データベース統合支援システムAdbis(3)：ホーン集合とHSIの仕様

松尾，文碩
九州大学大型計算機センター研究開発部

高木，利久
九州大学大型計算機センター研究開発部

二村，祥一
九州大学大型計算機センター研究開発部

https://doi.org/10.15017/1474966

出版情報：九州大学大型計算機センター広報．14（4），pp.525-549，1981-12-10．九州大学大型計算機センター
バージョン：
権利関係：

データベース統合支援システムAdbis（3）

—— ホーン集合とHSIの仕様 ——

松尾　文碩*　高木　利久*　二村　祥一*

1. はじめに

前稿［１］で予告しておいたように，Adbis／HSIの第１版が完成し，利用者へ公開できるようになったので，今回はHSIの紹介を行う．HSI（Horn Set Interpreter）は，Adbisの推論機能モジュールで，HSIとDMP［１，２］によりAdbisは推論関係型（inferential relational[3]）のデータベース管理システム（DBMS）を構成する．推論関係型データベースシステム**の特徴は，データベースの構成要素である関係が論理式によっても定義できることである．データベース上に実データとして存在する関係を基本関係（base relation又はbasic relation）又は外延的データ（extensional data）と呼び，論理式による関係を仮想関係（virtual relation）又は内包的データ（intensional data）と呼ぶ．推論関係型システムでは基本関係をもとにして複雑な仮想関係が定義できる．データベースに対する質問（query）文には，仮想関係を基本関係と全く同じように使うことができる．しかし，仮想関係型の場合は直ちにデータベースを検索することはできず，'推論'によって検索条件を作り出す．このシステムは，複雑な質問文を扱う能力とは別に，次のような利点がある．

ｉ）基本関係から別の基本関係を作る操作の代りに，仮想関係を定義することによりこの操作をなくすことができる．仮想関係の検索には推論を伴うので速度の点では不利であるが，ディスクスペースを節約でき，レコードの更新，挿入，消去に伴う異常や有害な冗長の発生を回避できる．

ii）基本関係のデータ保全（integrity）の検査に推論機能を使うことができる．つまり，データ保全のための拘束条件を論理式で与えることができる．

これを簡単に言えば，推論関係型システムではデータベースの管理が楽だということである．

HSIが扱う論理式は，**ホーン集合**（Horn set［４］）に限定した．これは，第一階述語計算（first order predicate calculus）における制限された形の論理式であるため，記述能力が非制限のものにやや劣るが，反面，推論が速い［４，５］．又，表現能力に関しても，ホーン集合は完全なプログラム言語として使えることが保証されている［６，７，８］ので，原理的にはプログラムで指定できる記述であればホーン集合によって表現できる．更に，プログラム言語的性質のため，数理論理学の素養のない人でも理解が容易である．

本稿では，まず２節でホーン集合について述べ，次いで３節で推論関係型データベースシステムについて述べる．その後，HSIの仕様を記述する．

2. ホーン集合

まず，図１を見て頂きたい．図１は，ホーン集合の例であり，説明の都合上その構成要素であるホ

* 九州大学大型計算機センター研究開発部

** データベースシステムは，大雑把に言えば，データベース，DBMS，アプリケーションプログラム，利用者によって構成される［９］．

[V1] parent (u , x) , parent (u, v), parent (v, y), man (x) → uncle (x, y)

[V2] father (x, y) → parent (x, y)

[V3] mother (x, y) → parent (x, y)

[V4] father (x, y) → man (x)

[B1] → father (桓武天皇, 葛原親王)

[B56] → father (忠盛, 清盛)

[B57] → father (忠盛, 経盛)

[B60] → father (忠盛, 忠度)

[B71] → father (経盛, 敦盛)

[B78] → father (忠度, 忠行)

[B151] →mother (時子, 知盛)

[B200] →man (敦盛)

図1. ホーン集合の例

ーン節(Horn clause*) に[V 1]とか[B 71]などと名札をつけている. ホーン節の構成要素である" parent (u , x)", " father (忠盛, 清盛)" などは**リテラル**(literal) と呼び, その解釈は次のとおりである.

リ テ ラ ル	解 釈
parent (x, y)	x は y の親である.
man (x)	x は男である.
uncle(x, y)	x は y のおじである.
father (x, y)	x は y の父である.
mother (x, y)	x は y の母である.
father (忠盛, 清盛)	忠盛は清盛の父である.
⋮	⋮

* この形式の論理式を最初に研究したのはMcKinsey [10]である. Horn [11]は, McKinsey の仕事を一般化した. 論理学者は, ホーン節をMcKinsey 論理式と呼んでいる[12 , p. 95]. ホーン節という命名は, 定理の自動証明に従事している人達によって行われた.

ここで，" parent "，" man " などを述語記号（ predicate symbol ）又は述語名（ predicate name）と呼ぶことにする．" x "，" y "，" u "，" v "などは変数であり，その役割は数式の変数と同じである．" 忠盛 "，" 清盛 " などは定数と呼ばれ，その意味も普通の数学におけるものと同じである．さて，**節**（ clause ）とは次のような表現を指す．

$$L_1, L_2, \cdots, L_m \rightarrow M_1, M_2, \cdots, M_n, \qquad (1)$$

ここで，L_1，L_2，…，L_m，M_1，M_2，…，M_nはリテラルであり，$m \geq 0$ 且つ $n \geq 0$．m，$n \geq 1$ のとき，(1)は " L_1 且つ L_2 且つ……且つ L_m ならば M_1 又は M_2 又は……又は M_n " と解釈する．

例えば，［ V 1 ］は，

" u が x の親であり，且つ u が v の親であり，且つ v が y の親であり，且つ x が男ならば x は y のおじである． "

と解釈する．そこで，**ホーン節**とは(1)において $n = 0$ 又は 1 の節のことを言うのである．従って，ホーン節は下記の 4 種に分類される．

名　　称	m	n	表　　現	解　　釈
正節（ positive clause ）	0	1	$\rightarrow M_1$	M_1 は真である．
負節（ negative clause ）	≥ 1	0	$L_1, \cdots, L_m \rightarrow$	L_1 且つ…且つ L_m であることは偽である．
混合節（ mixed clause ）	≥ 1	1	$L_1, \cdots, L_m \rightarrow M_1$	L_1 且つ…且つ L_m ならば M_1．
空節（ empty clause ）	0	0	$\rightarrow$	偽の節である．

図 1 の例では，［ V 1 ］～［ V 4 ］が混合節で，残りはすべて正節である．又，図 1 のようなホーン節の集りをホーン集合と呼ぶことにする．今，Xをあるホーン集合，L_1，…，L_mをリテラルとする．負節 " L_1，…$L_m \rightarrow$ " とXの節によって推論を行い，空節 " $\rightarrow$ " を導くことができたならば，" L_1 且つ…且つ L_m " はXから演繹できるのである．つまり，Xを公理とすると " L_1 且つ …且つ L_m " は定理というわけである．それでは，その 推論規則 ' とは何かということになる．これは**導出**（ resolution ）と呼ばれるものであるが，それを説明するためには少し準備が必要である．

