(20)

AI 86−33

大規模知識処理支援環境ＥＸ－ＤＤＢ

吉田　敦・田中英彦
（東大）
@嶺野和夫
（富士通）

１９８６年１２月１２日

社団法人　電子通信学会

大規模知識処理支援環境ＥＸ＿ＤＤＢ

Large Scale Knowledge Processing Support Environment ＥＸ＿ＤＤＢ

吉田敦† Atsushi YOSHIDA　嶺野和夫†† Kazuo MINENO　田中英彦† Hidehiko TANAKA

東京大学　工学部† University of Tokyo Fuculty of Engineering

富士通株式会社†† FUJITSU LIMITED

1．はじめに

エキスパートシステムを始めとする、大規模な知識処理システムを構築する方法の１つとして、推論機能を持った関係データベースシステム即ち演繹データベースシステムの構築が考えられる。演繹データベースの実現方法には、関係データベースと推論機構の単なる結合による方式と、関係データベースの拡張により推論機能と関係データベース機能の融合による方法の２通りがある。我々は推論処理方式選択の柔軟性と、処理効率という点から後者の方法を採用し、この方法に基づく演繹データベースとして、ＥＸ＿ＤＤＢと呼ばれるシテスムを開発中である。我々のとるアプローチの特徴は、演繹データベースの構築に当たって、応用、言語、実行方式、アーキテクチャの４つの側面から総合的に構築方法を考えるというものである。このようなアプローチは従来の、実行方式のみに着目した演繹データベースの研究に対するアプローチとは一線を画するものである。

推論の実行方式については別稿[1][2]で既に発表済みなので本稿では、言語の側面を中心に述べることにする。第２章では我々の開発している演繹データベースシステムＥＸ＿ＤＤＢの概要を述べる。第３章ではＥＸ＿ＤＤＢ上で動作する論理型言語ＲＯＴＨＯＲＮについて述べ、第４章で、これをエキスパートシステムを始めとする応用に利用する方法について考察する。

2．演繹データベースシステムＥＸ＿ＤＤＢ

本章では、演繹データベースシステムＥＸ＿ＤＤＢの概要を述べる。

2.1 ＥＸ＿ＤＤＢの開発方針

演繹データベースシステムＥＸ＿ＤＤＢは、演繹データベースが関係データベースの延長上にあるという理念に基づく研究の成果として開発されるものである。従ってその研究・開発は、以下のような方針に基づいて行なわれている。

(1) 関係データベースの機能拡張として演繹データベースをとらえ、各部分に関係データベースで培われた技術を適用する。（融合方式）

(2) 応用分野としてエキスパートシステムを仮定し、その開発・運用に必要な機能を実現する。

(3) 実行言語として、集合や関係を関係操作で実現する機能を持った論理型言語を設定しさらにプログラミング支援機能との一体化を図る。

(4) 推論の実行方式には、束縛情報を用いたreduction、束縛情報を用いないreduction 、関係演算系列に展開しておいたものを実行する方法などを用意し、前処理で決定した戦略に基づいてこれらの方式を柔軟に使いわける方法をとる。

(5) 基本となる関係代数演算は、単一化処理を含むよう拡張された関係代数演算ＥＸＴＲＡを用い、関係自体、複合項や変数を含むように拡張されたものを用いる。

(6) アーキテクチャはＥＸＴＲＡの効率良い実行を支援するものとし、関係データベースマシン実装に用いられる技術を採用したものとする。

現在までに、推論実行方式に関する検討はほぼ終了し、簡単なシミュレーションによる処理効率の評価もほぼ終了段階に入っている。プログラミング支援機能を実現するエディタやデバッガについては基本的な部分は完成し、メタ知識の拡充を行なっているところである。基本処理単位となる拡張関係演算ＥＸＴＲＡについても基本演算系はほぼ確定している。次節以降に現在までに確定した部分について述べる。

2.2 ＥＸ＿ＤＤＢの推論実行方式

2.2.1 質問の前処理

ＥＸ＿ＤＤＢではシステム立上げの際に、節定義を参照して、rule/goal graph を生成・解析し、リテラルに対する処

理戦略の決定を行なう[3]。処理戦略としては、ＥＸＴＲＡ系列の実行、束縛情報による定義絞り込みを伴うreduction、束縛情報を用いないreduction があり、再帰定義を含まないものに対してはＥＸＴＲＡ系列の実行が選択される。前処理の段階では、selection の条件部分は決定出来ないので、selection の条件部分を未定義のままにした演算系列を生成し、実行時に未定義部分を具体化する。このＥＸＴＲＡ系列を我々は演算系列のテンプレートと呼ぶ。前処理ではこれを生成し、heuristicsによる最適化を実行する。これらの処理は実際に質問を受ける前に行ない、これによって質問実行時の処理時間を減らしている。再帰定義を含むものについてはreduction の実行を選択し、関連する節の定義の数や、graph の形状などから束縛情報を使うかどうかを決定する。

