演繹機能を有したデータベースの一構成法

奥乃 博
（日本電信電話公社 武蔵野電気通信研究所）

1. はじめに

1970年にE.F. Coddが提案したデータベースの関係モデル[1][2]は、①データベースを表形式で実現すること、②記号ポインタを用いて別の関係中のタップルと結合できること、③関係に対する操作が数学的に取扱うことができること、といった特長を有しており、多くの研究者によって研究が進められている。データベースでは、取扱う対象となる実世界の情報はそれが実世界の中で持っている構造をも含めて表現されていなければならない。関係モデルでは、実世界の情報が1項関係で表現されており、かつ、すべての情報が具体的な実現値によって個々の事実として与えられている。

実世界の情報を表現するには、個々の事実として表現する方法と一般化された事実という意味での知識として表現する方法との二つが考えられる。集合を記述する時に用いる外延的記法と内包的記法にならって、前者で構成されるデータベースを外延データベース（extensional data base）、後者で構成されるデータベースを内包データベース（intensional data base）と呼ぶことにする。この用語を使うと、従来のデータベースは外延データベースと言うことができる。関係データベースは、情報を表の形で蓄積しているので簡単な構造となっているが、関数従属などの従属構造だけでは実世界の意味構造を充分に扱えているとは言い難い。すなわち、関係データベースを外延データベースとして構成するだけでは「実世界の情報をその構造まで含めて表現した」ことにはならない。

最近、外延データベースと内包データベースの両方を用いて構成したデータベースが、データベースの高水準化，高機能化の一環として盛んに研究されている。[3] このデータベースは演繹型データベース（deductive data base）と呼ばれている。演繹型データベースに対する検索は、内包データベースに含まれる知識と外延データベースに含まれる事実とを用いることから一種の演繹処理と考えることができる。内包データベースに含まれる知識としては、外延データベースにない用語の定義がある。Chang[4]は外延データベースに含まれる関係を基本関係、基本関係から構成される関係を仮想関係と呼んでいる。仮想関係は一種の用語の定義である。Reiter[5]は仮想関係を拡張し、一つの関係が内包データベースと外延データベースの両方にまたがって含まれていても良いとしている。内包データベースに含まれる知識はどちらの場合とも演繹処理における推論規則と考えられる。

内包データベースに含まれる知識は、推論規則と考えるほかにデータベースの無矛盾性を保証する完全性制約と考えた方が良いことがある。完全性制約がある時点で正しいからと言って、必ずしもそれが未来永劫に正しいという保証はなく、更新時に完全性制約を破ったという理由で破棄された更新データが実は正しく、逆に完全性制約が不十分であったという場合がある。この場合には完全性制約の方が更新の対象となるべきである（このような更新を非単調変更という）。演繹型データベースは、非単調変更という意味での「不完全なデータベース」を取扱うためには極めて適している。

本稿では、内包データベースに用いる論理の能力を一階述語論理ではなく、ホーン節に限定した演繹型データベースを取り上げ、その構成法について報告する。データベースを構成するにあたっては、述語型言語 PROLOG[6] の拡張版である DURAL[7] を用いた。DURAL の論理能力はホーン節であり、本データベースの論理能力と一致している。第2章では DURAL について述べ、第3章では演繹型データベースの構成法ならびにその処理機構について述べる。

2. 述語型言語 DURAL について

2.1 PROLOG とホーン節

DURAL のベースとなった PROLOG は、PROgram + LOGic という名が示すように一階述語論理を基にしたプログラミング言語である。その論理能力はホーン節に限定されているが、簡単な構造を持ち、柔軟で強力な述語型プログラミングの道具となっている。述語プログラミングでは「やりたいこと」を記述するだけでよく、「どうやるか」を記述する必要はない。この非手続性はデータベースの問合せ言語に対する要求条件とよく一致している。