リテラル father（忠盛，清盛）は，リテラル father（ x，y ）の x に忠盛，y に清盛を代入して得られたものと考えることができる．この代入を｛忠盛／x，清盛／y ｝のように，代入要素 " 忠盛／x " と " 清盛／y " の集合として表わすことにする．又，この代入をリテラル father（ x，y ）に施すことを father（ x，y ）｛忠盛／x，清盛／y ｝と書く．代入要素の " ／ " の左辺は，忠盛，清盛のような定数ではなく変数でもよい．例えば，x，y，w，z を変数とすると，代入 $\lambda = \{w/x, z/y\}$ を［ V 2 ］に対して施すと，

$$\text{father}(x, y)\lambda \rightarrow \text{parent}(x, y)\lambda$$

は，

［ V 2′］ father（ w，z ）→ parent（ w，z ）

となる．

［ V 1 ］の左辺の最初のリテラル parent（ u，x ）と［ V 2′］の右辺 parent（ w，z ）に対して $\theta = \{w/u, x/z\}$ と施すと，両方とも同じ parent（ w，x ）になる．［ V 2 ］から［ V 2′］

を得たように［Ｖ２′］と［Ｖ１］の両辺にθを施すと，

［Ｖ２″］ father（w，x）→ parent（w，x）

［Ｖ１′］ parent（w，x），parent（w，v），parent（v，y），man(x) → uncle（x，y）

ここで，［Ｖ２″］と［Ｖ１′］のそれぞれ右辺と左辺にあるparent（w，x）を除いて，両者の右辺と左辺同志をまとめると，

［Ｖ12］ father（w，x），parent（w，v），parent（v，y），man（x）→ uncle（x，y）

が得られる．［Ｖ２″］と［Ｖ１′］から［Ｖ12］を導く過程は，三段論法と同じである．導出とは，この［Ｖ１］と［Ｖ２］から［Ｖ12］を導く過程を言うのである．

［Ｖ２］から［Ｖ２′］を作ったのは，［Ｖ１］と［Ｖ２］に共通の変数x，yを含んでいるためで，両者に共通の変数がないように［Ｖ２］の変数を改名したのである．そこで，このλを**改名代入**（renaming substitution）と呼ぶ．勿論，［Ｖ１］の変数を改名してもよいのであるが，ＨＳＩでは消去するリテラルが右辺にある節を改名するようにしている．さて，parent（u，x）とparent（w，z）を同じリテラルにする代入，これをunifierと呼ぶ，はθだけではない．$\theta'=\{u/w, x/z\}$も$\theta''=\{u/w, z/x\}$もunifierである．ところで，

parent（w，z）θ'＝parent（u，x）＝（parent（u，x）θ）$\{u/w\}$

parent（w，z）θ''＝parent（u，z）＝（parent（u，z）θ）$\{w/u, x/z\}$

このように，どんなunifierで統一したリテラルも，それはθで統一した後，適当な代入によって得られるとき，このθをmgu（most general unifier）と呼ぶ．この場合，θ，θ'，θ''いずれもmguである．もし，２つのリテラルが統一されるならば，必ずmguは存在し，又それを見つけるアルゴリズムが存在する．ＨＳＩのアルゴリズムに関しては付録２を御覧頂きたい．

導出を形式的に表わすと下記のようになる．すなわち，導出とは中央の線の上にある二つの節から下の一つの節を導く'推論'である．

［導出］

$$\frac{\Gamma \rightarrow A \quad \Delta, B, \Lambda \rightarrow \Theta}{\Gamma\eta\sigma, \Delta\sigma, \Lambda\sigma \rightarrow \Theta\sigma}$$

ここで，AとBはリテラル；Γ，Δ，Λ，Θは，有限または空のリテラルの系列；

ηはAとBに対する改名代入；σは$A\eta$とBのmgu．

さて，今図１のホーン集合があるものとして，"忠度は敦盛のおじである"かどうかを問うてみよう．uncle（忠度，敦盛）が導けるかどうかを見るわけだけから，負節"uncle（忠度，敦盛）→"と図１のホーン集合から空節"→"が導出によって得られるかどうかを試してみることになる．図２にこの過程を示した．図２では，９回の導出によって空節が得られた．従って，"忠度は敦盛のおじであるか？"という質問に対しては，答えは"はい"である．図２の過程は，証明すべきことを否定しておいて矛盾を導いているので，ここでは**反証**（refutation）と呼ぶことにする．反証は，本質的に非決定的なしらみつぶしの手順である．図２は，最小の回数で空節に到達している．例えば，［Ｄ１］からparent（u，忠度）を消すのに，［Ｖ２］の代りに［Ｖ３］を選んだとすると決して空節には到達しないだろう．この場合は，反証の過程がどこかで行詰るので後戻り（back－track）

しなければならない．又，［Ｄ１］からparent（u，忠度）をまず消去する代りに，man（忠度）を消去する選択もありうる．図２の例では，これらの選択は反証の速度に関係するだけである．図２

［ＣＱ１］　uncle（忠度，敦盛）→

　　［ＣＱ１］と［Ｖ１］から

［Ｄ１］　parent（u，忠度），parent（u，v），parent（v，敦盛），man（忠度）→

　　［Ｄ１］と［Ｖ２］から

［Ｄ２］　father（u，忠度），parent（u，v），parent（v，敦盛），man（忠度）→

　　［Ｄ２］と［Ｂ60］から

［Ｄ３］　parent（忠盛，v），parent（v，敦盛），man（忠度）→

　　［Ｄ３］と［Ｖ２］から

［Ｄ４］　father（忠盛，v），parent（v，敦盛），man（忠度）→

　　［Ｄ４］と［Ｂ57］から

［Ｄ５］　parent（経盛，敦盛），man（忠度）→

　　［Ｄ５］と［Ｖ２］から

［Ｄ６］　father（経盛，敦盛），man（忠度）→

　　［Ｄ６］と［Ｂ71］から

［Ｄ７］　man（忠度）→

　　［Ｄ７］と［Ｖ４］から

［Ｄ８］　father（忠度，x）→

　　［Ｄ８］と［Ｂ78］から

［Ｄ９］　→

図２．閉質問に対する推論（反証）

の質問は"はい／いいえ"で答えるものであり，これを**閉質問**（closed query）と呼ぶ．一方，負節"uncle（忠度，y）→"で始る図２と同じような反証を考えてみよう．すると，ある反証はyに敦盛を代入することになるだろう．この質問は"忠度のおい又はめいは誰か"ということなので，答えは"敦盛です"となる．しかし，別の反証では，yに重盛が代入されるだろう．これも正しい答えである．このような質問は，**開質問**（open query）と呼ばれ，答えは，uncle（忠度，y）を真とするyの集合｛y | uncle（忠度，y）｝で，多分｛重盛，基盛，宗盛，知盛，…，敦盛，…｝となろう．このように反証を利用して値を求める問題では，特に最初に得られる値を答えとする場合には，反証における，導出の順序の選択方式が重要である．ＨＳＩの方式については，付録２を御覧頂きたい．しかし，開質問に対してすべての値を答えとする場合はあまり意識する必要はない．

以上でホーン集合の説明を終る．概略は一応つかんで頂けたと思うが，きちんと勉強したい人は文献［13］をお読み頂きたい．なお，証明を読むのが苦手の人は文献［14］の方が読み易い．この本にはHornの名前は出てこないが，証明しやすい形の節として実質的にはホーン集合に触れている．