前処理で得られたＥＸＴＲＡ系列または推論実行戦略などの情報は、リテラルに対する定義情報として推論実行時に参照される。

2.2.2 推論の実行

質問に対する推論処理は次のように実行される。ここで、ゴールは、処理待ちのためのスタックに格納されるものとし、質問を受けた際にそれに対する全解を格納するための全解関係を生成するものとする。

(1) ゴールを１つ選択し、予め決定した計算規則に基づいてその中のリテラルを１つ選択する。

(2) リテラルに対する定義情報を参照する。

(3-a) 全解探索のためのＥＸＴＲＡ系列がある場合は、リテラルの引数に関する情報を用いてその系列を具体化し、そちらを実行する。実行の結果得られた束縛情報と、ゴールに対する束縛情報との間でconsistency check のためのjoinを実行する。ゴールに残りのリテラルがなければ得られた束縛情報を全解関係に追加する。ゴールに残りのリテラルがある場合にはそのリテラルと束縛情報を新しく導出されたゴールとしてスタックに格納する。

(3-b) そうでない場合は束縛情報を用いたreduction または束縛情報を用いないreduction を実行する。束縛情報を用いたreduction は次の様に実行される。

◎ factリテラルの場合は、ゴールインスタンスの生成、ゴールインスタンスの集合と定義の集合の間での単一化結合による定義の探索と、そのfactに対する束縛情報の抽出、得られた束縛情報とゴールの束縛情報との単一化結合によるconsistency check を実行する。ここでゴールに残りリテラルがなければ得られた関係を全解集合に追加する。残りリテラルが存在する場合にはそれらと得られた関係を導出ゴールとしてゴールスタックに格納する。

◎ ruleリテラルの場合にはゴールインスタンスの生成、ゴールインスタンスを用いた定義の単一化パタン検索とその結果からの導出ゴールの生成を実行する。導出ゴールは、もとのゴールの束縛情報を与えられ、ゴールスタックに格納される。

◎ rule、fact両方の定義を持つ場合は、ruleに対する処理実行の後、factに対する処理を実行する。組み込み述語の場合は、その時点で評価を実行し、引数の束縛が不完全な場合には処理を後廻しにする。

(3-c) factの定義数が比較的少なく、探索木に拡りがない時には、束縛情報を用いないreduction が実行される。この場合には、ゴールが関係で表現され、その中に変数に対する束縛が記述される。

2.3 ＥＸ＿ＤＤＢのプログラミング支援環境

ＥＸ＿ＤＤＢでは、応用プログラムの開発・運用を支援するためのエディタとデバッガを提供する。エキスパートシステムのような応用の場合、システムの運用と開発が混在した形で運用が行なわれるので、このような機能に対する必要性は極めて高くこれらと実際の処理系の間の制御の受け渡しが柔軟かつ効率良く行なわれる必要がある。

エディタの機能としては、正規形で表現されるfactに対するスキーマ、キー、制約などの記述情報の定義・参照の機能やこれらの記述情報を用いたrule設計支援の機能の他に、既存の関係データベースに格納される関係をfactの集合として取り込む機能を持つ。また、factに対するスキーマなどを先に定義しておいてからあとでfactのインスタンスを追加することができ、設計型のエキスパートシステムの構築も可能になっている。

デバッガの機能についてはステップ動作やブレークポイントの設定などのほかに、rule実行の経歴をlog の形で残し、これを用いたデバッグが可能になっている。このlog は実行中に自動的に登録される。log の他の用途としては、エラー回復や説明機能の実現がある。

2.4 ＥＸ＿ＤＤＢのアーキテクチャ

以上で述べなかった部分即ち言語とアーキテクチャのうち、言語については本稿の中心的課題であるので次章にて述べる。アーキテクチャについては次の様に考えている。

ＥＸ＿ＤＤＢでは推論処理の基本単位となる演算は、単一化を伴うように拡張された関係代数演算ＥＸＴＲＡである。この演算は、高速化の為、hashとsortに基づくアルゴリズムを採用している[4]。hashは、タプルの属性値である項を木構造に展開した場合の各ノードに対して、それが定数ならそのシンボルに対する非零のhash値を求め、変数ならhash値として零を与え、これらの値のベクトル（hashベクトル）を求めることで実行する。タプルはこのhashベクトルの値により可変長文字列チューニング機構をもつソータでsortされ、１台の処理モジュールの処理単位であるバケットに分割される。単一化結合は対応するバケット同士の間でのみ実行すればよく、処理は２つの関係のタプル数の和のオーダになる。このような演算を支援するアーキテクチャを、論理的動作のレベルから検討を行なっている段階である。

