ICOT Technical Memorandum: TM-0255

TM-0255

KBMS PHI
問い合せ処理モデル

羽生田博美，高杉哲朗，宮崎収兄
(沖電気工業)
伊藤英則

February, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

KBMS PHI：問い合せ処理モデル

羽生田 博美*, 高杉 哲朗*, 宮崎 収兄*, 伊藤 英則**
*:沖電気工業(株), **:(財)新世代コンピュータ技術開発機構

1．はじめに

第5世代コンピュータ研究開発プロジェクトにおける分散知識ベース・システムPHIの研究では，先にシステム構成方式，及び知識ベースの扱いとデータの分散化個々の問題について報告した．

本稿では，これらの検討を基本にしたPHIの分散知識ベース・システムとしての問い合せ処理モデルについて述べる．PHIはこの処理モデルに基づきPSI上にESPを用いて実現する予定である．

2．システム構成

本章では，問い合せ処理の観点からPHIの論理的システム構成について述べる．

PHIは機能的には知識管理部と分散制御部から構成されている[伊藤86]．知識管理部は演繹データベース管理を核とする知識管理層と関係データベース管理層から構成されており，コンパイル・アプローチに基づく処理方式を基本としている[宮崎86]．また，分散制御部は分散データベース管理部を核として分散化を計っている．

分散データベースの制御方式では，1つのトランザクション・マネージャと複数のデータ・マネージャからシステムが構成される．各データ・マネージャは分散データの格納サイトに対応しており，各々サイトに閉じた処理を自律して行い，必要があれば他のデータ・マネージャと協調して処理を行う[吉田86]．

問い合せ処理の観点から見たPHIのシステム構成はこれらの方式を基本としている．問い合せ処理におけるPHIの論理システム構成を図1に示す．

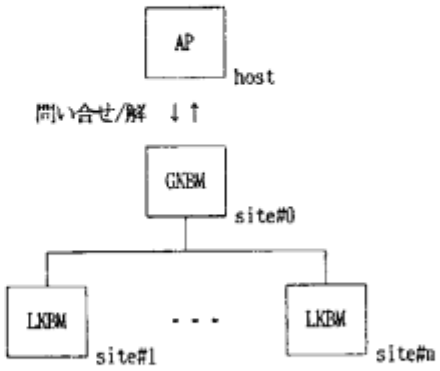
図1の構成は，ホストから出される問い合せに対応して動的に形成される．アプリケイション・プログラム（AP）はPHIが管理する共有知識ベースへアクセスするエキスパート・システム等の知識処理システムである．

グローバル知識ベース・マネージャ（GKBM），およびローカル知識ベース・マネージャ（LKBM）は，それぞれ分散データベース制御におけるトランザクション・マネージャ，データ・マネージャ[吉田86]を機能的に拡張したモジュールである．GKBMは問い合せに関して全LKBM に共通した処理を行い，LKBMは分散データに対応して各サイト毎に存在し，各々のサイトに閉じた処理を，必要に応じて他のサイト上のLKBMと協調して行う．

3．問い合せ処理方式

本章では上述したシステム構成での問い合せ処理方式について述べる．

KBMS PHI:Query Processing Model
Hiromi HANIUDA*, Tetsurou TAKASUGI*,
Nobuyoshi MIYAZAKI*, Hidenori ITOH**
*Oki Electric Industry Co., Ltd.
**Institute for New Generation Computer Technology



AP: Application Program
GKBM:Global Knowledge Base Manager
LKBM:Local Knowledge Base Manager

図1 問い合せ処理におけるPHIの論理システム構成

ホストからPHIに対してはホーン節形式の問い合せが出され[宮崎86,MIYA86]，1つのグローバル知識ベース・マネージャと，複数のローカル知識ベース・マネージャによって処理される．

前章で述べたようにPHIはコンパイル・アプローチに基づいた問い合せ処理を行う．したがって，(a)データへのアクセス・プランの生成と，(b)その実行は別々のフェーズに分かれている．またアクセス・プランの生成は，(a1)問い合せおよび内包データベースに対するホーン節の範囲での処理と，(a2)その結果に対する最小不動点(LFP)演算の繰り返し処理戦略の決定[宮崎86]と，関係代数演算の生成処理とに分けることができる．

LFP戦略の決定と関係代数演算の生成，及びその実行は外延データベースに密着した処理であり，データの格納サイトに処理を局所化できる．一方，問い合せそのものとIDBに対する処理は，特定のサイトに局所化することはできず，システム全体に関係している．したがって，GKBMではシステム全体に関係する処理(a)を，一方LKBMはサイトに局所化できる処理(a2,b)を行う．

以下，GKBMとLKBMに分けて処理方式について述べる．

3．1．グローバル知識ベース・マネージャ

グローバル知識ベース・マネージャは，問い合せと内包データベースをもとにホーン節の範囲で処理を行い，各ローカル知識ベース・マネージャに対する共通の内部問い合せを生成する．この内部問い合せは全LKBMに放送され，各LKBMで自律的に処理される．

以下，GKBMで行う処理について述べる．

(1) 問い合せ及び内包データベースに対する前処理

まずアプリケイション・プログラムから与えられたホーン節問

い合せと内包データベースを統合してホーン節集合を生成する。このホーン節集合はAPから与えられる問い合せとみなすことができるので単にホーン節問い合せ(HCQ)と呼ぶ。次に，一般にHCQには引数として定数が含まれるが，HCQに対する処理を単純にするためにこれらの定数を変数と比較述語に変換する。

(2) ホーン節変換

前処理を施されたHCQに対して，ホーン節変換[MIYA86]を行う。ホーン節変換は，一種の部分評価であり，不要な中間述語が削除される。再帰述語は自己再帰的な述語のみが残る。

(3) 強連結成分分解

HCQ中のルールで，ヘッド述語の評価はボディ部の述語の評価結果に依存していると考えることができ，HCQはこの依存関係をアークとし，各述語をノードとする有向グラフとして表現することができる。この依存グラフを強連結成分分解する。分解結果は求めるべき最終解，すなわちゴールを根ノードとし外延データベースに対応する述語を葉ノードとする木になる。各成分間の依存関係は半順序的であり，基本的には個々の成分を葉ノード側から順に評価すれば解を得ることができる。したがって，この分解により以後のHCQに対する処理は各成分毎に独立に行うことが可能となる。

また，ホーン節変換後のHCQに対して強連結成分分解を行うと各成分は，再帰述語の数，再帰が線形であるかどうか，他の成分の述語に依存するかどうかにより幾つかの限られたパターンに分類できる。

(4) 内部問い合せの生成

まず，上記強連結成分毎に問い合せの効率化を行う。PHIでは，HCQ全体としてはボトム・アップとトップ・ダウンを融合した方式による制約条件の伝播を基本としている。これと，上述した各強連結成分のパターンに基づいた，個々の成分に対する効率化を組み合せることにより問い合せの効率化を行う[宮崎87]。この効率化により，HCQはホーン節形式の範囲で制約を最初に行い限定された空間で探索を行う形式になる。