PROLOG のプログラムはホーン節の集合で与えられる。ホーン節は一般に

$$+A-B_1-B_2\cdots-B_n$$

という形をしている。$+A$ をヘッダ、$-B_1\cdots-B_n$ を本体と呼び、本体中の $B_1,\cdots,B_n$ をゴールと呼ぶ。ホーン節は次の4種類に分類することができる。

i) $+A-B_1\cdots-B_n$ ：「B_1 かつ $\cdots$ B_n ならば A は真である」と解釈される。
ii) $-B_1\cdots-B_n$ ：「B_1 かつ $\cdots$ B_n は偽である」と解釈される。
iii) $+A$ （単一節）：「A は真である」と解釈される。
iv) □ （空節）：「矛盾である」と解釈される。

例1 human, motal を各々「人間だ」,「死ぬ運命にある」という述語とすると

```
(1) +(human Socrates)
(2) +(motal *x)-(human *x)
```

という節の集合が与えられている時、(motal Socrates) を判定するには (1) と (2) から *x に Socrates を代入すると +(motal Socrates) が得られ*、真であることが分かる。この証明法は、証明したい節と同じヘッダを持つ節の本体のすべてのゴールが真の時に、証明したい節が真であるというボトムアップ式である。これは、ホーン節の宣言的な解釈に基づいている。

一方、上記4種類のホーン節は次のように手続き的にも解釈できる。

i) $+A-B_1\cdots-B_n$ ：手続きの定義である。手続きの名前は A、本体は手続き呼び出し $B_1,\cdots,B_n$ である。つまり、「ゴール A を達成するためにはゴール $B_1,\cdots,B_n$ を達成せよ」と解釈。
ii) $-B_1\cdots-B_n$ ：「ゴール $B_1,\cdots,B_n$ を達成せよ」というゴール文、あるいは、問合せと解釈される。
iii) $+A$ ：「ゴール A は達成されている」と解釈される。
iv) □ ：STOP 文と解釈される。

上の例のゴールは

```
(3) -(motal Socrates)
```

で与えられるから、まず (2) とマッチングが取られ（統合と言う）、

*) 推論規則として三段論法を用いた。

```
-(human Socrates)
```

という新たな手続きが呼ばれ，(1)と統合されて(3)が達成されることになり，証明が終了する。この証明法は，手続きを次々と呼び出し，もし途中で証明に失敗した場合には直前の手続きを元に戻すという逆戻りを行って再び繰返すというトップダウン式である。これは，ホーン節の手続き的解釈に基づいている。

PROLOG の証明法は，導出原理の一種である input resolution を用いている。input resolution は，統合すべき節の相手として必ず入力で与えられた節（プログラムとして与えられた節）を選ぶ方法である。PROLOG では，節の順序，あるいは，一つの節の中のゴールの順序は制御文としての意味を持っている。

例2

```
(1) +(on A B)
(2) +(on B C)
(3) +(above *x *y)-(on *x *y)
(4) +(above *x *z)-(on *x *y)-(above *y *z)
```

例2の節の集合に対して -(above A D) は証明できないという答が返されるが，(4)を(4)′で置き換えた節の集合に対しては，無限ループに入ってしまう。

```
(4)' +(above *x *z)-(above *x *y)-(on *y *z)
```

このような実行順序の概念は宣言的解釈とは相入れないので，手続き的解釈が PROLOGでは採られている。

例2において，-(on A B) という問合せは「AがBの上にあるか」という事実のチェックを行うし，-(on A *y) は「Aの下にあるのは何か」，-(on *x B) は「Bの上にあるのは何か」，-(on *x *y) は「上下関係にあるものは何か」という検索を行う。このように PROLOG では一つの述語が何通りの意味でも使うことができ，極めて柔軟なプログラムを書くことができる。

2.2 DURAL — PROLOGの拡張

Kowalski[6]によると，アルゴリズムは論理部と制御部から構成されており，論理部はアルゴリズムの意味付けを行い，制御部はアルゴリズムの効率を統括すると説いている。この解釈によれば，PROLOG の制御部は節の順序と一つの節の中のゴールの順序だけ*であり，極めて貧弱である。PROLOG を拡張した DURAL は論理部，制御部とも充実しており，述語型プログラミングにより相応しい道具となっている。