3．論理型言語ＲＯＴＨＯＲＮ

我々はＥＸ＿ＤＤＢの上で動作する論理型言語としてＲＯＴＨＯＲＮ(Relation OrienTed HORN clauses)と命名された言語を考案し、現在その第１版を実装中である。この言語は第２章で述べた方法により推論を実行する論理型言語で、集合・関係を集合単位で扱う機能を持っている。本章ではこの言語について説明する。

3.1 ＲＯＴＨＯＲＮ設計の動機

ＲＯＴＨＯＲＮはDec-10Prologに準拠した文法を持つ論理型言語である。この言語は、実際に応用を記述する時に必要になる幾つかの組み込み述語を持ち、さらに新しいデータ型として集合（関係）が導入されている。これらの特徴を持たせた動機は、以下の通りである。

従来の演繹データベースでは上で動作する言語としてpure Prolog を仮定してきたが、実際の応用開発を考えた場合pure Prolog では明らかに記述力不足である。pure Prolog は理論的な扱いは容易であるが、最初からpure Prolog を仮定して演繹データベースシステムを設計した場合、実用化の際に組み込み述語を考えることで、根本的な設計変更が必要になる危険性が高い。従って、応用プログラムの記述に十分なだけの組み込み述語を持った言語を最初から設定してやる必要がある。

演繹データベースは、関係データベースを基本とし、その延長上にあるものと考えられるので、その上で動作する言語に、関係即ち集合を効率良く扱う機能を持たせることが可能であり、自然でもある。従ってＲＯＴＨＯＲＮに対して、新しいデータ型として集合を導入し、これを扱う機能を、関係データベースの関係操作機能を用いて実現する。

次節で、ＲＯＴＨＯＲＮの組み込み述語とその意義について述べる。これらの組み込み述語が入っていることと、組み込み述語のシンタックスや意味が若干異なる点を除けば通常のPrologと同様の記述力を持ち、通常のPrologと同じ様に使用できる。但し、基本的には与えられた質問に対して全解探索を実行し、質問に対する解全体の集合を返すようになっている。通常のPrologでは条件を満す最初の解を探索し、全解探索は、バックトラックを用いる。

3.2 ＲＯＴＨＯＲＮの組み込み述語

3.2.1 四則・比較演算述語

組み込み述語の中で最も必要性が高いのが四則・比較演算である。四則演算としては演算の対称性より、加算と乗算があれば十分であるが、ＲＯＴＨＯＲＮではプログラミングの容易さを重視して減算、除算も実装した。同様に、比較演算としてはX > Y とX < Y 、X = Y があれば十分であるが同じ理由でX <= Y, X >= Y, X != Yも実装した。

これらの四則・比較演算は、現在の版では３引数または２引数述語のシンタックスで実装されている。　これらの述語とその意味を下の表に示す。

意味（演算）	組み込み述語
加算　X + Y = Z	plus(X,Y,Z)
減算　X - Y = Z	minus(X,Y,Z)
乗算　X * Y = Z	times(X,Y,Z)
除算　X / Y = Z	div(X,Y,Z)
比較　X > Y	gt(X,Y)
X < Y	lt(X,Y)
X >= Y	ge(X,Y)
X <= Y	le(X,Y)
X = Y	eq(X,Y)
X != Y	ne(X,Y)

表１．四則・比較演算一覧

これらの四則・比較演算により、束縛情報に対する絞り込みが実行される場合がある。

3.2.2 集合・関係演算

ＲＯＴＨＯＲＮでは、集合または関係を、集合単位で操作する機能を持つ。これらの集合・関係操作は、従来のPrologのように集合または関係をリストで表現して、それに対する操作をtail recursionのプログラムで記述するのではなく、直接に集合・関係操作のルーチンを起動して処理を実行する。従って集合操作はtuple at a time ではなく１まとまりの集合を単位とした操作で実行される。（シミュレータではＣ言語で直接記述、具体的には、拡張関係演算ＥＸＴＲＡを実行するサブルーチンの起動により実行される。）

集合・関係データに対する操作としては次のようなものがある。

◇　relof(Vlist,Goal,Relid)

