解　説

データベースとProlog†

伊草ひとみ††　鈴木克志††
佐藤裕幸††　太細孝††

1. はじめに

Prolog[20),21)]は，近年の知識情報処理技術の発展，及び，第5世代コンピュータプロジェクトの進展にともない，急速に注目を集めるようになり，プログラミング言語としての諸機能，基礎となる理論体系を始め，知識情報処理に関連する諸技術への応用方法や適性評価等，種々の観点から研究・開発・応用が行われている．

本稿では，Prologが，プログラミング言語であると同時に，演繹能力を持つ関係データベース処理言語ともみなせることから，関係データベースとPrologとの関連を取り上げる．以下，まず2章で，Prologによるデータベース実現について，関係データベースとの比較，特徴等を述べる．次に，3章，4章で，データ検索に関する話題として，データベース問合せの効率化手法と，データの大規模化を背景とした問合せの評価方法について解説する．

2. Prologによるデータベースの実現

Prologは，なぜ関係データベース処理言語とみなせるのか，また，その演繹能力とはどのようなものか[2),3)]．本章では，このような観点から，Prologによるデータベース実現の姿を見ていく．

2.1 事実の表現と問合せ

Prologのプログラムはホーン節の集合からなるが，ホーン節のうち，帰結部のみからなる節をユニット節，さらに，ユニット節の引数に変数を含まないものをファクトと呼ぶ．(ファクトは，ホーン節の一形態の名称であり，現実の世界における事実とは区別する．)

表-1　給料の関係表（例）

給　料	1	2
	a	15万円
	b	20万円

現実の世界における事実は，ファクトを用いて，表現することができる．たとえば，

「aさんの給料は15万円である．」

「bさんの給料は20万円である．」

といった事実は，

給料（a，15万円）．　　(1)

給料（b，20万円）．　　(2)

のように記述できる．これは，関係データベースの関係表（たとえば表-1）に対応するとみなすことができる．

また，データベースに登録された事実を検索する（問合せる）には，Prologでは，たとえば，

？－給料（a，X）．　　(3)

を実行する．これは，ホーン節の他の形態であり，ゴールと呼ばれる．Xは変数である．(3)のゴールを実行すると，(1)，(2)の事実を検索し，

X＝15万円

が得られる．

関係データベースでは，問合せは，関係表の操作に対応するため，表操作を効率よく行えるための問合せ言語の研究・開発が積極的に行われており商用化されたものもある．たとえば，関係データベースに対し，グラフィックを用いて操作性を向上させた言語として著名なQ. B. E.[7)]，自然言語風のSQL[5),6)]等がある．(3)に対応するQ. B. E.を用いた問合せ例を表-2に示す．

2.2 演繹能力

演繹能力とは，すでに定義されている事実と推論規則とを組み合わせて，新たな事実を得る能力のことで

† Database and Prolog by Hitomi IGUSA, Katsushi SUZUKI, Hiroyuki SATO and Takashi DASAI (Information Systems & Electronics Development Laboratories, Mitsubishi Electric Corporation).

†† 三菱電機(株)情報電子研究所

表-2 Q.B.E. を用いた問合せの例

給料	1	2
	a	P.* 15万円

* P. は Print の略であり，その属性の内容(値)を出力する.

ある．Prolog では，この推論規則を，ルールを用いて記述することができる．ルールとは，帰結部及び条件部からなるホーン節である．(2.1 と同様，ルールは，ホーン節の名称である．現実の世界における推論規則とは区別する.)

例として，

「*a* さんの上司は *c* さんである.」
「*b* さんの上司は *e* さんである.」
「*d* さんの上司は *e* さんである.」
⋮
「*a* さんの給料は 10 万円である.」
「*b* さんの給料は 12 万円である.」
「*c* さんの給料は 25 万円である.」
⋮ (4)

という事実があった時，これらから，

「*a* さんの上司の給料」

を求める場合を考える.

Prolog では，ある社員 X の上司の給料 Y という新しい事実を得るための推論規則を，ルールを用いて次のように定義することができる.

上司の給料 (X, Y) :－上司 (X, Z),
給料 (Z, Y). (5)

(4)の事実は，ファクトを用いて次のように書ける.

上司 (*a*, *c*). 給料(*a*, 10万円).
上司 (*b*, *e*). 給料(*b*, 12万円).
上司 (*d*, *e*). 給料(*c*, 25万円).
⋮ ⋮ (6)

そして，X に社員名 (たとえば *a*) を与えて

？－上司の給料 (*a*, Y).

を実行すれば，(5)により，(6)を自動的に検索して Y に答を返してくる．こうして，たとえば，

「*a* さんの上司の給料は 25 万円である.」

という事実が得られる.