次に葉ノードに対応する強連結成分，すなわち外延データベースに格納されているリレーション，に対応する述語に対し，そのリレーションの格納サイト情報を付加し内部問い合せとする。

3．2．ローカル知識ベース・マネージャ

ローカル知識ベース・マネージャはGKBMから与えられた内部問い合せの内，自分に関連する部分のみを自律的に処理する。以下LKBMの処理について述べる。

(1) 関連成分の識別

LKBMに与えられる内部問い合せは強連結成分に分解されおり，各成分は外延データベースに対する処理，またはLFP演算に関して1つのまとまった処理単位になっている。したがって，各LKBMはこの成分毎に自分が実行する可能性のある成分を識別する（識別された成分を関連成分と呼ぶ）。

この識別は次の方針に従って行われる。

(a) 分散環境においては，サイト間のデータ転送コストが全体の処理コストに大きな影響を与えるため，外延データベースに対する制約処理は格納サイトで行う。したがって，自サイトに格納されているリレーションに対応する述語が存在する成分は関連成分である。

(b) LFP演算結果のリレーションの大きさを予測することは困難であるため，LFP演算を実行する実行サイトは動的に決定する。したがって，依存グラフ上で自分が関連する成分に到達可能な成分は関連成分である。

(2) ローカル・プランの生成

各LKBM毎のデータ・アクセス・プラン（ローカル・プランと呼ぶ）を生成する。まず，関連成分に対してLFPを求める繰り返し戦略を決定し[宮崎87]，次に各成分に対して関係代数演算列を生成する。ホーン節形式では各ルールのボディ中における述語の順序及びルール自身の順序に意味はないが，これらの順序は関係代数演算列では二項演算の実行順序に対応する。したがって，これら関係代数演算の実行順序の最適化も合わせて行う。

(3) ローカル・プランの実行

(2)で生成されるローカル・プランは，依存グラフの個々の強連結成分に対応する部分問い合せの集合であり，上述したように半順序関係がある。各LKBMはこの順序に従って部分問い合わせを処理する。繰り返しを必要とする部分問い合せに対しては，その問い合せに関連するLKBMどうしが協調して処理の実行サイトを決定する[吉田86]。

4．おわりに

分散知識ベース・システムPHIの問い合せ処理モデルについて述べた。PHIに与えられるホーン節問い合せは，1つのグローバル知識ベース・マネージャと分散サイトに対応する複数のローカル知識ベース・マネージャによって処理される。

今後はPHIのプロトタイプの試作・評価をPSIにESPを用いて行うと共に，知識の一貫性制御方式等の検討を行う予定である。

《参考文献》

[MIYA86] Miyazaki, N., et al., "Compiling Horn Clause Queries in Deductive Databases:A Horn Clause Transformation Approach," ICOT Technical Report, TR-183, 1986.

[伊藤86] 伊藤他, "KBMS PHI(1) 分散知識ベースシステムのシステム構成方式," 情報処理学会第32回全国大会, 2M-5, 1986.

[宮崎86] 宮崎他, "KBMS PHI(2) 知識とデータに関する一考察," 情報処理学会第32回全国大会, 2M-6, 1986.

[宮崎87] 宮崎他, "KBMS PHI:問い合せ処理の効率化," 情報処理学会第34回全国大会, 3K-6, 1987.

[吉田86] 吉田他, "KBMS PHI(3) 分散知識ベース制御方式," 情報処理学会第32回全国大会, 2M-7, 1986.

KBMS PHI：問い合せ処理の効率化

3K-6

宮崎 収兄*, 羽生田 博美*, 大場 雅博**
*:沖電気工業(株), **:(財)新世代コンピュータ技術開発機構

1．はじめに

分散知識ベース・システムPHIの問い合せ処理モデルに基づく問い合せ処理の効率化について報告する。まず，分散型システムとしての観点から，従来の効率化方式とその問題点について述べ次いでボトム・アップ方式とトップ・ダウン方式を融合した方式を基本方式とするPHIの効率化方式について述べる。

2．知識ベース処理における効率化と問題点

本章では分散型知識ベース・システムにおける問い合せ処理方式の効率化と，その問題点について述べる。

2．1．コンパイル・アプローチにおける効率化

分散知識ベース・システムPHIは，分散型の知識ベース・システムであり，その問い合せ処理方式はコンパイル・アプローチ[HENS84, MIYA86]に基づいている[羽生87]。このような観点から見ると，PHIの特徴は次のように要約することができる。

(1) 問い合せ処理によるデータへのアクセス戦略の生成とその実行とは別々のフェーズに分離されている。
(2) データへのアクセス戦略は各LKBM（分散サイト）毎に自律的に決定および実行される[吉田86]。
(3) 分散環境においては各サイト間のインタラクションは処理コストに大きな影響を与える。

ここで演繹問い合せの処理方式を考えると，大きく分けてトップ・ダウン方式とボトム・アップ方式とがある[BANC86]。データベースに存在するリレーション側から順次部分解のリレーションを組みたてて行き最終解を得るのがボトム・アップ方式，逆に問い合せ側からルールを展開しながら最終的解を求めるのがトップ・ダウン方式である。

PHIのような分散型システムでは，トップ・ダウン方式を基本にした問い合せ処理方式を採用すると，不必要なサイト間のデータ転送およびインタラクションが増加し，処理コストを増大させる可能性がある。また，問い合せの効率化が複雑で困難になる可能性もある。したがって，分散環境におけるコンパイル・アプローチに基づくシステムでは，これらの欠点が発生しないボトム・アップ方式を基本とした問い合せ処理方式が望ましいと考えられる。

ボトム・アップ方式では，問い合せは互いに依存関係にある部分問い合せから構成されると考えることができ[CERI86, 羽生87]，問い合せ処理の効率化の問題は与えられた制約条件をいかに部分問い合せに伝播させ，各部分問い合せではその制約条件をいかに効率的に処理するかという問題に帰着する。内包データベースが再帰を含まない場合は，通常の関係代数の最適化と同様の方式が適用できる。再帰を含む場合にも，各部分問い合せに関して最小不動点（LFP）演算子と制約条件が可換な時には[AHO79, MIYA86]で述べられている方式が適用できる。また，非可換な場合でも再帰が単純であれば[CERI86, MIYA86]で述べられている方式が可能である。

2．2．ボトム・アップ方式における効率化の問題点

前節で述べた，ボトム・アップ方式における効率化の要点は，(a)制約条件の伝播をどこまで行うことが可能か，(b)再帰的部分問い合せに対して制約条件をどこまで利用することが可能かの2点である。この問題を例で考える。図1に内包データベースと問い合せの例を示す（表記法はPrologに準拠する）。外延データベースには内包データベース中の各e1に対応するリレーションが存在すると仮定する。

```
?-q(A, constant).
q(A, B):-e1(A, B).
q(A, B):-r1(A, C), q(C, B).
r1(A, B):-e2(A, B).
r1(A, B):-e2(A, C), r1(C, B).
```