Vlist に含まれる変数を属性とし、その変数の組のうちGoal で指定されたゴールを満すものの集合を生成し、その名前をRelid に返す。

```
例)  ?- relof([_0,_1],p(_0,_1),Rel)
     p(X,Y):-f1(X,Z),f2(Z,Y).
     f1(butaba,rep1).   f1(hisada,rep15).
     f2(rep1,e).        f2(rep15,a).
```

rel1

_0	_1
butaba	e
hisada	a

Rel = rel1

この述語の実行により生成された関係は内部関係管理テーブルにその名前と実体へのポインタの対応が登録され、主記憶上に保持される。消去は後述するdeleteにより実行される。

◇ union(Rel1,Rel2,Rel3)

Rel1、Rel2で指定される集合の和を生成して、その名前をRel3に返す。

◇ intersection(Rel1,Rel2,Rel3)

Rel1、Rel2で指定される集合の積を生成して、その名前をRel3に返す。

◇ diff(Rel1,Rel2,Rel3)

Rel1、Rel2で指定される集合の差を生成して、その名前をRel3に返す。

◇ member(Ele,Set)

要素Ele が集合（関係）Set の要素ならsuccess し、そうでなければfailする。

◇ select(Rel1,Attr,Value,Rel2)

Rel1のタプルのうち属性Attrの値がValue に等しいものを選択した結果を生成し、その名前をRel3に返す。

◇ join(Rel1,Rel2,A1,A2,Rel3)

Rel1とRel2をRel1の属性A1の値＝Rel2の属性A2の値という等価性で結合した結果を生成しその名前をRel3に返す

◇ project(Rel1,Alist,Rel2)

Rel1を属性集合Alist 上に射影した結果を生成し、その名前をRel2に返す。

◇ aggreg(Rel,At,Op,Value)

関係Rel の属性Atに対してOPで指定された集約演算を実行する。集約演算としては、最大値(max)、最小値(min)、平均(avg)、総和(sum) が指定可能である。

例) ?-select(rel1,_1,100,R)

rel1

_0	_1	_2
butaba	10	dbms
hisada	100	network
butaba	20	micom
hisada	100	ai

下のrel2が生成・登録され、R にはrel2が代入される。

rel2

_0	_1	_2
hisada	100	network
hisada	100	ai

3.2.3 関係型データ管理述語

3.2.2 で生成される関係データに対する管理操作を実行する述語である。これらの述語により、演繹データベース上で簡単なデータベース管理システムを記述することができる。述語としては、従来の関係問い合わせ言語を参考に、以下のようなものを用意した。

◇ delete(rel)

Rel で指定された関係を内部関係データ管理テーブルより抹消する。

◇ set_schema(Rel,Schema)

◇ get_schema(Rel,Schema)

Rel で指定された内部関係に対して、スキーマを設定または参照する。スキーマは意味記述のリストの形で指定・参照され、内部では次のような形で表現される。

&schema(Relname,[Sem1,Sem2,...,Sem$_N$])

ここでSemiはRel の第 i 属性の意味を表わす文字列定数であり、ユーザが自由に指定出来る。

◇ set_key(Rel,Keylist)

◇ get_key(Rel,Keylist)

Rel で指定された内部関係に対してキー属性を指定する。キー属性はキーとなる属性のリストで表現される。

キー属性情報は内部では次の様に表現される。

&key(Rel,[Key1,...,Key$_N$])

例) rel1

_0	_33	_2
butaba	graphics	3
jamajita	database	33
hisada	graphics	100
takashika	database	100

?- set_schema(rel1,[name,course,score]).

?- set_key(rel1,[name]).

により内部関係rel1に対してユーザが定義したスキーマ定義とキーの内部表現が次の様な形で生成される。

&schema(rel1,[name,course,score]).

&key(rel1,[name]).

◇ save(Rel,File)

◇ load(Rel,File)

Rel で指定された内部関係のsave、loadを実行する。saveまたはloadされるのはRel で指定された関係データと、それに付随するユーザ定義のスキーマ、キー情報である。これらのデータはfact定義とは異なり、単なる外部記憶として扱われる。

3.2.4 標準入出力述語

標準入出力端末を通してユーザ／プログラマとデータの受け渡しを行なう述語である。標準的なものとして次の６つを用意した。

◇ print(X)

X に束縛された値を標準入出力端末に表示する。この述語は必ずsuccess する。

◇ read(X)

標準入力から読み込んだ文字列を引数X に束縛する。

◇ put(X)

文字X を標準出力に出力（表示）する。

◇ get(X)

標準入力から読み込んだ文字をX に束縛する。

◇ putc(N)