DURAL の構文を下図に示す。構文は BNF 記法を拡張した AN 記法で書かれている。| は選択枝を，[ ] は随意要素を，α_{000} は α の任意個の並びを表わす。

```
<program> ::= <statement>...
<statement> ::= <positive literal>[<literal>...] | <goal statement>   †
<goal statement> ::= [] | <negative literal>[<literal>...]
<literal> ::= <positive literal> | <negative literal>
<positive literal> ::= +[<modal symbol>]<atomic formula>
<negative literal> ::= -[<modal symbol>]<atomic formula>
<atomic formula> ::= (<predicate><term>...)
<predicate> ::= <Lisp atom>                                          ††
<term> ::= <Lisp atom> | <variable> | (<function><term>...)
<variable> ::= *<Lisp atom>
<function> ::= <Lisp atom>
<modal symbol> ::= <Lisp atom>
```

DURAL の構文

†) <statement> のことを <clause>（節）と呼ぶ。 *) 後述する cut もある。
††) <LISP atom> は言語 LISP のアトムであり，ここでは定義されていない。

PROLOGで扱うホーン節のゴールには負のリテラルしか許されていない。一方、DURAL ではゴールに正のリテラルが許されており、実行可能述語として扱われる。このような節は相対ホーン節と呼ばれ、DURALの論理能力がPROLOGよりも強力になりえている点である。実行可能述語を用いると、LISPレベルで定義されている関数をDURAL レベルで実行することができるので、処理の高速化がはかれる。正のリテラル +P は、Pを実行した時の値によって次のように解釈される。

①値が丁ならば、統合が失敗したとみなし逆戻りを起こさせる。
②値がNILならば、統合が成功したとみなし、更に導出を進める。
③Pの実行中にエラーが生じた時は、導出を中止する。

ここで、導出原理は一種の反駁法であるから、DURAL レベルの真偽とLISPレベルの真偽とが逆転していることに注意しなければならない。問合せで得られた解を出力する実行可能述語としては次のものが現在用意されている。

ans	解を出力する。
query	強制逆戻りを活用してすべての解を求め、出力する。
average	*query* 同様すべての解を求め、次にその平均を出力する。
sum	*query* 同様すべての解を求め、次にその総和を出力する。

例えば、例2の問合せで解を出力したい時は、本来のゴールの後に *ans* を付けた

```
-(on A *x)+(ans *x)
```

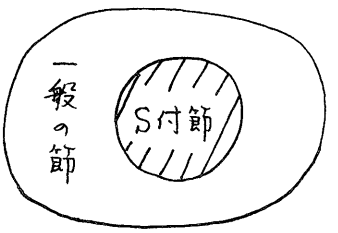
という問合せを用いる。+(quote T) は強制逆戻りを起こさせるので、*query* は

```
+(query *x)+(ans *x)+(quote T)
```

という手続きと同値である。

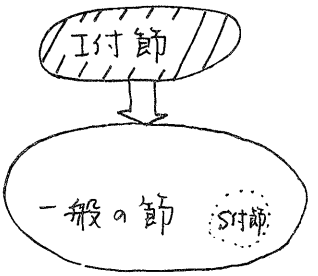
DURAL の第2の特徴は、様相記号（*modal symbol*）を導入した点である。現在提供されている様相記号には **S**, **EX**, **I** の三種類がある。以後、様相記号の付いていない節を「一般の節」と呼ぶことにする。

Sは節の区別を行う様相記号である。S付の節の集合は一般の節の集合の部分集合である（右図）。一般の問合せは一般の節の集合に対して実行されるのに対して、S付の問合せはS付きの節の集合に対してだけ実行される。[7]では、一般の節と高速の節を区別するのにSを用いている。



EX は相対ホーン節と同様に、LISP レベルの関数をDURAL レベルで実行するための様相記号である。EX の場合は、相対ホーン節と異なり、LISPレベルの真偽とDURAL レベルの真偽とが一致する。従って、強制逆戻りを起こさせるためには −EX (quote NIL) と書く。

I はSと同様に、節の区別を行う様相記号である。しかし、I付の節の集合は一般の節の集合と独立しており、I付の問合せも一般の問合せとは処理方法が異なる。I付の問合せは、まずI付の節の集合とS付の節の集合に対して実行されて、一般の問合せが作成され，次に、得られた一般の問合せが実行されるという手順を取る。Iの応用としては、次章で述べる演繹型データベースがある。