図1　十分な効率化が困難な問い合せの例

このような問い合せはボトム・アップ方式では，まず述語r1(A, C)に対応するリレーションを求め，次にこれを用いてq(A, constant)に対応するリレーションを求めることになる。このときqに対する制約条件B=constantは，q自身が再帰定義されているのでr1まで伝播させることができず，r1に対する探索は全解探索となり効率的ではない。

また，qが他の述語と相互再帰的に定義されており，LFP演算と制約条件が可換ではない場合には，qに対して全解探索を行わなければならず効率的な処理ができない。

このように，(a)再帰述語がさらに他の再帰述語に依存している場合には制約条件の伝播を十分に行うことができない，(b)1つの部分問い合せに関しても，再帰が相互再帰になる場合には制約条件を探索に利用できないという問題が発生する。

3．PHIにおける問い合せ処理の効率化

PHIでは前章で述べた効率化の問題点に対して，(a)ボトム・アップとトップ・ダウンを融合した方式を基本とし，(b)各部分問い合せに対しては前節で述べた効率化と制限子付最小不動点（RLFP）演算による効率化とにより対処している。本章ではこのPHIにおける効率化の基本方式とRLFP演算方式について述べる。

3．1．ボトム・アップ方式とトップ・ダウン方式の融合

2章で述べた第一の問題点は，部分問い合せ間で制約の伝播を十分に行えないことであった。例えば，図1ではqに対する制約がr1まで伝播していない。この問題を図2の例で考える。

KBMS PHI: Query Processing Strategy
Nobuyoshi MIYAZAKI*, Hiromi HANIUDA*,
Masahiro OHBA**
*Oki Electric Industry Co., Ltd.
**Institute for New Generation Computer Technology

```
ancestorFriend(Someone, AF):-
    friend(Someone, AF).
ancestorFriend(Someone, AF):-
    ancestor(Someone, A), friend(A, AF).
ancestor(Someone, A):-
    parent(Someone, A).
ancestor(Someone, A):-
    parent(Someone, P), ancestor(P, A).
```

図2　ancestorFriendの定義

ボトム・アップ方式では，ancestorFriendに対する問い合せはancestorに対する部分問い合せと，ancestorFriend自身に対する部分問い合せとから構成される。Someoneに関する制約条件は，単純にancestorまで伝播させることができるが，AFに関する制約条件は，そのままではancestorまで伝播させることはできない。しかし，Aの取りうる値{a|friend(a, constant)}を与えられれば，これを制約条件としてancestorまで伝播させることができる。

PHIではこれを，"SomeoneまたはAに関して，その取りうる値または値の集合を与えられてancestorに対する部分問い合せを解く"問題，として統一的に考える。

これは，Someoneに関する制約に対しては，?-ancestor(constant, A)を解き，次に，?-ancestorFriend(constant, AF)を解くという完全にボトム・アップ的方式になる。また，AFに関する制約に対しては?-friend(A, constant)を解いてから?-ancestor(Someone, A), A∈{a|friend(a, constant)}を解くというトップ・ダウン的方式となる。またancestorFriend自身が再帰的である場合や，複数の部分問い合せに依存している場合にも，ancestorFriendが依存する部分問い合せを，順次トップ・ダウン的に解くことにより，制約条件を伝播させることができる。

ここで，ancestor自身に対する問い合せは，制約条件の形式にかかわらず，2.1節で述べた方式や次節で述べるRLFP方式により効率的に処理することができる。

したがって，PHIではこの融合方式を基本方式とすることにより，問い合せを効率的に処理することができる。

3．2．制限子付最小不動点演算方式

2章で述べた第二の問題点は，各部分問い合せに関して場合によって制約条件を探索に利用できないことであった。この問題は再帰述語が相互再帰の場合や，再帰述語の定義ルールのボディ中にその再帰述語が複数回現われる場合等に発生する。

PHIではこの問題に対して，制限子付最小不動点演算を導入することにより対処する。図1のancestorに対する問い合せ?-ancestor(jiro, A)を例に考える。

各ancestorを定義するルールのボディ部は，ancestorの探索空間を限定していると考えることもできる。したがって，各ancestorの定義ルールのボディ部に，探索空間をさらに限定する節を新たに追加することにより，部分問い合せancestor(jiro, A)を効率化できる。すなわち，ancestorの定義ルール集合に対して，ancestorの探索空間を限定する節ancestor*を導入することにより図3のように修正する。

図3におけるancestor*の定義ルールは，元のancestor定義節のボディ中のancestorに対して，他の述語が取りうる値を与えることにより，ancestorの値の範囲を限定するように定義されている。また，ancestor自身はancestor*により探索空間が狭められている。このancestor*をancestorの制限子と呼び，制限子を導入することにより最小不動点を求める演算を制限子付最小不動点（RLFP）演算と呼ぶ。RLFP演算については別途報告する予定である。

```
ancestor*(Someone, A):-
    ancestor*_init(Someone, A).
ancestor*_init(jiro, A).
ancestor*(P, A):-
    ancestor*(Someone, A),
    parent(Someone, P).
ancestor(Someone, A):-
    ancestor*(Someone, A),
    parent(Someone, A).
ancestor(Someone, A):-
    ancestor*(Someone, A),
    parent(Someone, P), ancestor(P, A).
```

図3　制限子の導入例

PHIでは，各部分問い合せに対して，第2節で述べたような従来の効率化が不可能な場合，制限子によって探索空間を制限することにより問い合せ処理を効率化する。

4．おわりに

分散知識ベース・システムPHIの効率化方式について述べた。PHIでは，全体としてボトム・アップとトップ・ダウンを融合した方式を基本とし，これと各再帰的部分問い合せに対する効率化，すなわちLFP演算と制約条件の可換性に基づく効率化等従来の方式と制限子付最小不動点演算による方式，を組み合せることにより問い合せ処理の効率化を行う。これにより，PHIではすべての問い合せパターンに対して戦略決定の問題として対処することができる。

今後は本稿で述べた方式の詳細部分の検討，およびその有効性をプロトタイプ・システムで確認していく予定である。

《参考文献》

[AHO79] Aho, A. V., Ullman, J. D, "Universality of Data Retrieval Languages," 6th ACM Symp. on Principles of Programming Languages, pp. 110-120, 1979.

[BANC86] Bancilhon, F., Ramakrishnan, R., "An Amateur's Introduction to Recursive Query Processing Strategies, "ACM SIGMOD'86, pp. 16-52, 1986.

[CERI86] Ceri, S., et al., "Translation and Optimization of Logic Queries:The Algebraic Approach," VLDB86, pp. 395-402, 1986.

[HENS84] Henschen, L., Naqvi, S., "On Compiling Queries in Recursive First-Order Databases," JACM Vol. 31, No. 1, pp. 47-85, 1984.

[MIYA86] Miyazaki, N., et al.,"Compiling Horn Clause Queries in Deductive Databases:A Horn Clause Transformation Approach," ICOT Technical Report TR-183, 1986.