3. 推論関係型データベース管理システム＝ＨＳＩ＋ＤＭＰ

図１のホーン集合において［Ｂ１］以下の節は，すべて正節であり，変数を含んでいない（これを

正基礎節（positive ground clause）と呼ぶ）．ホーン集合は，すべてHSIによって管理できるが，正基礎節はDMPによっても管理できる．正基礎節が大量で，全部を主記憶上に置くことができない場合はDMPによってデータベース化しておかねばならない．今，図1の正基礎節はすべてDMPによって管理することにする．このようにDMPによって管理することになる関係，つまり述語 father，mother，man によって与えられるものを基本関係または外延的データと呼ぶ．又，単に**データベース**と言うときはこの外延的データによるデータベース（外延的データベース）を指すことにする．残りの［Ｖ１］～［Ｖ４］によって定義された関係を仮想関係または内包的データと呼ぶ．従って，仮想関係は，述語 uncle，parent，man によって与えられる関係である．しかし，HSIでは仮想関係と基本関係が同じ名前であることを許していない．図1のままではmanは，仮想関係であり且つ基本関係ということになるので少し修正が必要である．そこで，図1の［Ｖ１］と［Ｖ４］を次のように変え，次の［Ｖ５］を追加する．

［Ｖ１］ parent（u，x），parent（u，v），parent（v，y），vman（x）
→uncle（x，y）

［Ｖ４］ father（x，y）→vman（x）

［Ｖ５］ man（x）→vman（x）

こうすることによって，仮想関係はuncle，parent，vman，基本関係はfather，mother，manとなる．

ここで，図1のホーン集合を2つに分割する前にHSIによる構文*（４．２．１節）でこのホー

```
DEFINE UNCLE;
PARENT(U,X),PARENT(U,V),PARENT(V,Y),VMAN(X)->UNCLE(X,Y);
FATHER(X,Y)->PARENT(X,Y);
MOTHER(X,Y)->PARENT(X,Y);
FATHER(X,Y)->VMAN(X);
MAN(X)->VMAN(X);
->FATHER('KANMUTENNOU','KAZUHARASHINNOU');
          |
          |
->FATHER('TADAMORI','KIYOMORI');
->FATHER('TADAMORI','TSUNEMORI');
          |
          |
->FATHER('TADAMORI','TADANORI');
          |
->FATHER('TSUNEMORI','ATSUMORI');
          |
->FATHER('TADANORI','TADAYUKI');
          |
          |
->MOTHER('TOKIKO','TOMOMORI');
          |
          |
->MAN('ATSUMORI');
          |
          |
END
```

図3．図1のホーン集合のHSIにおける表現

*HSIにおけるホーン節の表現は，GentzenのSequenz［15］に基づいており，Kowalski［16］やPROLOG［17，18］の流儀とは右辺と左辺の位置，矢印の向きが逆である．

ン集合を表現してみる．図3がそれである．これをエディタによってファイルに入れ，これに対してPARS¥H（４．３．１節）を実行させると，このホーン集合は文字の形式から導出を高速に行うためのHSI内部形式に変換される．その後，質問に対する負節を与えてRESO¥H（４．３．３節）を実行させると答えを返してくる．この実行時に，HSIは基本関係以外のすべてのデータを主記憶上にHSI内部形式の形式で持つ．この主記憶上のデータはSAVE¥H（４．３．４節）でファイルに保存でき，REST¥H（４．３．５節）で復元できる．

さて，ホーン集合の分割を考えてみよう．今，［Ｂ１］以下は基本関係として定義することにし，仮想関係は{［Ｖ１］}と{［Ｖ２］，［Ｖ３］，［Ｖ４］，［Ｖ５］}の２つのホーン集合に分割することにする．図4に基本関係のDMP-DDLによる定義を示す．図5に２つのホーン集合を示す．

```
DATABASE   NAME(HEIKE)
           ADMINISTRATOR(F9876)
           MASTERPASSWORD(XXX)
           WRITEPASSWORD(YYY)

RELATION   NAME(FATHER)
           ALIAS(FA)
           VARIABLE(PERSON,PERSON)

RELATION   NAME(MOTHER)
           ALIAS(MO)
           VARIABLE(PERSON,PERSON)

RELATION   NAME(MAN)
           VARIABLE(PERSON)

VARIABLE   NAME(PERSON) TYPE(C20)

END
```

図4.　DMP-DDLによる基本関係の定義

```
DEFINE UNCLE;
GLOBAL PARENT,VMAN;
PARENT(U,X),PARENT(U,V),PARENT(V,Y),VMAN(X)->UNCLE(X,Y);
END



DEFINE PARENT,VMAN;
GLOBAL FATHER,MOTHER,MAN;
FATHER(X,Y)->PARENT(X,Y);
MOTHER(X,Y)->PARENT(X,Y);
FATHER(X,Y)->VMAN(X);
MAN(X)->VMAN(X);
END
```

図5.　分割されたホーン集合

分割されたホーン集合は，それぞれにPARS¥HによってHSI内部形式が作られるが，RESO¥Hのためにこれらを結合編集するのがLINK¥H（４．３．２節）である．実は，図3のように分割されていない場合にもPARS¥Hの後にLINK¥Hを呼ぶ必要がある．この結合作業のために述語名は，そのホーン集合だけに適用する局所名と他のホーン集合によって呼び出される大域名とに分れる．大域名は，参照されるためにはDEEINE文で，参照するためにはGLOBAL文で宣言しておかねばならない．図3でDEFINE文が必要なのは，質問に対する負節（これもホーン集合とみなす）と結合するため

で，FATHERについて質問するためには，これも宣言する必要がある．

HSI第1版では，DMPで関数，ベクトル関数と定義しても，それらはすべて関係とみなす．後で示すように図6で，FATHER，MOTHERは関数として定義しているが，HSIでは，それらを正基礎節の集合とみなし，関数表現として扱わない． DMPによって関数，ベクトル関数として定義した正基礎節のリテラルにおける引数の順序は，最初に関数の引数がきてその後に関数値がくる．2つの部分における順序は，DMP-DDLで定義した順序である．

例として図4の定義を少し変えて図6のようにFATHERとMOTHERを関数として定義してみよ

```
DATABASE   NAME(HEIKE)
           ADMINISTRATOR(F9876)
           MASTERPASSWORD(XXX)
           WRITEPASSWORD(YYY)

FUNCTION   NAME(FATHER) TYPE(C20)
           ALIAS(FA)
           VARIABLE(PERSON)
           INVERSE

FUNCTION   NAME(MOTHER) TYPE(C20)
           ALIAS(MO)
           VARIABLE(PERSON)
           INVERSE

RELATION   NAME(MAN)
           VARIABLE(PERSON)

VARIABLE   NAME(PERSON) TYPE(C20)

END
```

図6. 関数表現のDMP-DDL

う．すると，この規約では少し困った事態が発生する．すなわち，

→FATHER('a','b')

は，'bはaの父親である'を意味することになり，これまでと引数の順序が逆になる．MOTHERについても同様である．ここで，仮想関係PARENT，UNCLEについても引数の順序が逆になったと考えれば図5の定義はこのままでも有効である．但し，'忠度は敦盛のおじか？'という質問文は，

UNCLE('ATSUMORI','TADANORI')→

となる．これが厭ならば，図7のようにVFATHER, VMOTHERを使って引数の順序を逆転する手段もある．更にFATHERをCHILDのように変えて順序を自然にする手段もあろう（この場合は親の性の表現の問題は残る）．