指定された整数に対応するコードを出力する。

◇ getc(N)

標準入力から読み込んだコードを変数N に束縛する。これらの他に型変換として、以下のようなものを用意してある。これらを用いて、実数または整数値の入出力が実現できる。

◇ atoi(X,Y)

X に束縛された文字列を整数に変換し得られた整数をY の値として束縛する

◇ itoa(X,Y)

X に束縛された整数を文字列に変換し得られた文字列をY の値として束縛する。

◇ atof(X,Y)

X に束縛された文字列を実数に変換し、得られた実数をY の値として束縛する。

◇ ftoa(X,Y)

X に束縛された実数を文字列に変換し、得られた文字列をY の値として束縛する。

入出力ファイルの切り替えのための述語としては、C-Prologとの互換性を考え、以下のようなものを用意した。

◇ tell(File)

出力ファイルを指定されたファイルに切り替える。以後の出力先はtoldが評価されるまで指定されたファイルになる。

◇ telling(File)

現在の出力ファイルの名前を返す。

◇ told(File)

現在の出力ファイルを閉じ、出力を標準出力に戻す。

◇ see(File)

入力ファイルを指定されたファイルに切り替える。以後の入力元はseenが評価されるまで指定されたファイルになる。

◇ seeing(File)

現在の入力ファイルの名前を返す。

◇ seen(File)

現在の入力元ファイルを閉じ、入力元を標準入力に戻す。

3.2.5 ファイル更新を伴う述語

◇ fassert(X)

◇ fretract(X)

X は節の定義を表わす文字列である。この述語の実行により演繹データベースファイルの定義節に対する追加または削除が実行される。定義節の変更は通常エディタを呼び出して実行する上、これらの更新実行後に質問前処理結果の更新が実行されるので、この述語は頻繁には使用しない。

3.2.6 プログラミング支援環境の呼び出し

ＲＯＴＨＯＲＮは推論処理の実行系だけではなく、その上で応用を記述するための環境も提供する。以下にそれらの環境を使用するための述語について述べる。

◇ calledit

エディタを呼び出す述語。エディタ起動後は定義の集合が変更されている場合があるので、変更に応じて質問前処理結果を更新する操作を実行する。

◇ trace(Flag)

動作モードの設定を行なう。引数Flagをonにするとステップ動作を実行するモードになる。off にすると通常の動作モードになる。

◇ debug(Pred)

デバッガを呼び出し、制御をデバッガに移す。デバッガの機能として、スパイポイントの設定、ステップ動作実行log の表示がある。

◇ system(X,Vallist)

外部関数またはＯＳのコマンドの実行を行なう。外部関数の返す値は、第２引数のリストの中に引数の順番に値が返される。

3.3 ＲＯＴＨＯＲＮの知識体系

ＲＯＴＨＯＲＮは述語論理を基本とする言語であり、取り扱う知識はrule型知識とfact型知識の２つに大別される。fact型知識はさらに正規形の関係で表現されるものと、複合項を含むため正規形にはならないが関係表の形で扱えるものに分けられる。このうち正規形で表現されるfactについてはruleの記述を容易にするために、その定義の集合を関係とみなして関係スキーマやキー、制約を定義する。

例)

h4_sci3

head	body
sci3(8400026,takashika,database,100)	
sci3(8300171,butaba,graphicsII,10)	
sci3(8300084,hisada,graphicsII,100)	
sci3(8400166,jamashita,database,30)	

このfactの定義集合に対して以下のようなスキーマ、キー、

制約を与える。

integ(R,A,X)はX が関係R の属性A に対する値であるための条件を示すruleである。

スキーマ（fact定義に対するスキーマ）

```
$ sci3(student_id,student_name,cource,score)
```

キー（fact定義に対するキー）

```
key(h4_sci3,[student_id])
```

制約（fact定義に対する制約）

```
integ(h4_sci3,sutdent_id,X):-type(X,integer).
integ(h4_sci3,student_name,X):-type(X,string).
integ(h4_sci3,course,X):-type(X,string).
integ(h4_sci3,score,X):-type(X,int),ge(X,0),
                        le(X,100).
```

ruleは、関係の形で表現されたfactから新しい事実を取り出し格納するための仮想関係に対するビューとして定義されるものと、手続き的操作を示す組み込み述語に分けられる。仮想関係に対するビューとしては再帰的定義を含むものも可能であり、この点において従来のデータベース問い合わせ言語より記述力の上で優っている

手続き的操作を示す組み込み述語としては前節で述べたものがある。これらを最初から組み込んだことにより従来の演繹データベースで設定されたpure-Prolog とは異なり実用に耐えうる言語となっている。