[羽生87] 羽生田他, "KBMS PHI:問い合せ処理モデル", 情報処理学会第34回全国大会, 3K-5, 1987.

[吉田86] 吉田他, "KBMS PHI(3)分散知識ベース制御方式," 情報処理学会第32回全国大会, 2M-7, 1986.

3K-7

KBMS PHI : 重ね合せ符号を用いた関係演算の処理方式

和田 光教*, 山下 祥司*, 山崎 晴明*, 森田 幸伯**
*:沖電気工業(株), **:(財)新世代コンピュータ技術開発機構

1. はじめに

現在，筆者等は一階述語の枠組みを持つ分散知識ベース・システムPHIを構築中である[伊藤86]。PHIは階層構造を持ち，上位階層にはホーン節による問い合せを関係演算に変換する機能を有し，下位階層はデータベース管理層とし上位階層が与える関係演算を実行する。これによって，PHIはアプリケーションに対してホーン節によるインタフェースを提供する一方で，関係データベースの枠組みを用いて知識を管理することを可能としている。PHIでは，大量のファクトを格納することを想定しており，その処理を高速に実行することが望まれる。

既に，筆者等は重ね合わせ符号を項のインデクスとして用いることで，ある項と単一化可能な項を高速に検索できることを示した[森田86]。

本稿では，PHIが大量のファクトをリレーションとして格納することに注目し，重ね合わせ符号をリレーションのインデクスに適用させる方式について述べる。これは，タプルの高速な検索を可能とすると共に，PHIの上位階層が生成する関係演算の高速な実行も可能とするものである。

2. 重ね合わせ符号を用いた検索方式

大量のテキストデータを検索する手法として，重ね合わせ符号を用いた検索方式（本稿ではSCW方式と呼ぶ）が提案されている[ROBE79, 有川82, 有川83]。SCW方式はデータレコードに対し重ね合せ符号（superimposed code word）を用いたインデクスレコードを割り当て，大容量データに対しての部分照合（paritial match retrieval）による高速検索を実現したものである。

N個のレコード集合からなるファイル$F=\{R_1, \cdots R_N\}$を考える。レコードRiはri個のキーワードを持つものとする。各キーワードはハシュ関数により2進符号語に写像される。Riのri個の2進符号語をビットごとにORをとったものをRiに対する重ね合わせ符号Siと呼ぶ。Fの全レコードに対して重ね合せ符号を作り1つのファイルを構成し，SCWファイルSと名付ける。そして，任意のSiを指定すれば対応するRiを特定できるようにSを構成しておく。

ここでrq個のキーワードを指定した検索を考える。この検索では，指定されたキーワードの各々について先のハシュ関数を用いて2進符号語を生成しそれのORをとる。これはキュエリマスクQと呼ばれる。QとSiとの間で条件(※)を満足しなければ，対応するレコードはキュエリを満たさない。

$$Q \wedge Si = Q \quad \cdots\cdots\cdots \quad (※)$$

Fからキュエリqを満たすタプルを全て検索するには，次の2段階を経て処理は実行される。まず，Sから(※)を満たすSiの集合Σを抽出する。Σの各要素Siに対応するレコードRiがキュエリを満たす候補となる。次に，候補レコードの内容を吟味し，キュエリで指示された条件を満たすタプルのみを選び出す。当然，候補レコードの数は重ね合せ符号の設計に左右される。

KBMS PHI:Using Superimposed Codes for Relational Algebra Operations
Mitsunori WADA*, Shouji YAMASHITA*, Haruaki YAMAZAKI*, Yukihiro MORITA**
*Oki Electric Industry Co., Ltd.
**Institute for New Generation Computer Technology

3. SCW方式を用いた関係演算の実現方式

筆者等が現在構築中の分散知識ベース・システムPHIでは，大量のファクトを関係データベースとしてもつことを想定している。そのため，知識処理向けの演算（検索，集合演算，包含関係の検査，結合演算等）が高速に実行されることが望まれる。

前節の手法を適用して，リレーションに対して重ね合せ符号を用いたインデクス（SCWインデクス）を作成しておくことで，これを用いた大量のファクト集合に対しての高速な検索が可能となる。更に，重ね合せ符号の構成法に制約条件を付与することで，集合演算や包含関係の検査が高速に実現できることを示す。

1) インデクスの作成法

前節でレコードをタプル，ファイルをリレーション，キーワードをキー属性と読み替えることで，そのままリレーションに対してSCWインデクスを作成することができる。1つもしくは複数個の属性値を指定し，その値を満たすタプルを全て検索するキュエリを与えれば，このキュエリからキュエリマスクを生成し，条件を満足する全てのタプルを抽出できる。

[ROBE79]によれば，重ね合せ符号ではそのセットされているビットとリセットされているビットの数が等しくなるようにインデクスレコードを構成すると，実際にはキュエリを満たさないがキュエリマスクを満足するレコード数の期待値（N）が最小となる。キーワードを用いてテキストレコードを検索するシステムでは，テキストレコード毎にそのキーワード数は異なるが，リレーションの場合にはスキーマについてキー属性が与えられているので，タプル毎のキー属性数（及び種類）は同一である。このためハシュ関数を巧く設計すればインデクス値の分布を制御でき，先のNの値を小さくすることが容易である。このような点からも重ね合せ符号を用いたインデクスはリレーションの検索に適している。

2) 集合演算（集合和，集合積，集合差）および包含関係の検査の実現

ここでの，包含関係の検査とは，2つのリレーションR_1, R_2が与えられた際に，R_1がR_2を含んでいるか否かを調べる処理である。集合演算および包含関係の検査では，別々のリレーションに含まれている2つのタプルが同一値であるか否かを判定する必要がある。しかし，タプルの比較は属性毎に行われるので大きな処理コストとなる。そこで，タプルの値が同一であるか否かをSCWインデクスに反映させることで，各タプル対応にインデクス値（2進値）のみの比較で済ませ処理コストの低減を計る。次の条件1）を満足するようにインデクスを構成すれば，「別々のリレーションにおいて同一値を持つ2つのタプルは，それぞれに対応するSCWインデクスレコードの値が必ず一致する」ことが保証される。

条件1）キー値から2進符号語を作成する際に，同一の属性については同一のハシュ関数を用いる。

以下に比較の手順を示す。まずインデクスの値どうしで比較を行い，その値が一致するか否かによりその処理は分岐する。

・インデクスレコードの値が一致しない
⇒各々が指すタプルは同一値を持たない。

◦インデクスレコードの値が一致する
⇒各々が指すタプルが同一値を持つ可能性を判定できないため，各々が指すタプルどうしでの比較を実行するか，もしくはその履歴をとっておきインデクスどうしでの比較が終了してからタプルどうしでの比較を行うこととなる。

m個のタプルからなるリレーションとn個のタプルからなるリレーションの間で，タプルどうしで比較を実行すると$O(m \times n)$回の比較を行うこととなる。一方，上記の手順でSCWインデクスを用いた比較を実行すると，タプルどうしで比較したのと同じ回数のインデクスの比較に加え何回かのタプルの比較を行うこととなり，1回当りの処理コストは低減されても処理回数はむしろ増加している。そこで，次の条件2）を満足するようにインデクスを構成するとインデクスレコードの比較回数が減少する。