関係データベースでは，データベースは事実の集合であり，基本的に，推論規則は表現できない．したがって関係表として定義されていない事実を得るためには，利用する側で，その事実を得るための表操作を

表-3 上司表

上司	1	2
	a	*c*
	b	*e*
	d	*e*

表-4 給料表

給料	1	2
	a	10万円
	b	12万円
	c	25万円

表-5 Q.B.E. による表現例

上司	1	2
	a	WHO*

給料	1	2
	WHO	P. 25万円

* 下線は例要素と呼ばれ，「たとえば～としておく」といった意味である.

explicit に指示してやる必要がある．たとえば，(4)は，表-3，表-4 という 2 つの表に対応づけることができる．新しい事実，「*a* さんの上司の給料」を得るためにはこれらの表に対して，その構造を考慮した表操作演算をほどこさねばならない．Q.B.E. による例を表-5 に示す．SQL では，利用者各々の立場からデータベースをみて，必要に応じて新しい関係を定義できる機能を提供する view の概念がある．これは演繹能力とみなすことができる．また，演繹能力を持つ関係データベースシステムの研究も行われており[2]，たとえば Chang による DEDUCE 2[4] では，推論規則に対応する virtual relation が定義できる.

2.3 冗長表現の除去

2.2 で述べたルールによる推論規則の表現は，データベースにおける冗長な情報を除去し，メモリ効率の向上を計ることにも利用できる．たとえば，社員の基本給が，その所属と階級によって決っているような場合には，たとえば

基本給(X, 15万円) :－社員(X, 営業, 係長). (7)

といったルールを用意すればよく，(7)の右辺に対応する社員についてのファクトがあれば，基本給に関する個々の記述は必要ない.

2.4 統一表現

2.1, 2.2 で示したように，Prolog でデータベース

を記述する場合には，事実を表わすファクトも，推論規則を表わすルールも，またデータベースへの問合せを表わすゴールも，すべてホーン節という，Prologのプログラムで統一的に表現される．したがって，データベースも，その利用プログラムもPrologで記述できる．

2.5 再帰的な定義

Prologは，ホーン節を基礎とするため，再帰的な表現が可能である．したがって，ルールを用いた推論規則の定義にも再帰的な定義が許される．たとえば，再帰的なルールの著名な例である先祖の定義は，Prologでは，次のように書ける．

先祖 (X, Y) :－親 (X, Y). (8)①

先祖 (X, Y) :－親 (X, Z),

先祖 (Z, Y). (8)②

すなわち，XがYの先祖である，というのは，XがYの親であるか，または，XがZの親で，かつ，ZがYの先祖である，ということを意味している．ある人aの先祖を求めたい場合には，たとえば

親 (b, a).

親 (c, b).

⋮ (8)③

というファクトがあるとして，

？－先祖 (X, a).

というゴールを実行する．するとまず，①と③から，$X=b$ が得られる．Prologにはバックトラック機能があるため，ここでバックトラックを指示すると，今度は②と③から $X=c$ が得られる．このように，バックトラックを指示することにより，ある人aのすべての先祖を順次，求めることができる．また①は，再帰的な定義の停止条件となっている．すなわち，XがYの親であり，Yが誰の先祖でもない時，再帰的な定義の呼出しが終了し，解が求まるのである．

従来の関係データベースでは（例，System R[6]），再帰的な定義は扱えないのが一般的であるが，演繹能力を持たせる研究とともに，再帰的な定義を扱う研究も行われている[2]．

また，Prologには，一般にsetof[8]というメタ述語が用意されている．（用意されていなくても，Prologで記述することができる．）メタ述語とは，引数として述語をとれるような述語であり，この他にもいくつかのものが用意されているのが普通である．メタ述語は，Prologの基礎体系である一階述語論理の範疇を超えるものであるが，プログラミング言語としての機能を向上する上で非常に有効なものである．

setofは

setof (X, Y, S)

という形式をとり，Yという条件をみたすすべてのXの集合をSに与える．すなわち，強制的にバックトラックをおこして，すべての可能な解を得，それを集めてSとする．

このsetofと再帰呼出し機能を組み合わせることによって，Prologでは最小不動点演算と呼ばれる演算を実現することができる[9),17),18)]．

最小浮動点演算の例としては[9),10),17),18)]，飛行機の予約システムでおこるような，与えられた時間内における2都市間の可能なフライトを決定する問題や，ある人のすべての先祖を求める問題等があるが，たとえば後者の問題は，setofと(8)を用いて，

？－setof ((X, Y), 先祖 (X, Y), S).

