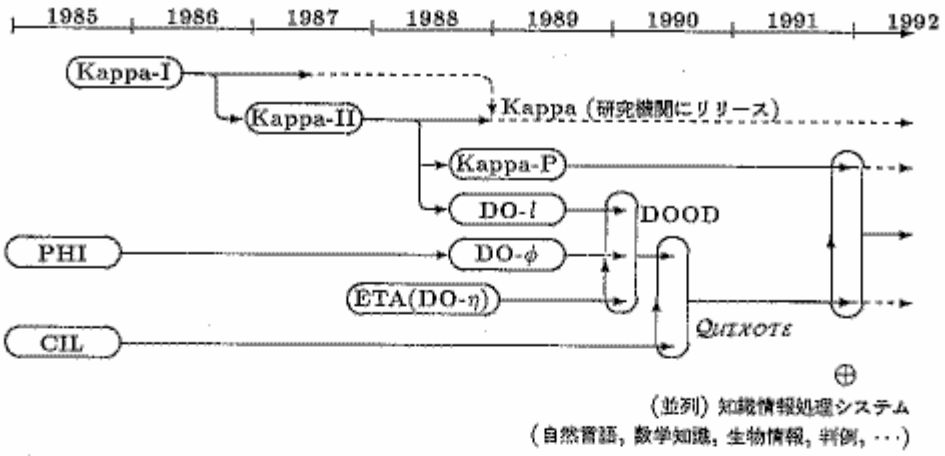


図 19: データベースと知識ベースのプロジェクトの簡単な歴史

c. CIL プロジェクト

CIL (complex indeterminates language) [Mukai 1988] と呼ばれる言語が 1985 年 4 月に提案された。この言語は自然言語処理における意味表現を目指しており、DUALS と呼ばれる談話理解システムで使われただけでなく、さまざまな言語情報の表現にも利用された。CIL の実装は何回にも渡って改善され、自然言語処理分野の多くの研究者にリリースされた。この言語は制約論理プログラミングの一種であり、状況理論・状況意味論に密接に関連している。この言語は、組構成子から構成される部分指定項 (partially specified term) に基づいている。5.1 節で述べた cu-Prolog は遍言的素性構造を表現するために、この部分指定項に集合構成子を導入している。

d. *QUIXOTE* プロジェクト

CRL を非正規関係だけでなく DOOD にも拡張しようとする一方で、CIL を遍言的素性構造のようなより効率的な表現に拡張しようと試みていた。それらの多くの試みと努力の後で、CRL の拡張として Juan 、CIL の拡張として *QUINT* という 2 つの言語を提案した。しかし、これらの言語設計の途中で、Juan と *QUINT* の間の、そしてデータベースと自然言語処理の概念の間の多くの類似性に気づいたため、これらの言語を統合することにした。この統合された言語が (スペイン語風の発音での) *QUIXOTE*[Yasukawa *et al.* 1992] である[18]。この統合の結果、*QUIXOTE* は、すでに述べたように、多くの特徴を持つことになった。*QUIXOTE* プロジェクトは 1990 年の 8 月に始まった。*QUIXOTE* の第 1 版は 1991 年の 8 月に限られたユーザにリリースされ、第 2 版は 1992 年の 4 月により多く応用にリリースされた。どちらも KL1 で書かれており、PIM 上で稼働する。

e. DOOD と STASS に関するワーキング・グループ

1987 年の終わりに、データベース分野での論理とオブジェクト指向の概念の統合を考え始めた。多くの研究者との議論の後、DOOD のためのワーキング・グループを組織し、DOOD のための新しい国際会議の準備を始めた[19]。このワーキング・グループは 1990 年には、データベース・プログラミング言語 (DBPL)、演繹データベースと人工知能 (DDB&AI)、拡張項 (ETR)、生物データベース (BioDB) の 4 つのサブワーキング・グループを組織した。1991 年になると、このワーキング・グループは知的データベース (IDB) と次世代データベース (NDB) の 2 つに分割された。これらの定期的な会合[20]で、DOOD の問題だけでなく、次世代データベースの方向性や問題点についても、幅広い議論を行ってきた。この議論はわれわれの DOOD システムに大いに寄与してきた。

また別の観点から、1990 年に状況理論と状況意味論のために STS と呼ばれるワーキング・グループ[21]を組織した。このワーキング・グループも *QUIXOTE* とその応用の他の側面の強化に大いに寄与した。

[18] この名前の由来は、Don Juan や Don Quixote のような DON シリーズから来ている。DON とは Deductive Object-Oriented Nucleus の略である。

[19] この準備の大部分は、第一回会議 (DOOD89) まで大阪大学の西尾助教授によって行われた。

[20] これらワーキング・グループの主査は、DOOD が田中譲教授 (北大)、DBPL が田中克己助教授 (神大)、DDB&AI と IDB が坂間千秋氏 (ASTEM)、ETR が西尾章治郎助教授 (阪大)、BioDB が小長谷明彦氏 (NEC)、NDB が吉川正俊助教授 (京産大) であった。

[21] この主査は田中穂積教授 (東工大) であった。