記号処理 18－4
（1982. 3. 25）

ロジック・プログラミングとデータベース

國藤 進，加藤昭彦，安達統衛，竹島 卓，
沢村 一（富士通・国際情報社会科学研究所）

〔概要〕 Prolog系論理型言語を関係データベース（RDB: Relational Data Base）への演繹的質問応答に適用する場合の基本的問題点とその解決の指針について述べる．まず関係データベース管理システム（RDBMS: RDB Mangement System）の基本概念とロジック・プログラミング（LPG: Logic ProgrammiG）との関連について述べる．ついでRDBモデルへの基本操作をユーザの問合せ言語の視点から分析し，関係論理や関係代数をサポートする問合せ言語を提案する．またメタ述語 setof を保有するProlog系言語の関係完備性（relational completeness）を示す．最後にこの種の問合せ言語を，RDBMS に組込んだLPGシステムの全体像について概観する．

〔1〕 はじめに

本小論では次のような理由で，外部データベース管理システム（EDBMS: External Data Base Management System）としてはRDBMSのみを考察対象とする．

(ア) 第5世代コンピュータ・システム用核言語（FGKL: Fifth Generation Kernel Language）がPrologベースの論理型言語であること[1]．

(イ) 論理でいう関係と関係モデルでいうRDBが概念的にも1対1に対応すること．

(ウ) 論理型言語の採用によりRDB，知識ベース（KB: Knowledge Base），推論機構の知識表現を一様なものとしめること．

(エ) 論理型言語は堅固な数学的基盤をもち，かつ分り易く，表現し易く，簡潔となり，また明解な記述力を提供すること．

ここに汎用大規模RDBMSやRDBマシンといったEDBMSとのインタフェースを考える．その理由はProlog系言語が通常取扱う内部データベースに格納されている一般的知識の総量は，EDBMSが外部データベースとして保有している特殊な知識の総量より相対的に極めて小さいからである．われわれの目的とするのは，Prolog系言語とEDBMSとの整合性の良さをいかしたインタフェースの提案である．

さてEDBMSとしてのRDBに対する基本操作にはデータ定義言語とデータ操作言語がある．ユーザにとって親しみ易いデータ操作言語，すなわち問合せ言語には関係論理型，関係代数型，要素操作型，写像型，例題型，自然言語型[2]等がある．本小論ではLPGシステムとEDBMSとのインタフェースをとることを主眼に，関係論理型と関係代数型の問合せ言語についての仕様を提案する．

〔2〕 語準備

〔2.1〕 関係データベースの1階述語論理的意味

Prologが処理しようとする論理は1階述語論理（1PL: First-order Predicate Logic）の部分クラスである1階Horn論理である．ところでRDBモデルは複数の定義域に関する直積の部分集合である関係の集合として構成されている．そこで単一の定義域の直積しか取扱わない1PLでは，必ずしも適切な概念構成法を提供し

えない．RDBモデルを論理的に埋込む最も適切な論理系は，1階多類論理(1ML: First-order Many-sorted Logic)である．1MLの構文法，構造等の厳密な構成法は文献3)に譲るが，1MLも1PLの部分クラスであることに変りはない．

〔2.2〕 非評価方式と評価方式

RDBへの演繹的質問応答システム構成法には，次の2方式がある4),5)．

(A) 非評価方式(non-evaluational approach)： RDBは他の一般的知識とともにある形式的体系の公理をなすと考える，推論方式としては，1PL用Theorem Proveテクニックが使える．

(B) 評価方式(evaluational approach)： RDBと他の一般的知識は全く別個のものとみなす．後者からなるKBがある形式的体系の公理をなすと考え，前者はその体系のひとつのモデルをなすと考える．従って推論方式としては，独自の評価機構の確立が必要である．

大まかにいえば内部データベース方式と(B)が，外部データベース方式と(A)が相性がよい．

〔2.3〕 関係データベースのProlog系言語表現

LPGシステムに親言語としての問合せ言語を埋込む方式について提案する．

〔2.3.1〕 関係論理的アプローチ

Prolog系言語に関係論理型問合せ言語を埋込むのは，極めて自然な考えである．直感的にはRDBの各タップルをFGKLの表明節に対応づけることに相当する．このアプローチによれば，更に次の2つの表現法が考えられる．