DURALでは証明法として，input resolutionだけではなく，unit resolutionも利用できる[9]。unit resolutionとは統合すべき相手の節として必ず単一節を選ぶ導出原理であり，input resolutionとunit resolutionとは，一方で証明できる節は必ず他方でも証明ができるという意味で，能力が同じである。unit resolutionはボトムアップ式証明法，input resolutionはトップダウン式証明法と両者は良き対照をなしているので，デバッグ等に活用できる。

以上のことから，DURALは相対ホーン節，様相記号，unit resolutionといった機能を充実することによって，PROLOGよりはるかに豊富な制御部を持つことになり，述語的プログラミングに適した道具となっている。

3. 演繹型データベースの構成法[10]

3.1 外延データベースと内包データベースの構成

外延データベースは関係データベースを表現する。関係の各タップルは単一節で表現される。タップルの表現法として，属性名を必ず指定するような表現法を採れば，タップル内に属性間の順序はない。それに対して，属性名を指定する必要のない表現法を採れば，タップル内に属性間の順序が仮定されており，タップル内の位置によって属性名が決まる。本データベースでは後者の表現法を採用している。関係のうち，タイプの定義を行っていると考えられるものについては，その関係のタップルをS付の節で表現する。

内包データベースはI付の手続きで表現する。様相記号Iを用いて，内包データベースを外延データベースから区別した理由は，完全性制約をデータベースで扱う対象とするためや，問合せ最適化を行うためである。

例3　演繹型データベース[5]

```
<The intensional data base>

[1] +I(teach Lang *z)-(algebra *z)
[2] +I(teach Knuth *y)-(algorithm *y)
[3] +I(teacher-of *w *u)-I(enrolled *w *v)-I(teach *u *v)

 <The extensional data base>

 +(teach Lang Number1)
 +(teach Knuth Number2)
 +(teach Minsky Number3)
 +(teach Mendelson Logic1)
 +(teach Mendelson Logic2)

 +(enrolled Yamashita Algebra1)
 +(enrolled Yamashita Number3)
 +(enrolled Yamashita Algorithm1)
 +(enrolled Takeuchi Algebra2)
 +(enrolled Takeuchi Algorithm2)
 +(enrolled Takeuchi Algorithm3)
 +(enrolled Saito Logic1)
 +(enrolled Saito Algebra1)
 +(enrolled Oda Logic2)
 +(enrolled Oda Number2)
 +(enrolled Oda Number3)

 +(teacher Lang)
 +(teacher Knuth)
 +(teacher Minsky)
 +(teacher Mendelson)

 +(student Yamashita)
 +(student Takeuchi)
 +(student Saito)
 +(student Oda)

 +(course Algebra1)
 +(course algebra2)
 +(course Algorithm1)
 +(course Algorithm2)
 +(course Algorithm3)
 +(course Logic1)
 +(course Logic2)
 +(course Number1)
 +(course Number2)
 +(course Number3)

 +S(algebra Algebra1)
 +S(algebra algebra2)

 +S(algorithm Algorithm1)
 +S(algorithm Algorithm2)
 +S(algorithm Algorithm3)
```

本データベースでは、一般の節の集合が外延データベースを構成しており、I付の節の集合が内包データベースを構成していると言える。

例3で定義されている8個の関係は次のように分類できる。

i) 外延データベース上だけで定義されている関係 … enrolled, teacher, student, course, algebra, algorithm

ii) 内包データベース上だけで定義されている関係 … teacher-of

iii) 外延データベースと内包データベースの両方の上で定義されている関係 … teach

内包データベースは仮想関係と異なり、teach のように一つの関係が外延データベースと内包データベースの両方にまたがって定義してもよいし、また、teacher-of のように、この teach を用いて定義することもできる。

例3において、algebra と algorithm に関して、内包データベースに次の節

```
[4] +I(course *x)-(algebra *x)
[5] +I(course *x)-(algorithm *x)
```