条件2）全てのインデクスについて，インデクスレコードはその値に関して昇順にならべておく。

この条件を満足するインデクスでは，m個のインデクスレコードとn個のインデクスレコードを比較する際には，各インデクスで先頭から末尾へと逐次アクセスしながら$O(m+n)$回の比較を実行するだけで済む。図1．に3タプル×3タプルでのINTERSECTIONではインデクスについては5回の比較で済むことを示す。ここで，タプルどうしでなら9回の比較が必要である。

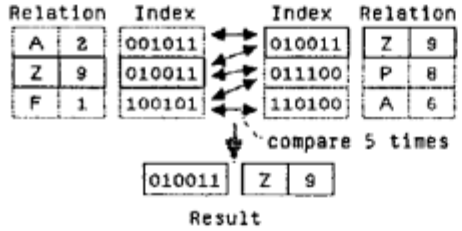


図1．INTERSECTIONの実行例

4．他方式との比較

まず，検索を実行する場合についてSCW方式とB⁺treeやハシュ関数による方式とで，1属性を指定して検索を行う場合と複数属性を指定して検索を実行する場合とに分けて比較する。ここでは，リレーションおよびインデクスはともに2次記憶上に格納されているものとする。

1) SCW方式

SCW方式ではインデクス値を全て調べているので，インデクス全体を1次記憶上に転送し，キュエリマスクと比較する必要がある。この時間コストは指定した属性の個数にかかわらず一定である。更に，インデクスにより絞り込んだ一部のタプルについてもその内容を吟味する必要があるが，吟味するタプルの個数は多くの属性を指定するに従って少なくなる傾向にある。

2-1) 他方式（1属性を指定した検索）

B⁺treeを用いて検索を実行しようとするとインデクスの一部についてのみ転送するだけでキュエリを満たすタプルは求まり，また，ハシュ関数を用いた検索では，候補となるタプルを抽出しその内容を吟味する処理だけで済み，SCW方式に比べてその実行コストは小さい。

2-2) 他方式（複数個の属性を指定した検索）

一般に，1属性に注目してキュエリを満たす候補となるタプルを抽出した後，その内容を吟味してキュエリを満足するタプルのみを残すという手順を踏む。この方式ではSCW方式による場合に比べ候補タプルの絞り込みは弱く，その実行コストはSCW方式に比べ大きくなってゆく。これは，指定する属性の個数が多くなるほど顕著となる。

次に，集合演算や包含関係の検査を実行する場合について比較する。ここで，m個のタプルとn個のタプルを持つ2つのリレーションの間での処理を考える。

1) SCW方式

インデクスを昇順に構成しておけば，$O(m+n)$回のインデクスの比較とおよびk(≪m，n)回のタプルの比較で済む。更に，インデクスは複雑なデータ構造を持たないので，集合演算の実行に伴ってその実行結果についてのSCWインデクスを作成していくことが可能である。図2．にUNIONの実行結果に対しインデクスを与えた例を掲げる。左側の2つのリレーションにUNIONを施した結果が右側のリレーションである。ここで，左側のインデクスを流用して右側のインデクスが作成されている。

2) 他方式

ハシュ関数を用いた方式では処理にインデクスを用いることができないため，$O(m \times n)$回のタプルどうしでの比較が必要である。一方，B⁺treeを用いた場合には，1属性に注目して同一値を持つタプルの候補を抽出した後，その内容を吟味して実際に同一値を持つタプルを取り出すため，比較回数はSCWインデクスを用いた方式とハシュによるものの間となる。更に，ハシュ，B⁺treeいずれの方式でも集合演算の実行結果に対してインデクスが必要となると，実行結果から改めてインデクスを作成することとなる。

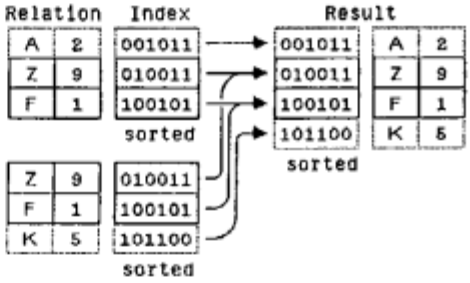


図2．UNIONの実行例

5．終りに

重ね合せ符号が関係データベース向けインデクスに適用できることを示した。現在，筆者等はSCWインデクスの比較ハードウェアを持つPHI用の関係演算用エンジンを設計中である。今後の課題としては，最適なハシュ関数やインデクスレコード長の選択に関する問題が残されている。

《参考文献》

[ROBE79] Roberts, C.S., "Partial-Match Retrival via the Method of Superimposed Codes," Proc. IEEE, Vol.67, No.12, pp.1624-1642, 1979

[有川82] 有川 他，"重ね合せ符号と逐次サーチを利用する文献情報検索システムについて，" 情報処理学会第25回全国大会，3P-10，1982

[有川83] 有川 他，" 重ね合せ符号と逐次サーチを利用する文献情報検索システムについて(2)，"情報処理学会第27回全国大会，7K-8，1983

[伊藤86] 伊藤 他，"KBMS PHI(1) 分散知識ベースシステムのシステム構成方式，" 情報処理学会第32回全国大会，2M-5，1986

[森田86] 森田 他，"スーパーインポーズドコードを用いた構造体の検索方式，" 情報処理学会第33回全国大会，6L-8，1986

3K-8

KBMS PHI：演繹データベース実験システムPHI/K²

阿比留 幸展*, 羽生田 博美*, 宮崎 収兄*, 森田 幸伯**
*：沖電気工業(株), **：(財)新世代コンピュータ技術開発機構

1. はじめに

分散知識ベース・システムPHIにおける問い合せ処理の基本方式は[MIYA86]で報告した。この処理方式が有効であることを確認するため，演繹データベース実験システムPHI/K²(PHI Knowledge Kernel)をDEC2060上に試作した。本稿では，PHIの中核となるアルゴリズムの有効性および実現性を示す。

2. 実験システムPHI/K²

PHI/K²は，PHIにおける問い合せ処理方式のアルゴリズムの有効性を確認するためのシステムであり，DEC10-Prologにより記述されている。システム構成を図1に示す。

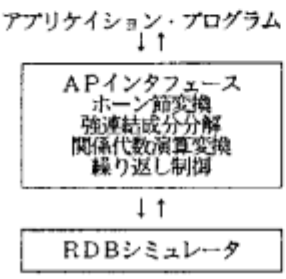


図1 PHI/K²のシステム構成

PHI/K²とAPとのインタフェースはホーン節問い合せ，関係データベース(RDB)シミュレータとのインタフェースは関係代数演算である。PHI/K²はAPからの最初のアクセスに対しては，RDBを用いて問い合せに対する解を集合として求め，APへはバックトラックによりタプル単位に返す。