というゴールを実行することによって解ける．

最小不動点演算を含むような問題は，上記の例の他にもデータベースの応用問題として，種々のものがある．しかし，これは，従来の関係データベースでは扱えない．このため，問合せ言語にこれを扱える機能を持たせる研究が，Aho, Ullmanによって行われている[10]．

2.6 関係演算

関係データベースでは，データベースを操作して目的の情報を得るための基本演算が用意されている．これらの基本演算はPrologには用意されていない．しかし，Prologの性質から，それらの基本演算を簡潔に実現することができる．Wrightは，その基本的な考え方を示している[9]．以下，ここでは，詳細は文献に譲り，いくつか例を挙げておく．

まず，関係は集合とみなせるから，2つの関係rとsについて，通常の集合演算である合併演算，共通部分演算，差演算，直積演算が可能である．このうち，たとえば合併演算は

$t(X_1, \cdots, X_n)$:－$r(X_1, \cdots, X_n)$; $s(X_1, \cdots, X_n)$.

のように表現される．；はORを意味する．また，共通部分演算は

$t(X_1, \cdots, X_n)$:－$r(X_1, \cdots, X_n)$, $s(X_1, \cdots, X_n)$.

と書ける．

また，関係データベースには，関係から新たな関係を作り出す関係代数独特の演算として，射影・結合・制約・割り算の4つの演算がある．このうち，たとえば結合は，

$t(X_1, \cdots, X_n, Y_1, \cdots, Y_m) :- X_i \theta Y_j,$
$r(X_1, \cdots, X_i, \cdots, X_n),$
$s(Y_1, \cdots, Y_j, \cdots, Y_m).$

と表わされる．θ は，関係 r の属性 X_i と関係 s の属性 Y_j との結合条件を示す．

3. 問合せの効率化

データベースに対する問合せをいかに効率よく行うか，という問題は，検索スピード，メモリ効率の両面から重要であり，関係データベースに対する問合せの最適化手法がこれまでに数多く研究されている[6),11)~13)]．たとえば，関係代数では，演算結果のメモリ容量がなるべく小さくなるような演算を優先して実行させる手法や[11)]，さらにそれをデータ表現にまで拡張した手法[12),13)]などが開発されている．

一方，Prolog のゴールの実行は，ゴールを構成するサブゴールの記述順序に従って行われるため，サブゴールの並び方によっては不必要なバックトラックが大量におこることがある．このため，Prolog でデータベースを記述する場合にも，サブゴールの順序の最適化という形で問合せの最適化が必要となる．

この最適化手法が Warren によって与えられている[14)]．Warren の最適化手法はプランニングアルゴリズムと呼ばれ，彼の作成した自然言語（英語）によるデータベース検索システム CHAT-80 上で実現されている．以下，このプランニングアルゴリズムの概要を紹介する．

Prolog のゴールの実行効率は，ゴールを構成するサブゴールの実行順序に依存しておこるバックトラックの数により決定される．したがって，ゴール実行中におこるバックトラックの数を最小にすることが，最適化に相当する．Warren は，サブゴールにおいて実行される可能性のある alternative 節の数の期待値を表わすコスト関数という数値を導入した．各サブゴールのコストは次のように計算される．

$$\text{コスト} \approx \frac{\text{サブゴールの述語に対応するファクト数}}{\text{instantiateされた引数の定義域のサイズ}}$$

instantiate された引数とは，変数でない引数，つまり実値が結合されている引数を意味する．たとえば，

borders (C, C_1)

という述語に対し，

ファクト数	900
引数の定義域のサイズ	$C=180$
	$C_1=180$

表-6 borders (C, C_1) のコスト

引数 (C, C_1) の状態	コ　ス　ト
両方，instantiate されていない	900
どちらか，instantiate されている	900/180=5
両方，instantiate されている	900/180×180=1/36

とする．borders のコストは**表-6** のようになる．

引数が instantiate されていれば，その引数に対するバックトラックはないから，alternative 節の数の期待値を示すコストは小さくなる．また，あるサブゴールを実行した結果，関連する引数に値が instantiate される可能性があり，それは，まだ実行されていない残りのサブゴールの実行に影響する．このような理由から，プランニングアルゴリズムでは，与えられたゴールの各サブゴールのコストを計算し，その最小のものをまず選択する．そして次に，そのサブゴールの引数がすべて instantiate されたと仮定して，残りのサブゴールのコストを計算しなおす．これを繰り返し，最終的に最適と予測されるゴールを生成する．

たとえば，CHAT-80 に対する質問

"Which countries bordering the Mediterranean border Asian countries?".