A. 属性値変数／定数を使う方法： 関係を述語とみなすやり方である．"タップル$\langle d_1, d_2, \cdots, d_n\rangle$が関係rに属する($\langle d_1, d_2, \cdots, d_n\rangle \in r$)"ことを，FGKL流に"$r(d_1, d_2, \cdots, d_n) \leftarrow$"と表記することにする．

B. タップル変数／定数を使う方法： 関係をタップルを引数とする1引数述語とみなすやり方である．"$\langle d_1, d_2, \cdots, d_n\rangle \in r$"を"$r(d_1.d_2.\cdots.d_n\langle\rangle) \leftarrow$"と表記することになる．

〔2.3.2〕 関係代数的アプローチ

Prolog系言語に関係代数型問合せ言語を埋込むのは，多少不自然な所もある．しかしながらハードウェアの特性やEDBMSとの結合を考慮し，関係代数レベルの集合演算をインタフェースとすることは，性能向上の用途がつきやすく，また将来的にも並列型マシンの研究開発へと発展していく可能性が高い5)．

このアプローチでは関係をひとつの項として扱う．そこで関係の表現法としてタップル(あるいはリスト)のリストで表現すると約束する．これにより関係代数の基本演算を，全てリスト処理で実現するというアプローチである．

〔3〕 関係論理サポート述語

Prolog系言語は関係論理となじみやすい．ここでは集合演算や関係演算といった基本となる関係論理サポート述語を，2.3.1.A／Bの表記法で示す．

〔3.1〕 集合演算

所与の関係f(*tuple)と関係g(*tuple)の論理和，論理積，論理差が関係r(*tuple)かどうか判定する述語は，FGKL流表記法によれば，それぞれ(1)，(2)，(3)で与えられる．また全く同様にして，所与の関係f(*tuple1)と関係g(*tuple2)の論理直積が関係r(*tuple1^*tuple2)かどうか判定する述語は，(4)で与えられ

(1) r(*tuple) ← p(*tuple)
r(*tuple) ← q(*tuple)
(2) r(*tuple) ← p(*tuple) ∧ q(*tuple)
(3) r(*tuple) ← p(*tuple) ∧ not(q(*tuple))
(4) r(*tuple1^*tuple2) ← p(*tuple1) ∧ q(*tuple2)

〔3.2〕 関係演算

所与の関係 p(*x1, *x2, …, *xm) の添字集合 {i1, i2, …, in} 方向への射影が r(*xi1, *xi2, …, *xin) かどうか判定する述語は，(5)で与えられる．所与の関係 p(*x1, …, *xi, …, *xm) と関係 q(*y1, …, yj, …, yn) の論理的な θ-結合関係が r(*x1, …, *xi, …, *xm, *y1, …, *yj, …, *yn) かどうか判定する述語は，(6)で与えられる．また所与の関係 p(*x1, …, *xi, …, *xj, …, *xm) の論理的な θ-制約関係が q(*x1, …, *xi, …, *xj, …, *xm) かどうか判定する述語は，(7)で与えられる．ここに2項述語 θ は evaluable predicate である．

(5) q(*xi1, *xi2, …, *xin) ← p(*x1, *x2, …, *xm)
(6) r(*x1, …, *xi, …, *xm, *y1, …, *yj, …, *yn) ← p(*x1, …, *xi, …, *xm) ∧ q(y1, …, yj, …, yn) ∧ θ(*xi, *yj)
(7) q(*x1, …, *xi, …, *xj, …, *xm) ← p(*x1, …, *xi, …, *xj, …, *xm) ∧ θ(*xi, *xj)

上記の射影，θ-結合，θ-制約を表わす関係論理サポート述語に対して，商を表わす関係論理サポート述語を定義することは，そんなに簡単でない．

〔3.3〕 その他の関係論理サポート述語

実用的見地からすれば外延化述語，存在化述語，numerical quantifier，及び aggregation function 等が必要である．そのうち最も有用なのは，述語 *pred を証明可能とするような *x のあらゆる実現値の集合を *set として取出す外延化述語 "setof(*x, *pred, *set)" (9) である．これについては，その利用法を後述する．