を加えると、対応する +(course Algebra 1), … という単一節を外延データベースから取除くことができる。[4], [5] で algebra, algorithm の前には様相記号Iがない（もちろん、Iが付いていても良い）。これは、algebra, algorithm が外延データベース上だけで定義されているからである。従って、[1], [2], [3] の algebra, algorithm の前のIを取除いても意味が変らない。問合せは様相記号Iを用いてI付の問合せで表現する。問合せについても内包データベースと関係が無いと予め分かっている場合にはIを付けなくても良い。

3.2 演繹型データベースの処理機構

様相記号Iの付いた問合せ処理は、次の二段階に分かれている。

<I> I付の問合せを内包データベースおよび外延データベースのうちのS付の節の集合の上で演繹処理を行って、一般の問合せを生成する。

<II> <I> で得られた一般の問合せによって外延データベースを検索し、求まった解の和を取ることで、I付の問合せに対する解とする。

<I>の演繹処理は、内包データベース中のI付の節と、S付の節を利用してI付の問合せの変形を行い、可能な限りの変形を行った後で様相記号Iを取除く処理である。内包データベース上で定義される関係は仮想関係ではないので、問合せ変形の過程で様相記号Iを直接取除く「I削除ルール」を使うことができる。

例3のデータベースに対する問合せの処理過程を示す。

例4 「山下君の先生は誰か」

```
(0) -I(teacher-of Yamashita *x)+(query *x)
```

演繹処理過程を下に示す。[0] はI削除ルールである。

```
(1) -(teacher-of Yamashita *x)+(query *x)                  ;by [0] from (0)
(2) -I(enrolled Yamashita *v)-I(teach *x *v)+(query *x)    ;by [3] from (0)
(3) -(enrolled Yamashita *v)-I(teach *x *v)+(query *x)     ;by [0] from (2)
(4) -(enrolled Yamashita *v)-(teach *x *v)+(query *x)      ;by [0] from (3)
(5) -(enrolled Yamashita *v)-(algebra *v)+(query Lang)     ;by [1] from (3)
(6) -(enrolled Yamashita *v)-(algorithm *v)+(query Knuth);by [2] from (3)
(7) -(enrolled Yamashita *v)-(algebra *v)+(ans Lang)       ;by optimization from (5)
(8) -(enrolled Yamashita *v)-(algorithm *v)+(ans Knuth)    ;by optimization from (6)
```

<I>の処理の結果、一般の問合せ (1), (4), (7), (8)が得られる。これらの一般

の問合せは、〈II〉で外延データベースに対して検索が行なわれ、各々，∅，{Minsky}，{Lang}，{Knuth} という解が得られる。従って，(0)に対する解は、これらの和である {Minsky，Lang，Knuth} となる。

〈I〉の処理途中(6)，(7)で得られた一般の問合せは、ゴールを達成するものが一つ見つかれば解が求まるので、query を ans に置き換えるという問合せ最適化を行っている。

例5 「Knuthが教えている生徒は誰か」

```
(0)-I(teacher-of *x Knuth)+(query *x)
```

〈I〉の処理により、次に示す一般の問合せ（外延データベースに対する問合せ）が得られる。

```
(1) -(teacher-of *x Knuth)+(query *x)
(2) -(enrolled *x *v)-(teach Knuth *v)+(query *x)
(3) -(enrolled *x *v)-(algorithm *v)+(query *x)
```

例6 「アルゴリズム1の講義の担当は誰か」

```
(0)-I(teach *x Algorithml)+(query *x)
```

〈I〉の演繹処理の途中段階で次のような一般の問合せが得られる。

```
(1) -(teach *x Algorithml)+(query *x)
(2) -(algebra Algorithml)+(ans Lang)
(3) -(algorithm Algorithml)+(ans Knuth)
```

ここで、S付の節を用いることによって，algebra と algorithm について更に演繹を行い，(2)は削除するとともに，(3)については、

```
(3)' +(ans Knuth)
```

という一般の問合せを得る。従って，最終的に生成される一般の問合せは(1)と(3)'となる。このように，S付の節をタイプ定義に活用することによって問合せの最適化が行える。

例7 「アルゴリズム論に属する講義の担当は誰か」

```
(0) -I(teach *x *y)-I(algorithm *y)+(query *x)
```

〈I〉の演繹処理により、次に示す一般の問合せが得られる。

```
(1) -(teach *x *y)-(algorithm *y)+(query *x)
(2) +(ans Knuth)
```