今，"父親でない"という述語を考えてみよう．これを基本関係によって実現するとしたら，一般にFATHERとは比較にならないくらいのディスクスペースを必要とするだろう．もし，FATHERという関係が処理に関係した人間についての父と子の情報を網羅しているとすると，この基本関係にある2人の人間の関係が含まれていないときには，この2人は父と子の関係にないと判断できる．このようなことは，現実によく起る．例えば，汽車の時刻表では，この表にない列車は運転されていないのである．従って，データベースではある関係を定義して，その関係が成立しない要素については否定の関係が成立するとみなす処理の要求がしばしば発生する．そこでHSIではホーン集合を少し

```
DEFINE UNCLE;
GLOBAL PARENT,VMAN;
PARENT(U,X),PARENT(U,V),PARENT(V,Y),VMAN(X)->UNCLE(X,Y);
END


DEFINE PARENT,VMAN;
GLOBAL FATHER,MOTHER,MAN;
VFATHER(X,Y)->PARENT(X,Y);
VMOTHER(X,Y)->PARENT(X,Y);
FATHER(X,Y)->VFATHER(Y,X);
MOTHER(X,Y)->VMOTHER(Y,X);
FATHER(X,Y)->VMAN(X);
MAN(X)->VMAN(X);
END
```

図7. VFATHER，VMOTHERを使った仮想関係定義

拡張して負のリテラル（ negative literal ）を使用できるようにしている．例えば－FATHER（'ATSUMORI'，'KIYOMORI'）は，FATHER（'ATSUMORI'，'KIYOMORI'）が基本関係にないときにその関係が成立するものとみなすのである．但し，仮想関係に対しては負のリテラルの指定ができない．

さて，HSIではプログラムをリテラルによって呼び出すことができる．すなわち，Fortranサブルーチン形式のロードモジュールであれば，CALL文の代りに対応するリテラルを指定することによりホーン集合に取り込むことが可能である．リテラルによって呼び出されるプログラムを**プロシジャ**（ procedure ）と言うことにする．この機能によってHSIは既存ソフトウェア資源を活用することができる．更に，ホーン集合をプログラム言語として使うとき，ホーン集合ではうまく扱えない制御構造を通常のプログラム言語によって救済することができる．

4. Adbis／HSIの仕様

4.1 概　要

Adbis／HSIは，2節で述べたようにホーン集合に対して導出を適用して反証を行うプログラムである．しかし，実際にHSIで扱うホーン集合は，プログラム言語として使い易いように拡張されている．この言語としての特徴については，4.2.1で述べる．

又，反証は導出の高速化を図るために文字列のパターンマッチング方式ではなく，HSI内部形式と呼ばれる表現（節形式をコード化したもの）に対して行うようになっている．さらに，その過程においてHSIは，Adbis／DMPで作成されたデータベースを検索したり，Fortranなどのプログラム言語で作成したサブルーチンプログラム（今後，プロシジャと呼ぶ）を呼び出したりする機能も持っている．これらの処理の概要を4.2.2，4.2.3で述べる．

さて，以上の機能を実現するため，又は，補助するためにHSIは表1．に示す6つのサブルーチンから構成されている．

PARS¥Hはホーン集合を解析してHSI内部形式に変換するサブルーチンである．LINK¥Hは，HSI内部形式とデータベース定義，プロシジャ定義とを結合するものである．これによりHSIはデータベース検索言語として使用できるのである．このサブルーチンは，反証順序の初期設定も行う．

このLINK¥Hの出力であるHSI内部形式に対して反証を行うのがRESO¥Hである．２節で述べた導出原理を用いて，与えられた質問文から答えを求める．なお，反証の終了条件として，CPU時間，導出の深さなどの指定ができるようになっている．

表1. HSIのサブルーチン

サブルーチン名	機　　能
PARS¥H	節形式解析
LINK¥H	結合編集
RESO¥H	反証
SAVE¥H	HSI内部形式保存
REST¥H	HSI内部形式復元
TRAC¥H	トレース出力

SAVE¥H，REST¥Hは，HSIの内部形式を任意の時点で保存したり復元したりするサブルーチンで，これにより反証の中断・再開が容易に行える．

TRAC¥Hは，RESO¥H実行中に蓄積したトレース情報を見易い形で出力するプログラムで，推論・検索の確認に役立つ．

これらのプログラムの使用法については，4.3で述べる．

4.2. ホーン集合

4.2.1 文　法

基本的には，２節で示した節形式と同じであるので，ここではプログラム言語としての特徴や拡張された部分についてのみ説明する．なお，BNF記法で書いた文法を付録１に示しておく．

(1) 定数

i) 数値：Fortranで扱える数値は，複素定数，４倍精度実定数（複素定数）を除いてそのまま書くことができる．内部では，整数型は４バイトの整数，実数型は８バイトの実数として扱われる．

```
1234          4バイトの整数型
123.45        }
1.23E2        } 8バイトの実数型
1.23D2        }
(1.2,3.45)    } エラー
```

ii) 文字列：任意の文字列を引用符で囲んだもの．これもFortranの文字定数と同じである．

```
'A1@B'        }
'KIYOMORI'    } 文字列
'AB''C'       }
3HABC         エラー
```

(2) 変数

英字で始まる８文字以内の英数字列．

```
X             }
Y             } 変数
X1Z           }
1XY           エラー
```

(3) 関数（ベクトル関数）*

関数記号は英文字で始まる8文字以内の英数字列で表わされるが，ベクトル関数の場合はその前に▼#数字▼を置くことにより，何番目の要素であるかを▼#▼の後の数字で表現する．これはAdbis／DMPで作成されたベクトル関数を検索するために追加されたものである．

```
G(X)          } 関数
F(X,Y)
#2V(X,Y,Z)    } ベクトル関数
#1VECT(X)
2F(X)         エラー
```

(4) 述語

英字で始まる8文字以内の英数字列．

HSIではある正基礎節がデータベース上にない場合，その否定がデータベース上に実在するかのように扱うのが可能になっている．これを表現するには，述語の前に－（マイナス）を置けばよい．

```
MOTHER(X,Y)
FATHER('TADAMORI','KIYOMORI')   } 述語
-FATHER(X,'TSUNEMORI')
```

(5) 質問文

質問文は，一般に負節を用いて表現されるが，HSIでは矢印の右側にANSという特別なリテラルが書けるようになっている．これは反証の終了後出力すべき変数の値を示すものである．つまり，負節で表わされた質問文には，yes／noが返され，ANSの付いた質問文には，それを満足するすべての答えが返される．

```
UNCLE(X,'ATSUMORI')->ANS(X);          開質問文
PARENT('TADAMORI','KIYOMORI')->;      閉質問文
UNCLE(X,Y)->ABC(X);
          ->UNCLE(X,Y);               } エラー
```

(6) 仮想関係定義

高級なプログラム言語ではサブルーチン形式プログラムが作成できるが，その機能を実現するためにDEFINE文，GLOBAL文が追加されている．DEFINE文は，例えばFortranのSUBROUTINE文に相当するもので，DEFINE以降の節形式がどんな仮想関係を定義するか示すためのものであり，又，DEFINEの範囲内にある述語はその範囲内のみで有効と考えられる．GLOBAL文は，それに続く名前が外部記号であることを示す．この機能により，他の仮想関係を意識することなく容易にホーン集合が作成でき，又，ホーン集合の蓄積にも役立てられる．

```
DEFINE UNCLE;
GLOBAL FATHER,MOTHER;
PARENT(U,X),PARENT(U,V),PARENT(V,Y),VMAN(X)->UNCLE(X,Y);
FATHER(X,Y)->PARENT(X,Y);
MOTHER(X,Y)->PARENT(X,Y);
FATHER(X,Y)->VMAN(X);
END
```