〔4〕 関係代数サポート述語

〔4.1〕 基礎となる集合演算

ある要素 *element が集合 *set に属するかどうか判定するメンバシップ述語 "member(*element, *set)" と，集合 *subset が集合 *set の部分集合であるかどうかを判定する部分集合述語 "subset(*subset, *set)" が最も基礎的な述語である．それらはそれぞれ(8)，(9)で与えられる．

(8) member(*e, *e.*r) ←
member(*e1, *e2.*r) ← member(*e1, *r)
(9) subset(⟨⟩, *s) ←
subset(*e.*r, *s) ← member(*e, *s) ∧ subset(*r, *s)

〔4.2〕 通常の集合演算

集合 *set1 と集合 *set2 の和集合，積集合，差集合が *set かどうか判定する述語は，それぞれ(10)，(11)，(12)で与えられる．

(10) union(⟨⟩, *s, *s) ←
union(*x, *y, *z) ← union1(*x, *y, *z)
union1(*e.*r, *s, *a) ← member(*e, *s) | union(*r, *s, *a)
union1(*e.*r, *s, *e.*a) ← | union(*r, *s, *a)
(11) intersection(⟨⟩, *s, ⟨⟩) ←
intersection(*x, *y, *z) ← intersection1(*x, *y, *z)
intersection1(*e.*r, *s, *e.*a) ← member(*e, *s) | intersection(*r, *s, *a)

```
    intersection1(*e.*r,*s,*a) ← | intersection(*r,*s,*a)
(12) difference(⟨⟩,*s,⟨⟩) ←
    difference(*x,*y,*z) ← difference1(*x,*y,*z)
    difference1(*e.*r,*s,*a) ← member(*e,*s) | difference(*r,*s,*a)
    difference1(*e.*r,*s,*e.*a) ← | difference(*r,*s,*a)
```

集合*set1 て集合*set2 の直積が*set かどうか判定する述語 " cartesian(*set1, *set2,*set)"を定義することも易しい.

〔4.3〕 関係演算

関係代数独自の演算である射影, θ-結合, θ-制約, 商を表わす閉合せ述語として, それぞれ以下のように概念規定される メタ述語を組込んでおくのが望まれる.

(a) 射影 projection(*rel,*indexes,*result): 関係*relの添字集合*indexes 方向への射影が*result かどうか判定する述語 (*rel[*indexes] = *result).

(b) θ-結合 join(*rel1,*ind1,θ,*rel2,*ind2,*result), ただし θ は " =, ≠, <, ≦, >, ≧ " のどれかとする: 関係*rel1 の添字集合*ind1 と関係*rel2 の添字集合*ind2 との間の θ-結合関係が*result かどうか判定する述語 (*rel1[*ind1 θ *ind2]*rel2 = *result)

(c) θ-制約 restriction(*rel1,*ind1,θ,*ind2,*result), ただし θ は ibid.: 関係*rel1 の添字集合*ind1 と添字集合*ind2 の間の θ-制約関係が*result かどうか判定する述語 (*rel1[*ind1 θ *ind2] = *result)

(d) 商 division(*rel1,*ind1,*rel2,*ind2,*result): 関係*rel1 の添字集合*ind1 方向への射影を, 関係*rel2 の添字集合*ind2 方向への射影で割算を行, た結果の商が*result かどうか判定する述語 (*rel1[*ind1 ÷ *ind2]*rel2 = *result)

以上の4演算のうち, (a), (b) と (c) はリストのリストというデータ構造を用いて, pure-Prolog (エジンバラ大学第1版程度の Prolog) でも簡単に定義できる (例えば射影の場合, 下記の例(13)を参照のこと). しかしながら商演算 (d) を pure-Prolog で定義することは, 有限集合の場合を除いて, 極めて困難のように思える.

```
(13) projection(⟨⟩,*s,⟨⟩) ←
    projection(*x.*y,*s,*x1.*y1) ← pickup(*x,*s,*x1) ∧ projection(*y,*s,*y1)
    pickup(*x,⟨⟩,⟨⟩) ←
    pickup(*x,*a.*b,*e.*y) ← elementof(*a,*x,*e) ∧ pickup(*x,*b,*y)
    elementof(1,*e.*x,*e) ←
    elementof(*m,*e1.*x,*e2) ← elementof(*m-1,*x,*e2)
```

〔4.4〕 その他の関係代数サポート述語

実用的見地からすれば集合の性質を操作する述語, 特に第2引数*arg1 のそれぞれに第1引数*x を作用させた結果を返す高階の作用述語 apply(*x,*arg1), ソート/マージ述語を標準装備しておくことが望まれる.