は(9)のゴールに変換され，さらに，プランニングアルゴリズムの適用により，(10)が最適なゴールとして生成される．

?−country (C), borders $(C,$ mediterranean), country(C_1), asian (C_1), borders (C, C_1). (9)

⇩

?−borders $(C,$ mediterranean), country (C), borders (C, C_1), asian (C_1), country (C_1). (10)

ただし，このプランニングアルゴリズムでは，各述語に対してコストが容易に計算できることが必要である．したがって，ファクトの場合は問題ないが，ルールにおいては，コストが容易に計算できるか否かは，データベースの定義者にまかされている．

CHAT-80 では，プランニングアルゴリズムの利用により，データベース検索スピードが約 1 桁向上することが，実験により，明らかとなっている．

また，このプランニングアルゴリズムは，日本語質問応答システムと呼ばれる，日本語によるデータベース検索システム[15)]でも利用されている．このシステムは，列車の時刻表関連のデータベースを持ち，柔軟

な日本文による検索が可能なシステムであり，入力文から検索用のゴールを生成した後，プランニングアルゴリズムに基づいて，ゴールの最適化を行っている．ただし，ルールのコスト計算も機械的にできるよう，経験に基づく算法を提案・利用している．

Warrenの手法は，従来の関係データベースにおける最適化，特に，System R[6]における最適化とよく似ている．System Rでも，関係表へのアクセス経路に基づくアクセスコストの概念が導入されている．しかし，Warrenの手法は，Prologプログラムの一部である小規模なデータベースに対してのみ適応できるものである．

4. 実用化への方策

以上，Prologによるデータベースの実現とその特徴，及び，問合せの最適化について述べてきたが，実用化という観点から考えた場合には，Prologだけでデータベースとそれを利用するシステム全部を記述するのは困難である．なぜなら，2章で述べたように，Prologでは，ルールもファクトもプログラムの一部である．プログラムは主記憶上に置かれるため，どうしてもその大きさが制限されてしまう．

そこで，実用化に向けては，Prologが演繹能力を持つ関係データベース処理言語とみなせることから，Prologと従来の関係データベースシステムとの結合が有効と考えられ，すでにいくつかの研究がなされている[16)~19)]．

演繹能力を持つ関係データベースにおける問合せの評価方法については，従来から研究がなされており，evaluational approachとnon-evaluational approachが提案されている[2)]．前者では，問合せの評価は事実に対してのみ行われ，推論規則は与えられた問合せを事実に対する問合せに変換するのに利用される．事実に対する検索をまとめて行うため，集合演算が使える．また，従来の関係データベースシステムに対し，直接利用可能なものと言える．これに対し，後者では，事実も推論規則も問合せの評価を行うのに，区別なく用いられる．このように，事実と推論規則を区別しない点，Prologに適合していると言える．しかし，集合演算は扱えない．

Prologと関係データベースシステムを結合する場合の問合せ評価方法としては，non-evaluational approachは非効率的である．すなわち，問合せの中に，Prologプログラムの実行要求と，関係データベースへの検索要求が入り混じる場合が考えられる．逆にevaluational approachは，効率の点では有利であるが，Prologの世界と関係データベースシステムとの世界をどのように結合するか，すなわち，問合せから関係データベース検索にあたる部分をどのように分離・抽出するか，を考えなくてはならない．このような観点に基づく研究例に，国藤，横田らの手法[17),18)]がある．

彼らは，事実や推論規則の格納場所を示すidb, edbという述語を導入し，関係データベースへの検索要求を生成する方法を開発している．この方法では，事実の大部分は関係データベースシステムに，推論規則と残りの事実（一時的に使われる事実など）はProlog側に格納するものとしている．idbは，事実・推論規則のうち，Prolog側に格納されているものを示す述語であり，internal data baseの略である．edbは，関係データベースシステムにあるものを示す述語であり，external data baseの略である．Prologのプログラムの実行は，idbとedbから，関係データベースシステムへの検索要求を示すプランを生成することに相当する．そして，このプランが関係データベースの諸演算に変換されて，関係データベースの検索が行われる．

この方法では，事実や推論規則の格納場所にあわせて，あらかじめこれらをidb, edbを用いて定義しておく．たとえば先祖を求めるプログラムにおいて，先祖の定義はProlog側，親に対応する事実は関係データベース側にあるとすると，プログラム(8)は，次のような形で定義される．

idb ((先祖 (X, Y) :— 親 (X, Y))).
idb ((先祖 (X, Y) :— 親 (X, Z), 先祖 (Z, Y))).
idb ((親 (X, Y) :— edb (親 (X, Y)))).