この定義ではPARENT，VMANという述語記号は，このDEFINEの範囲内でのみ有効である．

* HSI第1版では（ベクトル）関数は扱えない．

(7) 例

```
UNCLE(X,'ATSUMORI')->ANS(X);
DEFINE UNCLE;
GLOBAL PARENT,VMAN;
PARENT(U,X),PARENT(U,V),PARENT(V,Y),VMAN(X)->UNCLE(X,Y);
DEFINE PARENT,VMAN;
GLOBAL FATHER,MOTHER;
FATHER(X,Y)->PARENT(X,Y);
MOTHER(X,Y)->PARENT(X,Y);
FATHER(X,Y)->VMAN(X);
END
```

なお，上記の様にホーン集合が拡張されているので，ANS，DEFINE，GLOBAL，END は予約語となっていて，利用者はその名前を述語記号などには使えない．

4.2.2 解 釈

4.2.1 の文法で記述された質問文や仮想関係定義が与えられると，HSI はまずそれらをHSI内部形式と呼ばれる形式に変換する．これは，反証の高速化を図るためのもので，定数，変数を2バイト，関数記号を4バイト，述語記号を2バイトにコード化したものであり，利用者はそれを意識する必要はない．次に，HSI は，GLOBAL 文で指定した述語の名前とデータベースの関係名，利用者が定義したサブルーチン名とを結合する．又，節形式内の関数記号についても，データベースの（ベクトル）関数や利用者の関数副プログラムと結合する．以上の前準備が完了すると，2節で述べた導出原理に従って反証を行う．反証の過程において，データベースの検索やサブルーチンの呼出しが可能になるとHSI はHSI 内部形式の定数や変数をそれらの定義に合うように型変換を行い，検索処理やプログラムの起動を行う．このようにして空節に達すると反証を終了し答えを出力する．

4.2.3 プロシジャとの結合

前節で述べたように，Adbis／HSI は，反証中にデータベースの検索やプロシジャの呼出しを行う．データベースの検索においては，HSI はDMP のサブルーチン群を用いるので，データベース名さえ与えられれば検索条件の設定や型変換などを自動的に行うようになっている．しかし，利用者が作成しだプロシジャの実行をHSI が行うには外部からプロシジャの仕様を与えなければならない． ここでいう仕様とは，サブルーチン名，属性（サブルーチン副プログラムか関数副プログラムかの区別），引数の個数，引数の属性（型，長さ，入力引数か出力引数かの区別）を示す．これらを定義するには，プロシジャ定義言語HSI－PDL（Procedure Definition Language ）を用いる．

(1) プロシジャ定義言語

i) プロシジャ定義ファイルは，レコード形式がF 又はFB で，レコード長が 80 バイトの順データセットあるいは区分データセットのメンバでなければならない．

ii) 書式

・コマンドは1けたから72けたの間，どこに書いてもよい．

・コマンドとオペランド，オペランドとオペランドの区切りは1個以上空白を置く．

・コマンド，及びオペランドのキーワードは他と区別可能であれば任意の先頭部分が省略形となる．

1 （コマンド）	（オペランド） 72	73 80
{ SUBROUTINE / FUNCTION	NAME（{サブルーチン名 / 関数名}） NUMBER（引数の個数） PARAMETER（型と長さ（[{IN / OUT}]）， 型と長さ（[{IN / OUT}]），…）	

iii） コマンドの説明

サブルーチン副プログラムか関数副プログラムかの区別を指定する．

iv） オペランドの説明

a．NAME（{サブルーチン名 / 関数名}）

サブルーチン名（関数名）を指定する．6文字以内の英数字列（ただし先頭は英字）．

b．NUMBER（引数の個数）

引数の個数を指定する．

c．PARAMETER（型と長さ（[{IN / OUT}]），…）

型と長さは，I2，I4，R4，R8，CKの中から選ぶ，入力，出力の区別は，IN又はOUTを指定する．関数副プログラムの場合はIN，OUTは省略可能．但し，関数副プログラムの場合は，最後に，関数値の型と長さを書く．

定義例

```
SUBROUTINE NAME(ADD) NUMBER(3)
           PARAMETER(I4(IN),I4(IN),I4(OUT))
```

4.3. Adbis/HSIサブルーチン

この節では，Adbis/HSIの各サブルーチンの使用法を述べる．使用法の説明において次の記法を使用する．なお，使用例はFortran 77文法で書かれている．

P_n ：復帰時に内容が変化しない引数を表わす．

$\boxed{P_n}$ ：復帰時に内容が変化する引数を表わす．

I ：整数を表わす．

R ：実数を表わす．

A ：文字列を表わす．

array：配列を表わす．

V ：変数を表わす．

*n ：nバイトの長さであることを示す．

[…] ：省略可能であることを示す．

△ ：空白を表わす．

4.3.1 PARS¥H (parse)

(1) 機能

ホーン集合（付録１）をHSI内部形式（4.2.2節）に変換する．

(2) 呼出し形式と引数の説明

CALL PARS¥H(P_1 , P_2 , P_3 , P_4 , P_5 , P_6 , P_7)

P_1 ：完了コード（Iv*4）

0＝正常終了

１００１＝ホーン集合の終わりを示す文字列ENDがない．

１００２＝DEFINE(GLOBAL)に対応するセミコロン（；）がない．

１００３＝カンマ（，）がない．

１００４＝８文字を越える，又は，文法的に正しくない名前（文字列）がある．

１００５＝かっこ‘（’，‘）’の対応がとれない．

１００６＝引用符‘▼’の対応がとれない．

１００７＝引数の個数が不一致の述語（関数）記号がある．

１００８＝節の形式が間違っている．（矢印の右側にリテラルが２個以上あるなど．）

１００９＝質問文の形式が間違っている．（矢印の右側にANS以外のリテラルがあるなど．）

１０１０＝質問文が２個以上ある．

２００１＝第４引数（ P_4 ）の領域が不足している．

３００１＝作業領域が足りない．（リージョンサイズ不足）

９００１＝関数記号が使われている．（HSI第１版では関数記号は使用できない．（４．５節））

P_2 ：エラー位置（Iv*4）

エラーを起こした箇所を入力ホーン集合（ P_6 ）の先頭からのバイト数で示す．

P_3 ：エラー文字列（Av*8, Aarray * 8 ）

エラーを起こした述語（関数）記号，又は，文字列が設定される．８文字以上の文字列に関しては先頭の８文字のみが返される．

P_4 ：HSI内部形式格納領域（Iarray ）

変換されたHSI内部形式が設定される．

P_5 ：第４引数（ P_4 ）の領域のバイト長（I*4, Iv*4）

P_6 ：ホーン集合（Av, Aarray ）

ホーン集合（付録１）を指定する．

P_7 ：第６引数（ P_6 ）の領域のバイト長（I*4, Iv*4）

(3) 使用例

```
INTEGER HSI(1000)
CHARACTER SOURCE*200
CHARACTER ERR*8
      |
SOURCE='UNCLE(X,''ATSUMORI'')->ANS(X);DEFINE---'
      |
CALL PARS¥H(IRC,INDERR,ERR,HSI,4000,SOURCE,200)
      |
```