〔5〕 関係完備

Codd[7] によれば, 関係論理を用いたアルファ式と同等以上の表現能力をもつ閉合せ言語のことを関係完備 (relational complete) という. 彼は関係論理を用いたいかなるアルファ式も, 関係代数演算に置換可能なことも示し, 関係代数が関係完備であることを証明した. 本節では pure-Prolog に メタ述語 setof[9] を組込ん

でProlog（エジンバラ大学第3版の部分クラス）が，関係完備であることを示す．
〔関係完備性証明の筋道〕
4. 通常の集合演算である和集合，積集合，差集合，直積に相当する概念は，setof を含むPrologではそれぞれ次のようなメタ述語で実現される．

i) setof(*tuple, p(*tuple) ∨ q(*tuple), *set)
ii) setof(*tuple, p(*tuple) ∧ q(*tuple), *set)
iii) setof(*tuple, p(*tuple) ∧ not(q(*tuple)), *set)
iv) setof(*tuple1^*tuple2, p(*tuple1) ∧ q(*tuple2), *set)

3. 関係から関係を作り出す関係代数独自の演算である射影，θ-結合，θ-約約，商に相当する概念は，それぞれ次のようなメタ述語で実現される．

i) setof((*X_{i_1}, *X_{i_2}, ..., *X_{i_m}), p(*X_1, *X_2, ..., *X_n), *result)
ii) setof((*X_1, ..., *X_i, ..., *X_n, *Y_1, ..., *Y_j, ..., *Y_m), p(*X_1, ..., *X_i, ..., *X_n) ∧ q(*Y_1, ..., *Y_j, ..., *Y_m) ∧ θ(*X_i, *Y_j), *result)
iii) setof((*X_1, ..., *X_i, ..., *X_n), p(*X_1, ..., *X_i, ..., *X_j, ..., *X_n) ∧ θ(*X_i, *X_j), *result)
iv) setof((*X_{R_1}, *X_{R_2}, ..., *X_{R_{n-k}}), setof((*X_{i_1}, *X_{i_2}, ..., *X_{i_k}), p(*X_1, *X_2, ..., *X_n), *projof p) ∧ setof((*Y_{j_1}, *Y_{j_2}, ..., *Y_{j_k}), q(*Y_1, *Y_2, ..., *Y_n), *projof q) ∧ subset(*projof q, *projof p), *result)

AとBによりsetof 付Prologの関係完備性が証明された． (q.e.d.) ■

(iii) (i), (ii), (iv), (iii) が pure-Prolog でも実現できることは，それぞれ(ii), (iii), (i), (iv)で示した．setof は(1)～(7)等で与えた関係論理サポート述語を，直接に関係代数サポート述語に変換する極めて強力なメタ述語である．

〔6〕 システム構成

FGKL用推論マシン単体で，内部知識データベースに格納されている公理集合をもとに，質問応答している際のLPGシステム構成イメージを図1に示す．これに対して，推論マシンとRDMSを結合し，ひとつのLPGシステムへ構成したイメージが，図2に与えられる．本節では現状における技術的展望や理論的背景を前提とし，図2で与えられるLPGシステム像を実現する方式を概観する．