このシステムでは，PHIの問い合せ処理の核となる以下の特長を実現した。

(1) 複雑な相互再帰を含むルールの処理の実現
(2) ホーン節変換による再帰述語の検出およびルールの簡略化による問い合せ処理の効率化[伊藤86]
(3) 再帰述語が複数の場合の，再帰述語の分解による繰り返し演算の効率化
(4) 再帰述語が唯一つの場合の，制約条件の埋め込みによる繰り返し演算の効率化
(5) 論理型言語からの関係データベース機能の利用

本稿では，(2)～(4)の実現方法を述べる。

2. 1 ホーン節変換

ホーン節変換(HCT)は，与えられたゴールに対しルールを簡略化する等価変換であり，これにより以後の処理の効率化を図るものである。HCTではまず再帰述語を検出する。次にルールを部分評価により，データが関係データベース中に存在することを示す述語(edb述語)および再帰述語のみを含む形にする。

再帰述語検出および変換には，Prolog処理系に部分評価機能を持たせるためにメタプログラミング手法を用いている。図2は変換プログラムのメイン部分で，あるヘッドを与えたときに，変換されたボディ部を出力する。述語rel_memberは，述語と述語集合が与えられたとき，述語集合の中に同じ述語名の述語が存在するかどうかをチェックするための述語である。また，述語tr_quelの第3引数は，変換後のボディ部に相当する。記号'::'は，差分リストを表わす。この処理は，すべてのルールのヘッド部について行われる。

```
tr_quel(Idb, edb(P), [Q|X]::X, Rec):-!, P=..Q.
tr_quel(Idb, (A;B), F, Rec):-!,
    (tr_quel(Idb, A, F, Rec);tr_quel(Idb, B, F, Rec)).
tr_quel(Idb, (A,B), X1::X3, Rec):-!,
    tr_quel(Idb, A, X1::X2, Rec),
    tr_quel(Idb, B, X2::X3, Rec).
tr_quel(Idb, Body, [Q|X]::X, Rec):-
    Body=..Q,
    rel_member(Q, Rec), !.
tr_quel(Idb, Body, X, Rec):-
    clausekb(Idb, Body, Next_body),
    tr_quel(Idb, Next_body, X, Rec).
clausekb(KB, P, Q):-X=..[KB, (P:-Q)], X.
clausekb(KB, P, true):-X=..[KB, P], X.
```

図2 変換部のメイン部分

2. 2 強連結成分分解

ルール中の述語間の依存関係を表す依存グラフは，強連結成分分解することにより互いに依存しあう述語集合を1つのノード(強連結成分)とする木に分解することができる[羽生87]。これにより，成分の中で閉じた演算の実行による繰り返し演算の効率化や，関係代数演算の効率化戦略の決定が可能になる。PHI/K²では再帰述語に関する依存関係のみを考慮して，上述した強連結成分分解を行う。

処理は以下の手順で行われる。

(1) ルールおよび再帰述語リストからアークを生成する（ゴールと再帰述語に対してのみアークを生成する）
(2) アークからパスを生成する
(3) パスから強連結成分を生成する
(4) 各強連結成分に対して成分間の依存関係による順序付けを行う

2. 3 制約演算の可換性を用いた繰り返し演算の効率化

関係代数演算変換では，問い合せ中に制約条件がある場合に，制約演算を生成する。繰り返し演算と制約演算は，可換になる場合があり，制約演算を先に実行することにより計算量を大幅に削減できる。

この可換性の判定は，述語間の一般性の比較により行う。ルールが再帰のとき，ヘッド部の述語が，ボディ部の同じ述語名の述語より一般的であれば可換である。

図3は，第一引数が第二引数より一般的であるかどうかをチェックするプログラムである。このプログラムは比較する述語をリストで表わしている。第一引数で定数が現われたら，第二引数でも同じ場所に同じ定数が現われ，かつ第一引数の変数間に等号関係があれば，第二引数でも満たしているかどうかを調べる。ただし，このアルゴリズムは，再帰述語が唯一つの場合であり，複数の再帰述語がある場合は考慮していない。

KBMS PHI:An Experimental Deductive Database System PHI/K²
Yukihiro ABIRU*, Hiromi HANIUDA*,
Nobuyoshi MIYAZAKI*, Yukihiro MORITA**
*Oki Electric Industry Co., Ltd.
**Institute for New Generation Computer Technology

```
more_general([Head|_],[Bhead|_]):-Head  == Bhead,!.
more_general([_|Harg],[_|Barg]):-!,
    const_check(Harg,Barg),
    eq_check(Harg,EQ_List),
    check_eq_cnd(Barg,EQ_List).
const_check([],[]):-!.
const_check([H1|Hrest],[B1|Brest]):-!,
    (var(H1) -> true;H1==B1),
    const_check(Hrest,Brest).
```

図3　述語間の一般性をチェックするプログラム

3．実行例

2種類の例を用いて，PHI/K²を評価する。再帰述語が一つで，制約演算に関して可換な場合と，再帰述語が複数で，強連結成分が複数存在する場合の処理例を示す。これにより，可換な場合に制約演算を先に実行することによる効率化，および強連結成分分解による繰り返しの効率化を評価する。

（例1）　再帰述語が唯一つで，制約演算と繰り返し演算が可換な場合

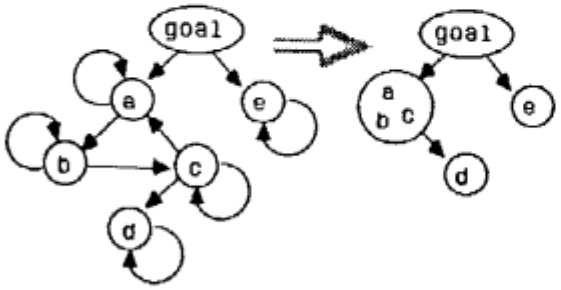
ホーン節問い合せ ?-ancestor(X,mayumi). をPHI/K²に適用するときは，retrieveという述語を用いて，ancestorに関するルールが格納されているidbを指定する。

この問い合せは制約演算に関して可換である。したがって，制約演算を先に実行してから，繰り返し演算を実行する。図4に制約演算が可換なルールについての実行例を示す。

```
もとのルール
idb1((ancestor(X,Y):-parent(X,Y))).
idb1((ancestor(X,Y):-parent(X,Z),ancestor(Z,Y))).
idb1((parent(X,Y):-father(X,Y))).
idb1((parent(X,Y):-mother(X,Y))).
idb1((father(X,Y):-edb(father(X,Y)))).
idb1((mother(X,Y):-edb(mother(X,Y)))).

| ?- retrieve(idb1,ancestor(X,mayumi)),time.

ancestor(X,mayumi):-mother(X,Z),ancestor(Z,mayumi).
ancestor(X,mayumi):-father(X,Z),ancestor(Z,mayumi).
ancestor(X,mayumi):-mother(X,mayumi).
ancestor(X,mayumi):-father(X,mayumi).

3873
X = etsuko
yes
```