集合・関係操作により生成される内部関係データはユーザからは名前のみで管理するようになっている。名前と実体の対応は内部関係管理テーブルで管理される。これらの内部関係に対してユーザがスキーマやキーを自由に定義できるようになっているので、ユーザが独自の関係データベースシステムを実装することも可能である。これらの内部関係データはfactの定義を表わす関係とは別のものである。

4．考察

本章では、応用面からＲＯＴＨＯＲＮの機能に関する考察を行なう。

4.1 問い合わせ言語としての機能

データベースのユーザからは、ＲＯＴＨＯＲＮは論理型言語の記述力により取り扱えるビューのクラスが拡張され、扱える関係データも拡張されたデータベース操作言語とみなせる。

通常のデータベース問い合わせ言語では、言語自体の記述力の制約から、関係に対するビューは再帰を含まないものに限定されていた。再帰を含むビューを扱うためにはＣ言語など他の言語に埋め込む必要があった。これに対してＲＯＴＨＯＲＮは論理型言語の本来持つ記述力により、最初から再帰を含むビューの扱いを可能にしている。

また通常の関係データベースでは複合項や変数の格納が出来ないが、ＲＯＴＨＯＲＮでは関係の概念を拡張し複合項を属性の値として許している。このためＲＯＴＨＯＲＮにおけるfactには正規形の関係で表現されるものと、正規形の関係にはならないが関係表の形で扱えるものの２種類がある。正規形で扱えるfactについては、スキーマ、キー、制約などの記述を許し、これらの記述の参照や設定・修正はエディタ側で行なう。正規形で表現されないfactについても、関係の概念拡張と、関係操作に単一化を伴う拡張を行なうことにより、関係の形で表現することを許している。従ってfactに対する操作は、集合を単位とした操作になる。

4.2 論理型言語としての機能

ＲＯＴＨＯＲＮを論理型言語の拡張とみなした場合、拡張された部分は、集合・関係データの取り扱いと、関係データベース操作機能である。これらの拡張は、関係データベースの環境を活かしたものである。

集合・関係の操作は、拡張関係演算命令の直接実行により行なわれる。これは通常のProlog処理系において集合を要素のリストで表現し、集合・関係演算をリストに対するtail recursionのループで実現するのとは異なる方法である。つまり、通常のPrologによる集合・関係演算が本質的にtuple at a time であるのに対してＲＯＴＨＯＲＮではrelation(set) at a time になっている。この方が集合型のデータに対する操作としては自然であり、関係データの持つ並列性を生かしている。並列動作と処理の効率については関係データベースマシンの技術を用いることによりアーキテクチャのレベルで支援を行なう。

ＲＯＴＨＯＲＮにおいては集合・関係は、生成された時に内部関係管理テーブルに登録され、そのテーブルで実体と名前が対応付けられる。集合・関係演算（操作）述語では名前を指定して関係を取り出し、操作を実行する。これらの操作では参照される関係自体には更新操作をせず、操作の結果として新しい関係（集合）を生成する。生成された集合（関係）には唯一の識別名が与えられる。変数の値としては集合または関係の実体ではなくその名前が代入されるので、集合や関係をデータとして取り入れたことにより正規性を損なうことはない。

4.3 知識処理システム記述言語としての機能

4.1 及び4.2 の考察からＲＯＴＨＯＲＮはエキスパートシステムなどの応用を開発する上で次のようなメリットを持つと考えられる。

(1) 関係データベース操作機能により、関係データベースを用いたエキスパートシステムが記述できる。
(2) 集合・関係データを直接サポートしているので集合を操作する処理の多い応用に向いている。
(3) エディタ、デバッガの呼び出しが言語から可能なので運用と開発のフェイズが混在可能であり、エキスパートシステムの開発・運用に適している。

(1) に関しては演繹データベースが拡張された関係データベースとして単独でエキスパートシステムとしての動作をする場合と、演繹データベースが外部の関係データベースにアクセスまたは処理の依頼をして推論を進めていく分散演繹データベースの２通りの実現方法がある。前者の場合は実現に対しての問題点は少ないが、後者の場合は、分散システム独特の問題、つまり、通信方式、同期や並行動作制御の問題がある。このような分散型の演繹データベースは、例えば外部のデータベース（画像データベース等が考えられる。）で、演繹データベースではサポート出来ない処理を実行してもらうような形態のものになる。実際の応用を考えていく場合このような形態の演繹データベースの研究は必要であり、このような演繹データベースシステムの実現方法は今後の重要な研究課題の一つである。