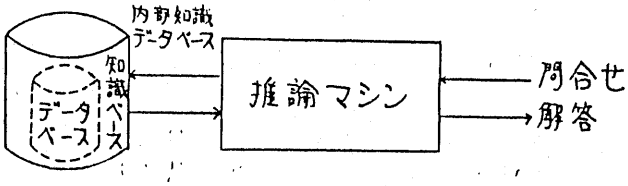

図1 ロジック・プログラミング・システム

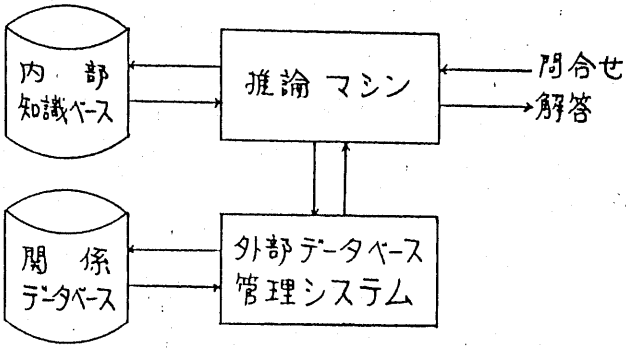

図2 結合方式ロジック・プログラミング・システム

〔6.1〕 結合グラフ法を用いる評価方式

2.2節で示唆した評価方式の特徴は，① KBとRDBの完全分離，② 結合グラフ法という評価方式の利用，にある．本方式は理論的基盤もしっかりしており，明確なシステム設計方針を与えるので，LPGシステム実現方式として有望である．しかしながら論理型言語FGKLの基本／拡張計算機構，すなわちパターン照合機構で結合グラフ法をどのように結合していくかという技術的課題を抱えている．

〔6.2〕 関係論理の関係代数への埋込み方式

EDBMSは通常，関係論理型あるいは関係代数型のどちらかを問合せ言語にしていることが多い．そこで本小論で考察したLPGシステムも，両者の問合せ言語を保有していた．関係論理型でいくか関係代数型でいくかは，アーキテクチャ決定要因のひとつであるが，今後の研究開発課題であるRDBマシンへの接続可能性，並列型KBマシンへの拡張可能性，やハードウェアの特性等を考慮した上，関係代数型を基本とすることにし，関係論理型問合せ言語を全て関係代数型問合せ言語に埋込む方式を提案する．LPGシステム・ユーザが使用するある種の関係代数コマンドが，EDBMS側の関係代数コマンドに，マクロとして受渡され処理される訳である．本方式実現のキー・テクノロジーはメタ述語setofである．setofを用いれば，主要な関係論理述語が関係代数述語に変換されることは，既に5章で示した．実際のProlog処理系では，setofの性能は余り良くないといわれている．そこでEDBMSのもつ高速の集合／関係演算をフルに利用する関係代数コマンドへ置換した後，実際の演算処理をEDBMS側で行えば，大巾な性能向上が期待される．

〔7〕 おわりに

Prolog系論理型言語をRDBへの演繹的質問応答に適用する場合に，本小論で得られた結論を要約する．

(a) RDBのProlog系言語表現に関係論理的アプローチと関係代数的アプローチの両者があることの指摘．

(b) 関係論理／関係代数サポート述語のうち基本となるものを，FGKL流言語表現を用いて明示．

(c) メタ述語setofを含むPrologの関係完備性証明

(d) LPGシステム像の中でRDBへの質問応答を位置付け，具体的実現方式として関係論理の関係代数への埋込み方式を提案．

今後の研究課題は多いが，各章でその概略を指摘したので，ここでは省略する．

〔謝辞〕 本小論は第5世代コンピュータ"ロジック・プログラミング技術調査委員会"報告書の第II編第4.2節"外部データベース・システム"の要約である．本報告内容に関してご討論していただいた横井俊夫委員長（電総研）他13名の委員の方々に感謝する．また日頃ご指導いただく北川敏男所長，関連分野についてご教授いただく情報システム研究会（大須賀節雄東大教授他）の諸先生方に感謝する．

〔参考文献〕

1) ロジック・プログラミング技術調査委員会：'FGKL/S マシンの基本構想，財日本電子工業振興協会，March 1982.
2) 植村俊亮：データベースシステムの基礎，オーム社，1979.
3) Kunifuji, S.: Second-order Many-sorted Boolean Logic for Knowledge Representation Language, IIAS-SIS Fujitsu Ltd, R.R. No. 14, Feb. 1981.
4) Gallaire, H. and Minker, J. (eds.): Logic and Data Bases, Plenum Press, 1978.
5) 古川康一：データベースの知的アクセスの研究，東京大学学位論文，1979.
6) Byrd, L. et al.: A Guide to Version 3 of DEC-10 Prolog and Prolog Debugging Facilities, DAI Occasional Paper 19, Univ. of Edinburgh, 1980.
7) Codd, E.F.: Relational Completeness of Data Base Sublanguages, in Courant Computer Science Symposium 6 Data Base Systems, Prentice-Hall, 1972.