図4　制約演算が可換なルールについてのPHI/K²の実行

上記例は，曾孫までを含む親子関係のタプル数が１４という条件で実行している。同じ条件で，制約条件を先に実行せずに繰り返し演算を実行すると，11025msecかかった。したがって，制約条件を先に実行すると，タプルが１４しかない場合でも，約3倍速くなったことがわかる。



［分解前］　　［分解後］

図5　強連結成分分解を行う前と行った後のルールのグラフ

両親に対し，子が一人という二分木を仮定し，k番目までの先祖を求める場合には，制約条件を先に実行した場合と後から実行した場合の繰り返し演算の解となるタプル数の比は，最低で$(2^{k+1}-2):((k-1)2^{k+1}+2)$，最高で$1:(k-1)2^{k+1}+2)$の比になる。タプルの数は$2^{k+1}-2$であるから，タプルの数をnとすると，処理時間が中間解の大きさに比例すると考えても最低log n，最高n·log nのオーダーで違ってくる。

（例2）　問い合せが複数の成分からなる場合

再帰述語が複数ある場合には，繰り返し演算を，各強連結成分に閉じて行う。分解前と分解後のルールのグラフ表現を図5に，問い合せ処理を行い，分解されたルールを図6に示す。

```
| ?- retrieve(idb2,goal(X,Y)),time.

d(X,Y):-x(X,Z),d(Z,Y).
d(X,Y):-y(X,Y).

e(X,Y):-u(X,Z),e(Z,Y).
e(X,Y):-u(X,Y).

a(X,Y):-b(X,Z),a(Z,Y).
a(X,Y):-z(X,Y).
b(X,Y):-c(X,Z),b(Z,Y).
b(X,Y):-y(X,Y).
c(X,Y):-z(X,Z),d(Z,Y).
c(X,Y):-a(X,Z),c(Z,Y).
c(X,Y):-x(X,Y).

goal(X,Y):-e(X,Y).
goal(X,Y):-a(X,Y).

13866
X = a,
Y = b
yes
```

図6　問い合せが複数の成分からなるルールの実行

上記例において，最小不動点(LFP)を求める繰り返し演算は，初めに[d]のLFPを求め，次に[e]のLFPを求めてから，[a,b,c]の同時最小不動点(SLFP)を求める。LFPまたはSLFPを求めた後の述語は，edb述語と等価に扱うことができる。したがって，各成分ごとにSLFPを求めるほうが，[a,b,c,d,e]全体のSLFPを求めるよりも繰り返し演算が効率的である。PHI/K²では分解により繰り返しを分けることしか行っていないが，PHIでは分解に基づき[宮崎87]で述べたような成分間の制約条件の伝播を考慮した効率化を行う予定である。

4．おわりに

演繹データベース実験システムPHI/K²は，演繹データベースの問い合せ処理の核部分の実験システムであり，基本的な部分は前章で見たように実現されたと言える。今後は否定述語（not）が入った場合のルールの扱い方，および分散環境における効率化[羽生87,宮崎87]を考慮して，PSI上でシステムの構築を行う予定である。

《参考文献》

[MIYA86]　Miyazaki, N., et al., "Compiling Horn Clause Queries in Deductive Databases:A Horn Clause Transformation Approach," ICOT TR-183, 1986.

[伊藤86]　伊藤他，"知識ベースシステム-KBMS PHI，" 61年電気・情報関連学会連合大会，32-5

[羽生87]　羽生田他，"KBMS PHI:問い合せ処理モデル，" 情報処理学会第34回全国大会，3K-5，1987.

[宮崎87]　宮崎他，"KBMS PHI:問い合せ処理の効率化，" 情報処理学会第34回全国大会，3K-6，1987.

3K-9

KBMS PHI：例外のある性質継承の表現

宮崎収兄*，　坂間千秋**，　伊藤英則**
*:沖電気工業(株)，**:(財)新世代コンピュータ技術開発機構

1．はじめに

第5世代コンピュータ・プロジェクトの一環としてのKBMS PHIの研究では知識ベース管理システムを知識管理層とデータベース管理層の2つの階層から構成するモデルを採用している。本稿ではこのようなシステムでの論理型知識表現のありかたについて例外のある性質継承の表現を例として考察する。

2．KBMS PHIと知識表現

知識ベース管理システム（KBMS）は知識情報処理システムにおいて推論エンジンや応用プログラムの必要とする知識を供給するとともに知識獲得時の整合性の管理を行う。KBMS PHIでは中ないし大規模の知識ベースを効率良く管理するためのアルゴリズムやアーキテクチャの研究を行っているが，モデルとしては以下のようなアプローチをとっている[宮崎86]。

(1) 一階述語論理に基づく知識表現を基本とする。
(2) 大量知識管理のためデータベース技術を用いる。
(3) ファクト型知識の処理にはデータベース演算を用い，ルール型知識の処理は演繹推論とデータベース演算を基に行う。
(4) ファクト型知識を管理するデータベース管理層とルール型知識を管理する知識管理層の2階層からシステムを構成する。

このようなアプローチをとったのは，関係データベースにおける演算が一階述語論理により基礎づけられ演繹推論の1つの実現方式であるとみなせるためである。このような階層構成をとることにより大量のファクト型知識をデータベース演算により効率良く処理できる。これに対しルール型知識はファクト型知識では記述できない抽象度の高い知識やファクト型知識より推論できる知識の記述，手続き型知識，メタ知識等からなる。これらの処理はデータベース演算への変換や手続きごとのプロシージャによってなされる。現在知られている方式ではルールに対するデータベース演算の適用はセレクションに限定されている。従ってPHIのような演繹データベース技術を用いたシステムで知識を効率良く処理するためには知識をできるだけファクトで記述することが望ましい。逆に知識の大部分がルール型でしか記述できないならデータベース技術を知識ベースに応用するアプローチの有効性は限定される。

手続き的知識は特別な扱いを必要とするので除外して考えると，論理型知識表現は主に2つの観点から検討されている。

(1) データベース記述の拡張によるアプローチ
ルール型知識をビューの拡張と考える。例えばファミリー・データベース上の先祖などの定義。
(2) 論理型プログラミングによるアプローチ
DCKR[小山85]などのアプローチ。

(1)ではファクト型知識の量が多く，ルール型知識が少ないのは当然であるが現実の世界の知識を表現するのに十分であるかどうかが問題となる。これに対し(2)は記述力が大きいと考えられているが，現在提案されている方式では多くの知識がルール型で表現されるのでPHIのようなシステムにそのまま格納した場合，データベース演算による効率向上はそれほど期待できない。従って従来(2)でしか検討されていなかった問題を(1)と同様に扱い両アプローチを統一的にとらえることにより，データベース演算で効率的に扱える知識の範囲を拡げることが重要な問題となる。

3　性質継承の表現

知識表現の分野では良く知られているが，従来はデータベースによるアプローチでは扱われていなかった代表的な問題に性質継承の表現がある。この問題は意味ネットワークなどで表現されることが多いが述語論理で表現することもできる。論理プログラミングの例として良く知られているのは次のような表現である[KOWA79]。

```
fallible(X):-mammal(X).
mammal(X):-human(X).
human(socratesu).
human(turing).
greek(socratesu).
```