(2) に関しては、条件を満す事実（タプル）の集合を生成して、これに対して集約演算や関係・集合演算を実行する形で問題解決を行なうようなエキスパートシステムが、演繹データベース上で実現可能であるといえる。このような処理方法を採るエキスパートシステムとしては例えば、法律関係や医療診断などの分野のものが考えられる。このようなエキスパートシステムを演繹データベースの上で実現する場合、問題解決に必要な事実は、関係または値として複合項を持つように拡張された関係で表現され、ruleは関係に対する再帰を含むビューや関係に対する制約の形で表現される。問題解決は、質問に対する全解探索を実行しその結果を直接利用するかまたは集約演算の形で最適なタプルを１つだけ取り出すことで実行される。これは大量のデータを集合として扱うエキスパートシステムシステムを記述するパラダイムを与えるものと考えてもよい。

(3) について、ＲＯＴＨＯＲＮは一種のshell を提供するものと考えられる。現在の版では実行系、エディタ、デバッガの間の制御の受け渡しは、図１のようになっている。

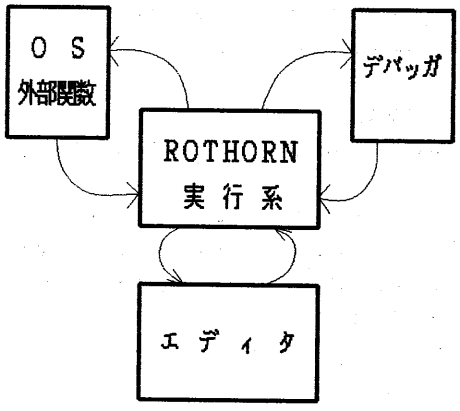

図1. EX_DDB処理系制御遷移図

現在の制御機構は、このように柔軟性の低いものであるが将来的には言語処理系から直接エディタのコマンドの呼び出しを実行できるようにする予定である。

4.4 他の言語との比較

ここまでに何度も述べてきた通り、ＲＯＴＨＯＲＮは従来の演繹データベースにおけるpure-Prolog とは実用性と言う点で一線を画するものである。これは最初からエキスパートシステム等の応用を仮定して設計したからである。

purePrologにデータベース機能を持たせようとしたものは演繹データベース上で動作する言語の候補と考えられる。このような言語の例としては大阪大学のＤＢ－Ｐｒｏｌｏｇや ＭＣＣのＬＤＬがある。

ＲＯＴＨＯＲＮをＤＢ－Ｐｒｏｌｏｇ[5] と比較した場合、記述力はほぼ同じであるが、ＤＢ－Ｐｒｏｌｏｇでは基本となる演算が通常の関係演算であり、定義に対する単一化パタン検索一回につき選択と結合操作を何回も実行しなければならないのに対し、ＲＯＴＨＯＲＮにおける基本演算は拡張関係演算であり、一回の単一化パタン検索に必要な演算の数はせいぜいjoinが２回と関係の間の変換操作が２回で済む。

ＬＤＬ[6] と比較した場合、記述力の上ではほぼ同等であり、基本演算を考えた場合の処理性能も同じ程度であると予測される。ＬＤＬとの顕著な違いは、集合の導入に対する考え方の相違である。ＬＤＬでは変数に集合の実体を直接代入するため、関係の正規性は全く保証されない。これに対してＲＯＴＨＯＲＮは関係の正規性を極力重視して、変数には関係の名前だけを代入し、実体へのアクセスは内部関係管理テーブルを通して行なうようにしている。

4.5 今後の課題

以上の考察から演繹データベース上の論理型言語ＲＯＴＨＯＲＮは、エキスパートシステムなどの開発・運用にほぼ適した言語であると言える。但し、実際の応用プログラムを記述した場合、現在の仕様に追加をする必要が生じる可能性がある。

現段階で必要と思われる拡張としては、推論実行系自体を現在のままで機能を拡張する方向と、推論実行系自体の変更を伴う拡張の２つが考えられている。

推論実行系を変更しない拡張としては、言語側から、エディタで使用するコマンドを直接呼び出す機能、外部の関係データベース側に処理を実行してもらう分散データベース機能などが考えられる。前者を実装した場合より柔軟な制御構造を持つプログラミング支援環境や知識獲得機能を持ったエキスパートシステムが実現できる。後者の機能により、分散型の知識処理システムが実現できる。

言語の実行系自体を変更するような拡張としては、ＭＹＣＩＮのようにＣＦ値による制御を伴う推論を実行可能にする方向と、マルチプロセッサシステムの上での実装を考え、並列処理実行向けにする方向が考えられている。