4.3.2 LINK¥H(link and edit)

(1) 機能

HSI内部形式とデータベース定義，プロシジャ定義を結合編集し，新たにHSI内部形式を出力する．又，反証順序の初期設定なども行う．

(2) 呼出し形式と引数の説明

```
CALL LINK¥H( P1 , P2 , P3 , P4 , P5 , P6 , P7 , P8 )
```

P1 ：完了コード（ Iv*4 ）

0＝正常終了

1～50＝Adbis／DMPのサブルーチンでエラーが発生した．第2引数（ P2 ）の先頭6バイトにDMPのサブルーチン名が返されるので，エラーの詳細については文献[1]を参照のこと．

1001＝第7引数（ P7 ）で指定したデータベースが存在しない．

1002＝第7引数（ P7 ）で指定したデータベースのパスワードが間違っている．

1003＝プロシジャ定義ファイル（ DD名＝FT40F001 ）が割り当てられていない．

1004＝プロシジャ定義ファイルのリードエラー．

1005＝プロシジャ定義エラー．第2引数（ P2 ）の先頭6バイトにプロシジャ名が返される．

1006＝第8引数（ P8 ）で指定したLINK制御情報が間違っている．

1007＝第4引数（ P4 ）がHSI内部形式でない．

1008＝第6引数（ P6 ）のHSI内部形式が正しくない．

1009＝未定義の，又は，引数の個数が不一致の述語（関数）記号が存在する．
エラーを起こした名前は，第2引数（ P2 ）に設定される．

2001＝第4引数（ P4 ）のHSI内部形式格納領域が足りない．

2002＝第2引数の領域が足りない．

3001＝作業領域不足（ リージョンサイズ不足 ）

P2 ：エラー設定領域（ I*4 array ）

完了コードが1009の場合，次の形式でエラー起こした名前が設定される．

エラー述語，関数の個数（ n ）	I*4
（本配列中の）個数（ m ）	I*4
エラー種別1	I*4
述語（関数）名1	A*8
⋮	
エラー種別m	I*4
述語（関数）名m	A*8

・エラー種別

11：未定義の述語

12：未定義の関数

21：引数の個数が不一致の述語

22：引数の個数が不一致の関数

完了コードが１～５０の場合は先頭の６バイトにAdbis／DMPのサブルーチン名が入る．

P_3 ：第２引数の領域のバイト長（ I*4, Iv*4 ）

P_4 ：HSI内部形式格納領域（ I array ）

この引数は，呼出し時は結合対象のHSI内部形式として用いられ，第６引数（ P_6 ）のHSI内部形式，第７引数で定義されるデータベースなどと結合される．復帰時には，その結合結果が設定される領域である．

P_5 ：第４引数（ P_4 ）の領域のバイト長（ I*4, Iv*4 ）

P_6 ：HSI内部形式（Iarray ）

結合すべきHSI内部形式を指定する．

P_7 ：データベース定義情報（ A*21 ）

課題名（ F＋課題番号 ）（ ５文字），データベース名（ ８文字），パスワード（ ８文字 ）をこの順序で指定する．但し，データベース名は空白を省略してはならない．

P_8 ：リンク制御情報（ I*4, Iv*4 ）

３桁の数で与える．

１００の位　０…データベース定義情報に関して第４引数の内容を変更しない．

１…データベース定義情報に関して第４引数の内容を新規作成する．

既に存在する場合は置換える．

１０の位　０…プロシジャ定義に関して第４引数の内容を変更しない．

１…プロシジャ定義に関して第４引数の内容を新規作成する．

既にある場合は置換える．

１の位　０…仮想関係定義（ 質問文も含む ）に関して第４引数の内容を変更しない．

１…仮想関係定義（ 質問文も含む ）に関して第４引数の内容を新規作成する．

既にある場合は置換える．

２…仮想関係定義を追加する．

(3) 使用上の注意

プロシジャと結合するときは，プロシジャ定義ファイル（ DD名＝FT40F001 ）（ ４．２．３節）を割り当てておく必要がある．

(4) 使用例

```
INTEGER HSI(100000),HSI1(1000)
INTEGER ERR(100)
CHARACTER DBINF*21
      |
DBINF='F1234DBNAME  PSWD'
      |
CALL PARS¥H(--------------,HSI1,4000-----------)
      |
CALL LINK¥H(IRC,ERR,400,HSI,400000,HSI1,DBINF,111)
      |
```

4.3.3 RESO¥H (resolve)

(1) 機能

反証を行う．各種の打ち切り条件やトレース出力の指定ができる．

(2) 呼出し形式と引数の説明

```
CALL RESOLVH(P1 , P2 , P3 , P4 , P5 , P6 , P7 , P8 )
```

P_1 ：完了コード（ Iv*4 ）

0＝正常終了

－1＝第6引数（ P_6 ）の第1番目の打ち切り条件を満足して終了した．

－2＝ 〃 2 〃

－3＝ 〃 3 〃

－4＝ 〃 4 〃

1～50＝Adbis／DMPのサブルーチンでエラーが発生した．第2引数（ P_2 ）の先頭6バイトにDMPのサブルーチン名が返されるので，エラーの内容については参考文献［1］を参照されたい．

1001＝プロシジャのロードモジュール（ DD名＝STEPLIB，又は，JOBLIB ）が割り当てられていない．

1002＝第5引数（ P_5 ）のHSI内部形式が正しくない．

1003＝トレース用ファイル（ DD名＝FT41F001 ）が割り当てられていない．

1004＝打ち切り条件の指定が間違っている．

1005＝トレースの指定方法が正しくない．

2001＝第3引数（ P_3 ）の解答格納領域が足りない．

2002＝第5引数（ P_5 ）の領域不足．

3001＝作業領域不足．（リージョンサイズ不足）

P_2 ：エラーサブルーチン名（ A*6 ）

データベース検索時にエラーを起こしたAdbis／DMPのサブルーチン名が返される．

P_3 ：解答格納領域（ I*4 array ）

次の形式で答えが返される．

質問文の形式	I*4	・質問文の形式
答えの個数（ n ）	I*4	0：閉質問
（本配列の）答えの個数（ n_1 ）	I*4	1：開質問
ANSの引数の個数（ m ）	I*4	閉質問文の場合，答えの個数には，
答え1の第1番目の値の型と長さ	I*4	0：no
答え1の第1番目の値		1：yes
⋮		が設定される．
答え1の第m番目の値の型と長さ	I*4	・型と長さ
答え1の第m番目の値		次の式で決められる．
⋮		長さ（単位バイト）×10＋型

答えn_1の第1番目の値の型と長さ	I*4	型＝ 1 整数型；2 実数型；3 文字型．
答えn_1の第1番目の値		
⋮		
答えn_1の第m番目の値の型と長さ	I*4	
答えn_1の第m番目の値		

P_4 ：第3引数（P_3）の解答領域のバイト長（I*4，Iv*4）

P_5 ：HSI内部形式（Iarray）

LINK¥Hの出力を指定する．

この領域は反証の作業領域としても使用される．

P_6 ：第5引数（P_5）のバイト長（I*4，Iv*4）

P_7 ：反証の打ち切り条件（I*4 array）

CPU時間（単位秒）	I*4	0を指定すると，その条件での打ち切りは行わない．
導出回数	I*4	
導出の深さ	I*4	
答えの個数	I*4	

P_8 ：トレース情報（I*4，Iv*4）

反証のトレース情報を必要とするならば1，必要ないならば0を指定する．

(3) 使用上の注意

・プロシジャを使用する場合は，そのロードモジュールを以下の様に割り当てておく必要がある．

i）バッチ処理

DD名＝STEPLIB，又は，JOBLIBに割り当てる．

ii）TSS

LIBRARYコマンドを用いてタスクライブラリとしておく．

・トレース情報を出力する場合は，トレース用ファイルをDD名＝FT41F001に割り当てておかなければならない．

なお，トレース用ファイルは，レコード形式がF又はFBで，レコード長が80バイトの順データセットあるいは区分データセットのメンバでなければならない．

・第5引数（P_5）は，反証過程のリテラルやデータベース検索により得られた定数が格納されるので，なるべく大きい領域をとる必要がある．

(4) 使用例

```
INTEGER HSI(100000)
INTEGER ANSWER(200)
INTEGER COND(4)
CHARACTER ERR*6
     |
CALL LINK¥H(-----------,HSI,400000,--------------)
     |
COND(1)=600
COND(2)=100
COND(3)=50
COND(4)=10
CALL RESO¥H(IRC,ERR,ANSWER,800,HSI,400000,COND,1)
     |
```