この例ではhumanのfallibleという性質をソクラテスとチューリングが継承する。また，DCKRでは以下の例がある。

```
sem(X,is_a:X).
sem(clyde#1,Property):-sem(elephant,Property).
sem(elephant,color:gray).
sem(elephant,Property):-sem(mammal,Property).
sem(mammal,bood_temp:warm).
```

この例ではelephantの性質をclyde#1が継承し，mammalの性質をelephantが（従ってclyde#1が）継承する。

上記2つの表現方法の考え方はかなり異なっているが，両者とも個々の物または集合の関係や属性を一部はファクトにより，一部はルールにより表している。すなわち第1の例ではhumanの属性を，第2の例では継承関係（is_aリンク）をルールで表している。従ってこれらの表現法をPHIで採用した場合，知識ベース中のルールとファクトの量には大きな差がなく，データベース演算による処理効率の向上はあまり期待できない。

性質継承を意味ネットワーク的に表現した場合は継承の処理はis_aリンクをたどることによりなされる。DCKRではis_aリンクをルールにより表現しており，演繹推論処理がリンクをたどる処理に対応している。これらの表現方式でルール数が多くなる主な理由はリンクをたどることに関する知識が陽に表現されず，論理型言語に内在する演繹推論（三段論法）機能をそのまま用いるためにリンクをルールで表現していることにある。

一般にリンクをたどることを三段論法に対応すると考えれば，三段論法そのものは個々のis_aリンクとは独立であり，ルールとして表現できる。すなわち，

KBMS PHI:A Representation of Inheritance Hierarchies with Exceptions
Nobuyoshi MIYAZAKI*, Chiaki SAKAMA**, Hidenori ITOH**
*Oki Electric Industry Co., Ltd.
**Institute for New Generation Computer Technology

```
A→B,
   B→C
-------
A →  C
```

は

```
is_a(A,C):-is_a(A,B),is_a(B,C).
```

と表現すれば個々のis_aリンクをファクトにより表現し，そのリンクをたどることができる。従って個々の性質をファクトで表現しても性質継承をis_aを用いて実現できる。この方法を用いれば，性質継承の表現は少数の一般的知識をルールで表現し，個々の知識はファクトで表現できるので，データベース演算を有効に利用できる。このような方法のもう１つの利点は推論規則が陽に表現されるため，推論規則の変更を行ったり，各種のリンクを別々に扱うことが可能になる点にある。例えば，要素関係や部分集合関係をすべてis_aリンクで表現することもできるし，element_ofやhas_a等を用いて扱うこともできる。このような表現方式を，KRBF (Knowledge Representation By Fact) と呼ぼう。

KRBFで第１の例を表現すると，

```
is_a(socrates,human).
is_a(turing,human).
is_a(socrates,greek).
is_a(human,mammal).
property(mammal,fallible).
is_a(X,Y):-is_a(X,Z),is_a(Z,Y).
property(Object,Property):-
    is_a(Object,Class),property(Class,Property).
```

となる。第２の例についてもほぼ同様にして表現できる。知識表現で扱われる多くの問題が性質継承問題の変形であるため，広範囲の知識の処理をデータベース演算で効率よく処理できる。

4．例外の表現

知識表現においては性質継承が有用であると同時に，継承にあたっては例外を考慮する必要が生じる場合が多い。例外の問題は非単調論理やデフォールト論理によって扱うことができる[ETHE83]。例えば，"鳥は飛べる"，"ペンギンは鳥である"，"ペンギンは飛べない"の継承は，

```
bird(X)∧M[can_fly(X)]→can_fly(X).
```

のように表現される。Mオペレータは否定が証明できないことを意味する。このような例外はDCKRなどでも容易に扱え，この例では，

```
sem(penguin,Property):-
    sem(bird,Property),not(Property=can_fly:X).
```

のように表現される。この表現方法では，例外の数だけルールのボディにnotが必要となる。また例外の例外（例外のキャンセル）は例外を否定するのではなく，その対象に対して再び同じ性質を記述することによってなすことができる。

KRBFでは個々の例外に関する知識はファクトで表現し，例外の扱いをルールで記述する。この時，is_aの推論規則に厳密なis_aとデフォールトのis_aを区別する立場もあるが，以下ではis_aは厳密なis_aとして性質継承のルールに例外処理を記述する。ペンギンの例では，

```
prop(bird,can_fly).
is_a(penguin,bird).
excep(penguin,can_fly).
property(Obj,Prop):-
    prop(Obj,Prop).
property(Obj,Prop):-
    is_a(Obj,Class),
    prop(Class,Prop).
    not(exception(Obj,Class,Prop)).
```

となる。exceptionの定義は，

```
exception(Obj,Class,Prop):-
    excep(Obj,Prop).
exception(Obj,Class,Prop):-
    is_a(Obj,Subclass),
    is_a(Subclass,Class),
    excep(Subclass,Prop).
```

である。ここでexceptionの定義が複雑になったのは例外の例外を考慮したためである。例外の例外は，DCKRと同様再び同じ性質を記述することによって行う。例えば"スーパーペンギンは飛べる"とすれば，

```
is_a(emperor_penguin,penguin).
is_a(super_penguin,penguin).
prop(super_penguin,can_fly).
```

となり，皇帝ペンギンは飛べないがスーパーペンギンは飛べることを推論できる。例外の例外がない場合にはexceptionのルールは簡単化できる。

これまでの議論では"not"はPrologの"not"と同じくCWAのもとでのnegation as failureと同じものとして扱ってきた。データベース演算ではこの"not"は差演算に変換して実現できる。差演算と再帰定義を含む系では解の存在性の問題が発生するが，一定の制限のもとでこのような系での解を求めることができる[MIYA86]。従って従来は意味ネットワークや論理型言語の処理系（Prologなど）で扱われていた例外のある性質継承の扱いが，データベース演算を用いても実現可能である。

5．おわりに

知識ベース管理をデータベース管理をもとに行うシステムでの知識表現について検討し，例外のある性質継承の表現もファクト型知識と少数のルール型知識によって行うKRBFを提案した。KRBFでは知識処理の効率化はデータベースにおける処理最適化問題として扱うことができる。

《参考文献》

[ETHE83] Etherington, D. W., Reiter, R., "On Inheritance Hierarchies with Exceptons," Proc. AAAI, 1983.

[KOWA79] Kowalski, R., "Logic for Problem Solving," North-Holland, 1979.

[MIYA86] Miyazaki, N., et al., "Compiling Horn Clause Queries in Deductive Databases:A Horn Clause Transformation Approach," ICOT Technical Rep. TR-183, 1986.

[小山85] 小山，田中，"Definite Clause Knowledge Representation," Proc. of Logic Programming Conf., 1985.

[宮崎86] 宮崎 他 "KBMS PHI(2)知識とデータの扱いに関する一考察," 情報処理学会第32回全国大会, 2M-6, 1986.