ＣＦ値による推論は、定義の属性にＣＦ値を追加し、ＣＦ値による選択の制限や結合操作時のＣＦ値の再計算などを実行しつつ行なうことで実現できる。並列版については、ＯＲ並列実行の場合の処理制御機構として、ＧＨＣのガード機構のようなものの導入や、ＡＮＤ並列実行を考えた場合に集合型データを実装するデータ構造としてストリームの導入が必要になる。またゴールの表現形式も、現在のようにゴールのひな型と束縛情報による表現ではゴールに対する資源割り付けの際に特別な制御機構が必要になる。これを回避する方法として、ゴールも関係の形で表現する方法が考えられる。

現在ＲＯＴＨＯＲＮは本稿で述べた仕様を満すよう実装が進んでおり、この上で動作するエキスパートシステムとしてパーソナルコンピュータの構成決定支援システムの試作を検討中である。本節で述べた拡張については今後必要性を検討していく予定である。

5．結論

以上の考察により、演繹データベース上の論理型言語ＲＯＴＨＯＲＮは、大量のデータを集合操作的に扱う応用を記述するのに向いていることが予測される。但し、より柔軟で効率の良い処理を可能にするために、現在の機能に対しさらに拡張を行なう必要性がある。現在ＲＯＴＨＯＲＮを用いた応用として、エキスパートシステムを開発中であり、この開発を通して論理型言語ＲＯＴＨＯＲＮ及び演繹データベースシステムに必要な機能についてさらに検討を進めていく予定である。

＜＜参考文献＞＞

〔1〕吉田、嶺野、田中他「演繹データベースにおける推論実行方式の評価」電子通信学会人工知能と知識処理研究会AI86-24 1986年10月

〔2〕吉田、土井、嶺野、田中他「推論機能と関係データベースの融合－実行方式の評価」第34回情報処理学会全国大会5L-8 1986年10月

〔3〕土井、吉田、嶺野、田中他「推論機能と関係データベースの融合－その質問処理方式」第34回情報処理学会全国大会5L-7 1986年10月

〔4〕大森、吉田、田中「推論機能と関係データベースの融合方式」電子通信学会人工知能と知識処理研究会AI86-21 1986年７月

〔5〕今中、上原、林、河合、豊田「Prologと関係データベースとの結合システムDB-Prolog 」情報処理学会知識工学と人工知能研究会45-8 1986年 3月

〔6〕S.Tsur,C.Zaniolo:"LDL:A Logic-Based Data-Language",Proc. of 12th International Conference on Very Large Data Bases,2A-1 Aug.1986

付録　ＲＯＴＨＯＲＮの文法

```
<rule>::= <head>:-<body>.
<fact>::= <head>.
<body>::= <literal>|(<literal>,)* <literal>
<head>::= <literal>
<literal>::= <fname>|<fname>(<arg>)|
             <fname>((<arg>,)* <arg>)
<arg>::= <term>
<term>::= <list>|<var>|<str>|<num>|<literal>
<list>::= []|[<term>]|[(<term>,)* <term>]|
          [<term>|<var>]|[(<term>,)*<term>|<var>]
<var>::= <Upper>|<Upper>(<alphanum>)*
<str>::= <lower>|<lower>(<alphanum>)*
<num>::= <nonzero>|<nonzero>(<digit>)*
<nonzero>::=1|2|3|4|5|6|7|8|9
<digit>::= 0|<nonzero>
<Upper>::= A|B|...|Z
<lower>::= a|b|...|z
<alphanum>::=<Upper>|<lower>|<digit>
```

(element)*はそのelement が１回以上繰り返されることを示す。listの表記中で〔と〕の間にある｜は差分リスト表現において要素数が確定した部分と要素数未定のリスト部分との境界である。

◎　エディタの機能

現在ＲＯＴＨＯＲＮのプログラミング環境の一環として作成されているエディタは以下のような機能を持つ。

関係で表現されるfactのスキーマ、キーの定義・参照

・定義を表わす関係の表示機能

・定義を表わす関係へのタプル追加・削除などの更新

・外部の関係データベースからのfactの抽出

現在の版ではエディタのコマンド選択は階層構造をなすメニューにより選択するようになっている。

◎　デバッガの機能

・log 内容の参照（起動されたruleや束縛情報の表示機能）

・step動作の実行

・spy point の設定と、次のspy point までの継続実行

step動作モードでは、起動されたruleのidまたは探索の対象となるfactの名前が表示される。また推論動作のために生成された関係の表示も行なう。