4.3.4 SAVE¥H (save Horn set in internal form)

(1) 機能

HSI内部形式をファイルに保存する．このHSI内部形式は，PARS¥H, LINK¥H, 及びRESO¥Hのどの出力でも構わない．

(2) 呼出し形式と引数の説明

```
CALL SAVE¥H(P1 , P2 )
```

P_1 ：完了コード（ Iv*4）

0＝正常終了

1001＝SAVE用ファイル（ DD名＝FT42F001 ）が割り当てられていない．

1002＝第2引数（ P_2 ）がHSI内部形式でない．

P_2 ：HSI内部形式（ Iarray ）

HSI内部形式を指定する．

(3) 使用上の注意

・このサブルーチンを呼出す前に，SAVEファイルをDD名＝FT42F001に割り当てておく必要がある．

なお，SAVE用ファイルは，レコード形式がF又はFBで，レコード長が80バイトの順データセットあるいは区分データセットのメンバでなければならない．

(4) 使用例

```
INTEGER HSI(10000)
      |
CALL SAVE¥H(IRC,HSI)
      |
```

4.3.5 REST¥H (restore Horn set in internal form)

(1) 機能

SAVE¥Hを用いてファイルに保存されたHSI内部形式を復元する．

(2) 呼出し形式と引数の説明

```
CALL REST¥H(P1 , P2 , P3 )
```

P_1 ：完了コード（ Iv*4 ）

0＝正常終了

1001＝SAVE用ファイル（ DD名＝FT42F001 ）が割り当てられていない．

1002＝SAVE用ファイルの内容が正しくない．

2001＝第2引数（ P_2 ）の領域不足

P_2 ：HSI内部形式格納領域（ Iarray ）

P_3 ：第2引数（ P_2 ）のバイト長（ I*4, Iv*4 ）

(3) 使用上の注意

・REST¥Hを使うには，あらかじめSAVE用ファイルをDD名＝FT42F001に割り当てておく必要がある．

(4) 使用例

```
INTEGER HSI(10000)
        |
CALL RETR¥H(IRC,HSI,40000)
        |
```

4.3.6 TRAC¥H (trace)

(1) 機能

サブルーチンRESO¥Hで出力したトレース用ファイルを見易い形で出力する．

(2) 呼出し形式と引数の説明

CALL TRAC¥H(P_1)

P_1 ：完了コード

0＝正常終了

1001＝トレース用ファイル（DD名＝FT41F001）が割り当てられていない．

1002＝トレース用ファイルの形式が正しくない．

1003＝トレース出力用にDD名＝FT06F001が割り当てられていない．

(3) 使用上の注意

・TRAC¥Hを使用するには，トレース用ファイル（DD名＝FT41F001）と出力用ファイル（DD名＝FT06F001）を割り当てておかなければならない．

(4) 使用例

```
CALL TRAC¥H(IRC)
```

4.4 標準関数

標準的な述語及び関数として次のものが用意されている．

述語名と引数	関数名と引数	意味 注1)
PLUS(P_1 , P_2 , P_3)	PLUS(P_1 , P_2)	$P_1+P_2=P_3$
MINUS(P_1 , P_2 , P_3)	MINUS(P_1 , P_2)	$P_1-P_2=P_3$
TIMES(P_1 , P_2 , P_3)	TIMES(P_1 , P_2)	$P_1*P_2=P_3$
DIVIDE(P_1 , P_2 , P_3)	DIVIDE(P_1 , P_2)	$P_1/P_2=P_3$

注１）関数の場合は，P_3 ではなく関数値としてその値が入る．

又，引数の型は数値であればよく，型変換はFortranの演算子（＋，－，＊，／）と同様に行われる．

上記以外にも，Fortranの組込み関数が自由に使える．

なお，上記の述語（関数）名，及びFortranの組込み関数名と利用者が定義した述語（関数）名と一致した場合は利用者の定義が優先する．

4.5 Adbis/HSI使用上の制限事項

Adbis/HSI第1版における使用上の制限は次のとおりである．

・前節までの説明及び例において関数記号が使われているが，第1版では関数表現は扱えないのですべて述語の形式で記述しなければならない．そのため，データベース上の（ベクトル）関数を参照する場合は，それを表形式とみて述語とする必要がある．又，FUNCTION型のプロシジャを組み込むことは許されない．Fortranの組込み関数を使う場合は，引数の最後にもう一つ引数をつけ加えて使用し，その引数によって値を受け取る必要がある．

・Adbis/HSIを利用するアプリケーションプログラムは，関数副プログラム，サブルーチン副プログラムの名前として，次のものは使用できない．

1）¥Hで終わる6文字の名前

2）FACOM OS Ⅳ/F4，TACライブラリ［6］のうち，次のもの．

基本ライブラリ……QANUCK, QAPHCK, QBITOF, QBITON, QBITCK, QDBLNK, QNUCHK, QSBSTR

サービスライブラリ…FR¥MAN, GT¥MAN, LM¥CHK, LM¥LDG, ¥BLNKC

データセットライブラリ…D¥DDCK

・Adbis/HSIは，データベースの検索にはAdbis/DMPのサブルーチン群を利用するので，データベースの検索を行うときは，Adbis/DMPの使用制限も守らなければならない［1］．

5. おわりに

Adbisの開発計画は，DMPとHSIの第1版の完成によってひとまず初期の目的を達成した．今後は，効率の改善を図ると共にNLC（Natural Language Conventer）［1］や知識ベース管理機構を開発する計画である．又，サブルーチン形式の基本インターフェイスだけではなくハイレベルのインターフェイスも提供したいと考えている．

参考文献

1. 松尾，二村，高木　データベース統合支援システムAdbis(1) ― Adbisの概要とDMPの仕様 ―，九大大型計算機センター広報，**14**，2，1981，172－217．
2. 二村，松尾，高木　同(2) ― DMPのユーティリティプログラムと機能追加 ― ，同上，**14**，3，1981，388－398．
3. Minker, J. Search Strategy and Selection Function for an Inferential Relational System, ACM Trans. Database Syst., **3**, 1, 1978, 1－31.
4. Henschen, L. and Wos, L. Unit Refutations and Horn Sets, J. Assoc. Comput. Mach., **21**, 4, 1974, 590－605.
5. Kuehner, D. Some Special Purpose Resolution Systems, in Machine Intelligence 7 (B. Melzer and D. Michie, eds.), American Elsevier, New York, 1972, 117－128.