例8 「山下君の先生で，アルゴリズム1を担当していないのは誰か」

```
(0)-I(teacher-of Yamashita *x)-I(not (teach *x Algorithml))+(query *x)
```

〈I〉の演繹処理により、次に示す一般の問合せが得られる。

```
-(teacher-of Yamashita *x)-(not (teach *x Algorithml))+(query *x)
-(enrolled Yamashita *v)-(teach *x *v)-(not (teach *x Algorithml))+(query *x)
-(enrolled Yamashita *v)-(algebra *v)-(not (teach Lang Algorithml))+(ans Lang)
-(enrolled Yamashita *v)-(algorithm *v)-(not (teach Knuth Algorithml))+(ans Knuth)
```

一番最後の一般の問合せは、not の中が更に演繹ができて，削除されるように思われるが，現在は not の中については何も行っていないので、上記4つの一般の問合せが生成されることになる。しかし、not の定義

```
(5)+(not *p)-*p+(cut not)+(quote T)
(6)+(not *p)
```

を用いて、(0)のI付の問合せをnotの部分について展開したI付の問合せを与えれば、notの中についても問合せ最適化が行えることになる。このnotの定義は、「データベースで未知ものは偽である」という閉じた世界の仮定(Closed World Assumption)を使っている。cutという実行可能述語はそれが実行されるとその後で逆戻りによって、cutの引数である手続きに戻ってきても、その手続き全体が失敗し更に逆戻りするようにさせる制御文である。

4. おわりに

本稿では、述語型言語DURALを用いた演繹型データベースの構成法について述べてきた。DURALはPROLOGに相対ホーン節と様相記号とunit resolutionを付加した言語であり、PROLOGと比べて充実した制御部が言語体系の中にスマートに取り入れられている。演繹型データベースを実現するにあたっては、様相記号を用いて外延データベースと内包データベースとを区別するとともに、更に外延データベースについてもタイプ定義を別の様相記号を用いて一般のデータから分離させた。この結果、問合せ処理にあたっては、内包データベースおよびタイプ定義を用いて演繹処理することにより、大幅な問合せ最適化を行うことができた。

本稿で報告したのはデータベースの検索についてであった。今後、データベースの更新、挿入、削除といったストレッジ演算を実現することによって、データベース完全性の問題を取扱って行く。本データベースの主眼点は、完全性を規定する完全性制約をデータベースの対象にしようという点にある。この非単調変更の問題については今後検討して行く。

DURAL自身の問題としては、input resolution(トップダウン式証明法)のcutといった制御文とunit resolution(ボトムアップ式証明法)との関係、あるいは、様相記号とunit resolutionとの関係、についての検討がある。今後、これらの問題についても研究を進めて行く。

謝辞

DURALについて御教示いただいた横須賀通研データ室後藤調査員(前基一室)に深謝します。また、日頃御指導をいただく山下基一室長、御討論いただいたデータベースグループおよび理論グループの方々に感謝します。

参考文献

[1] Date, C.J.: An Introduction to Database Systems, 2nd ed., Addison-Wesley, 1977.
[2] 有沢: データベース理論, 情処学会, 1980.
[3] Gallaire and Minker (eds.): LOGIC AND DATABASES, Plenum Press, 1978.
[4] Chang, C.L.: DEDUCE 2 : Further Investigation of Deduction in Relational Data Bases, in LOGIC AND DATABASES ([3]), 201-236.
[5] Reiter, R.: Deductive Question-answering on Relational Data Bases, in LOGIC AND DATABASES ([3]), 149-177.
[6] Pereira et al: User's Guide to DECsystem-10 PROLOG, Univ. of Edinburgh, 1978.
[7] Goto, S.: Dural - an extended Prolog language, 京大数解研講究録396, 1980.
[8] Kowalski, R.: Algorithm=Logic+Control, Comm.ACM, vol.22, no.7(1979), 424-436.
[9] 後藤: 述語型言語DURALにおける双モード証明機構, 情処学会全国大会, 1B-5, 1981.
[10] 奥乃: 述語型言語PROLOGのデータベースへの拡張, 情処学会全国大会, 1K-4, 1981.