ここで，ある人aさんの先祖を捜すために

?—先祖 (Y, a)

という問合せをすると，関係データベースへの検索要求を示すプラン

[親 (Y, a)]

が生成される．これは，次に関係データベースの諸演算に変換され，検索結果がYに返される．バックトラックが起こって別解が要求された場合には

[親 (Y_1, a), 親 (Y, Y_1)]

が生成され，やはり検索結果がYに返される．このように，順次，新しいプランが生成される．

この方法は，internal data base，すなわち，Prolog

側のデータベースに対しては，non-evaluational approach を，そして，external data base，すなわち，関係データベースシステム側にあるデータベースに対しては，evaluational approach を適用している，とみなせる．これは，前記の2つの approach を統合したものとみなすことができる．

なお，この方法は，第5世代コンピュータ・プロジェクトにおいて，知識情報処理システムの第1ステップとして開発中の関係データベースマシン (Delta) と逐次推論型マシン (SIM-P) を結合した研究開発用システムで利用されている．

5. おわりに

本稿では，Prolog と関係データベースとの関連について，主に Prolog によるデータベースの実現手法，及びその問合せについて述べたが，このように，Prolog は，演繹能力を持つ関係データベース処理言語とみなせるなど，関係データベースと密接な関係を持っている．したがって，これらを結合・利用する技術を確立することにより，知識情報処理の分野における種々の応用システムの実用化の可能性を向上させることができる，と考えられる．今後も，これら技術の積極的な研究・開発が望まれる．

参考文献

1) 植村俊亮：データベースシステムの基礎，オーム社 (1979).
2) Gallaire, H. and Minker, J.: Logic and Data Bases, Plenum Press (1978).
3) Kowalski, R.: Logic as a Database Language, Dept. of Computing, Imperial College, London (1981).
4) Chang, C. L.: DEDUCE 2: Further Investigations of Deduction in Relational Data Base, in ref. 2).
5) Astrahan, M. M. and Chamberlin, D. D.: Implementation of a Structured English Query Language, CACM, Vol. 18, No. 10, pp. 580–588 (1975).
6) Astrahan, M. M. et al.: System R: A Relational Approach to Data Base Management, ACM Trans. Database System, Vol. 1, No. 2, pp. 97–137 (1976).
7) Zloof, M. M.: Query-by-Example: A Data Base Language, IBM Syst. J., Vol. 4, pp. 324–343 (1977).
8) Warren, D. H. D.: Higher-order Extensions to Prolog—Are They Needed?, D. A. I. Research Paper, 154 (1981).
9) Wright, D. J.: Prolog as a Relationally Complete Database Query Language Which Can Handle Least Fixed Point Operations, Univ. of Kentucky Technical Report, No. 73-80 (1980).
10) Aho, A. V. and Ullman, J. D.: Universality of Data Retrieval Languages, ACM/SIGPLAN Conf. on Principles of Programming Languages (1979).
11) Smith, J. M. and Chang, P. T. Y.: Optimizing the Performance of a Relational Algebra Database Interface, CACM, Vol. 18, No. 10, pp. 568–579 (1975).
12) 古川康一：データベースの知的アクセスに関する研究，電総研研究報告 804 (1979).
13) 古川康一：関係データベースに対するデータアクセスの数式処理による最適化について，情報処理学会論文誌，Vol. 22, No. 1, pp. 68–75 (1981).
14) Warren, D. H.: Efficient Processing of Interactive Relational Database Queries Expressed in Logic, IEEE, CH1701-2 (1981).
15) 鈴木，太細，伊草，関本，向井：日本語質問応答システムにおける知識の表現と利用，電子通信学会，AL82-70, PRL82-58 (1982).
16) Chakravarthy, U. S., Minker, J. and Tran, D.: Interfacing Predicate Logic Languages and Relational Databases, Proc. of the 1st Int. Logic Programming Conf. (1982).
17) Kunifuji, S. and Yokota, H.: Prolog and Relational Databases for Fifth Generation Computer System, ICOT Technical Report, TR-002 (1982).
18) 横田，国藤，柴山，宮崎，角田，村上：Prolog による推論機構と関係データベースの結合，ICOT Technical Report, TR-031 (1983).
19) 田中，堀内，田川：推論システムとデータベースシステムとの部分評価機構による結合，計測自動制御学会，第1回知識工学シンポジウム (1983).
20) Pereira, L. M., Pereira, F. C. N. and Warren, D. H. D.: User's Guide to DEC System-10 PROLOG, Laboratorio Nacional de Engenharia Civil (1978).
21) Clocksin, W. F. and Mellish, C. S.: Programming in Prolog, Springer-Verlag (1981).

(昭和59年9月11日受付)