6. Hill, R. RUSH Resolntion and its Completeness, DCL Memo 78, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1974.
7. Andreka, H. and Nemeti, I. The Generalized Completeness of Horn Predicate Logic as a Programming Language, D. A. I. Res. Rep. 21, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1976.
8. Tärnlund, S.-Å. Horn Clause Computability, BIT, **17**, 2, 1977, 215 — 226.
9. Date, C. J. *An Introduction to Database Systems, 3rd ed.*, Addison-Wesley Pub. Co., Reading, Mass., 1981.
10. McKinsey, J. C. C. The Decision Problem for Some Classes of Sentences without Qualifiers, J. Symbolic Logic, **8**, 1943, 61 — 76.
11. Horn, A. On Sentences which are Tree of Direct Unions of Algebras, *ibid.*, **16**, 1, 1951, 14 — 21.
12. Shoenfield, J. *Mathematical Logic*, Addison-Wesley Pub. Co., Reading, Mass., 1967.
13. Loveland. D. W. *Automated Theorem Proving: A Logical Basis*, North-Holland Pub. Co., Amsterdam, 1978.
14. Chang, C. L. and Lee, R. T. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.
15. Gentzen, G. Untersuchungen über das logische Schliessen, Math. Z., **39**, 1935, 176 — 210, 405 — 431.
16. Kowalski, R. Predicate Logic as Programming Language, Information Processing 74, North-Holland, Amsterdam, 1974, 569 — 574.
17. Battani, G. and Meloni, H. Interpreteur dur language de programmation PROLOG, Groupe de l Intelligence Artificielle, U. E. R. de Luminy, Marseille, 1973.
18. Roussel, J. A. PROLOG, Manual de reference et d'utilization, U. E. R. de Luminy, Marseille, 1975.
19. 計算機マニュアル FACOM OS IV/F 4 TAC／LIB解説書（64AR-9500-1），富士通（株）.

付録 1. ホーン集合の構文

```
          SYNTAX(BNF)

<digit> ::= 0|1|2|3|4|5|6|7|8|9

<capital letter> ::= A|B|C|---------|X|Y|Z

<small letter> ::= a|b|c|---------|x|y|z

<special symbol> ::= !|"|#|$|%|&|(|)|*|=|~|:|-|^|\|
                     +|`|{|}|<|>|?|;|@|_|[|]|,|.|/||

<sign> ::= +|-

<unsigned integer> ::= <digit>|<unsigned integer><digit>

<integer> ::= <unsigned integer>|<sign><unsigned integer>

<decimal fraction> ::= .<unsigned integer>

<decimal number> ::= <unsigned integer>|<decimal fraction>|
                     <unsigned integer><decimal fraction>|
                     <unsigned integer>.

<exponent part> ::= E<integer>

<unsigned number> ::= <decimal number>|<exponent part>|
                      <decimal number><exponent part>

<number> ::= <unsigned number>|<sign><unsigned number>

<alphabet> ::= <capital letter>|<small letter>

<identifier> ::= <alphabet>|<identifier><alphabet>|
                 <identifier><digit>

<variable> ::= <identifier>                    * length<=8

<constant> ::= <number>|'<string>'

<string> ::= <alphabet>|<digit>|<special symbol>|''|<string><string>

<predicate symbol> ::= <identifier>            * length<=8

<function symbol> ::= <identifier>             * length<=8

<vector function symbol> ::= #<unsigned integer><function symbol>

<term> ::= <constant>|<variable>|<function>

<term list> ::= <term>|<term list>,<term>

<function> ::= <function symbol>(<term list>)|
               <vector function symbol>(<term list>)

<atom> ::= <predicate symbol>(<term list>)

<negation of atom> ::= -<atom>

<literal> ::= <atom>|<negation of atom>

<literal list> ::= <literal>|<literal list>,<literal>
```

```
<arrow> ::= ->

<negative clause> ::= <literal list><arrow>

<positive clause> ::= <arrow><atom>

<mixed clause> ::= <literal list><arrow><atom>

<empty clause> ::= <arrow>

<closed query> ::= <negative clause>

<open query> ::= <negative clause><answer atom>

<answer atom> ::= ANS(<term list>)

<query> ::= <closed query>|<open query>

<virtual relation definition> ::= <virtual relation dcl part>
                                  <global relation dcl part>
                                  <non-negative clauses>

<virtual relation dcl part> ::= DEFINE <predicate list>;

<global relation dcl part> ::= <empty>|GLOBAL <predicate list>;

<predicate list> ::= <predicate symbol>|
                     <prediacte list>,<predicate symbol>

<non-negative clauses> ::= <mixed clause>;|<positive clause>;|
              <non-negative clauses><non-negative clauses>

<empty> ::=

<virtual relation> ::= <virtual relation definition>|
                       <virtual relation><virtual relation>

<born set> ::= <query>END|<virtual relation>END|
               <query><virtual relation>END
```

付録2．導出と統一化のアルゴリズム

1. 仮想関係定義内の変数名を質問文または反証中に変換された質問文（今後，質問文と書く）内の変数名と一致しないように変更する．

2. 質問文から一つ述語を選ぶ（ANS述語は選ばない）．なお，4．5又は3から飛んできた場合は，以前選んだものと別の述語を選ぶ．又，もう選ぶものがなくなった場合は，終了する．

   述語の選び方は以下のように行う*

   質問文中の述語についてそれぞれ統一可能な仮想関係定義，データベース，プロシジャをすべて求め，次の順序に整列する．そしてこの順序が一番高いものと統一可能な述語を選ぶ．

	順位
プロシジャ	高
データベース	↓
仮想関係（明らかな再帰的定義を含まない）	↓
仮想関係（明らかな再帰的定義を含む）	低

   上の順序だけで一意に選択できない場合は，その述語内の定数の割合が高いものを選ぶ．

   それでも決まらない場合は，その中で最も左側の述語を選ぶ．

3. 2で選んだ述語名と同じ名前を矢印の右側に持つ仮想関係定義をすべて見つける．複数個ある場合は，その中から1つ選ぶ．なお，4．5から飛んできた場合は，以前とは別の仮想関係を選ぶ．

   又，もう選ぶものがなくなった場合は2へ行く．

   複数個存在する場合は，明らかな再帰的定義を含まないものを選ぶ*

   それで一意に決まらないときは，節内の述語数の少ないものを選ぶ．

   以上の条件が同じならば，先に定義されたものが優先する．

4. 2で選んだ述語（Qと書く）の引数と3で選んだ仮想関係定義の矢印の右側の述語（Vと書く）の引数について，以下の統一化を行う．

4.1 i を1とする．

4.2 Qの第 i 番目の引数とVの第 i 番目の引数を比較し，等しければ4.7へ，そうでなければ4.3へ行く．

4.3 Qの第 i 番目の引数が変数ならば4.4へ，そうでなければ4.5へ行く．

4.4 Q内の第 i 番目の引数と同じ名前をもつ変数をすべてVの第 i 番目の引数で置き換える．

これは，Q，V両方に適用する．なお，この代入規則を保存する．4.7へ行く．

4.5 Vの第 i 番目の引数が変数ならば4.6へ，そうでなければ，いままで施した置換を復元し，3へ行く．

4.6 V内の第 i 番目の引数と同じ名前をもつ変数をすべてQの第 i 番目の引数で置き換える．

これは，Q，V両方に適用する．なお，この代入規則を保存する．

4.7 i に1を加える．i が引数の個数より大きければ5へ行く．それ以外は4.2へ行く．

5. 4で得られた代入規則を質問文に適用する．

6. 空節ならば終了し，そうでなければ1へ行く．

---

* この選び方は将来変更する可能性がある．