KAU system の 実 施

山 内 平 行・大須賀 節 雄

An Implementation of KAU system

By
Hiroyuki YAMAUCHI and Setsuo OHSUGA

Abstract: In this paper, we concern the expeimental KAU (Knowledge Acquisition and Utilization) system which was realized in accordance with the design philosophy presented in the paper [*1*] and [*2*].

In the first, the outline of the system is described, where the deductive inference mechanisms, especially, the search method of information contained in the knowledge base and the algorithm of the test of implicative conditions (TIC) are mainly concerned. Then, the formalization of the system's own external language features and their uses for knowledge representations and database definitions are given. Finally, we give the method of the automatic program generation for database accesses and three examples are illustrated with Appendix [A] and [B].

摘　　要

本報告は，知識取得・利用システム KAU の基本設計概念[*1*], [*2*] に基いて試作された実験システムに関する．

はじめにシステムの概略がのべられ，推論機構，特に知識の探索および含意判定（TIC）のしかたが詳述される．次に，システム固有の言語形式とその表現例が与えられ，続いて，データベースアクセスのための自動プログラム生成についてのべられる．最後に3つの実験例について概略説明される．

1. は じ め に

本報告は，知識取得・利用システム—KAUの基本設計概念[*1*],[*2*]に基づいて試作された実験システムに関し，これによる各種知識表現の実際と三つの実験例について述べる．

最初に，実験システムの概要が述べられ，次にシステム固有の言語形式とこれを用いた各種知識表現のしかたが述べられる．続いて，データ・ベース・アクセスのためのプログラム生成の方法が述べられ，最後に三つの実験例を示す．巻末にこれらの例に必要な知識および生成されたプログラムの実際を掲げ，その説明の概略が施される．

2. システムの概要

入力は，システム固有の拡張された述語形式で行われ，知識ベースの知識の移植，実験用データベースのデータ蓄積，システムへの質問などの表現がなされる．知識ベースは実体語のそれぞれの意味関係をグラフ構造で表現したスケルトン構造と，知識ユニットと呼ばれるアトムのAND・OR表現のセットなどからなる．データベースは関係型データベースを扱う．

入力された質問は，推論・探索機構によってそれと等価な意味をもつ表現に順次変換され，最後に直接的に評価可能な手続きノード（PTA）のみからなるAND・OR構造が生成される．これを中間手続き表現と呼ぶ．このとき，知識ベース内の知識が利用される．

この中間手続き表現は，ファイル操作型プログラムの生成機構によって，データベース・アクセスおよび処理順序を表わすシーケンシャルなプログラムに変換される．生成されたプログラムは，実行機構により実際に実行され実験用データベースに実際にアクセスしその結果を出力する．この実行機構は，KAUシステムを各種データベースの知的フロント・エンド・システムとして考えたばあい，実際の実行は外部のDBMSに委ね，KAUシステムより分離してもよいものである．実験システムでは，関係データベースに対して生成されたプログラムの実行機構が用意される．図1に実験システムの構成図を示す．

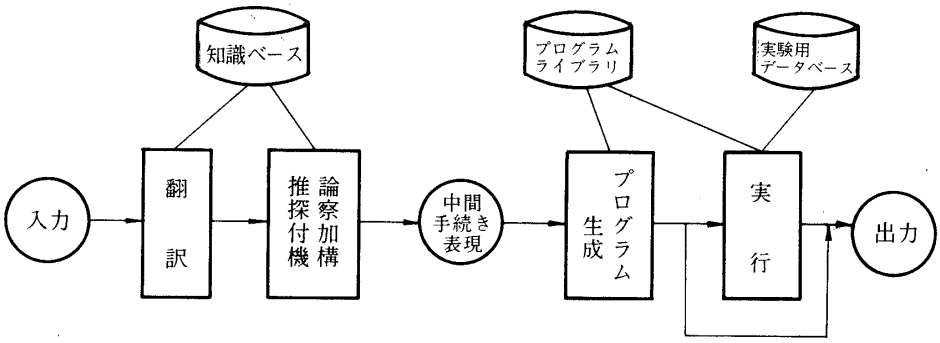


図1 実験システムの構成

2.1 知識ベースとデータベース

計算機システムを人間との知識の交換が円滑に行える一つの知能的実体とするためには，一つに計算機は人のことばによる情報表現を理解しその意味処理ができなければならない．この意味処理が可能であるためには，計算機は語および文の意味を理解するための一つの知識体系をもたなければならない．しかし，一般にこのような意味処理を行うと計算機の応答速度がそれだけ遅くなる欠点があるので，これを速めるような知識の体系づけおよび探索処理メカニズムを計算機に持たせる必要がある．このため，語は意味的に順序づけられたスケルトン構造で表わし，意味変換のための知識ユニットの集合，自然語表現とシステム固有のアトム表現との対応関係を表わすアトム定義形式（単一アトムからなる知識ユニット）の集合を用意し，これらの知識ユニット内の各アトムは，それがもつ変数の変域を表わすスケルトン構造内の各ノードに結合され，双方向のアクセス・パスが定義される．そして，処理メカ

ニズムは，このような知識ベースの中から必要な知識の探索が効率よく行われる推論方式とした[*1*]．

また，物理，化学などの自然科学分野における各種実験データや，図書管理のための諸データ，その他，企業における在庫管理，人事管理のための各種データなどはデータ・ファイルとして一般に記録されるが，実験システムでは，これらの各種データを関係型のデータ・ファイルとしてデータベースに組み込む機能をもち，必要に応じて推論・探索を行ないこれを利用できる形式をとっている．図2にシステムの情報構造を示す．

2.2　推論機構

図3に推論機構の概略を示す．まず，推論は拡張された述語の AND・OR　表現内の間接

構成要素		形　　式	記憶媒体
知識ベース	辞　　書	木構造 { 文字　ノード / データノード(品詞, 骨組構造へのポインタ, その他)	ディスク / コア
	スケルトン構　造	グラフ構造(D型ノード, I型ノード, P型ノード, E型ノード)	コア / ディスク
	アトム定義形式集合	拡張一階述語単一論理式(手続き, 非手続きノード)	ディスク
	知　識ユニット	拡張一階述語複合論理式(アンド, オア構造)	ディスク
データベース	データファイル	関係型データファイル	ディスク

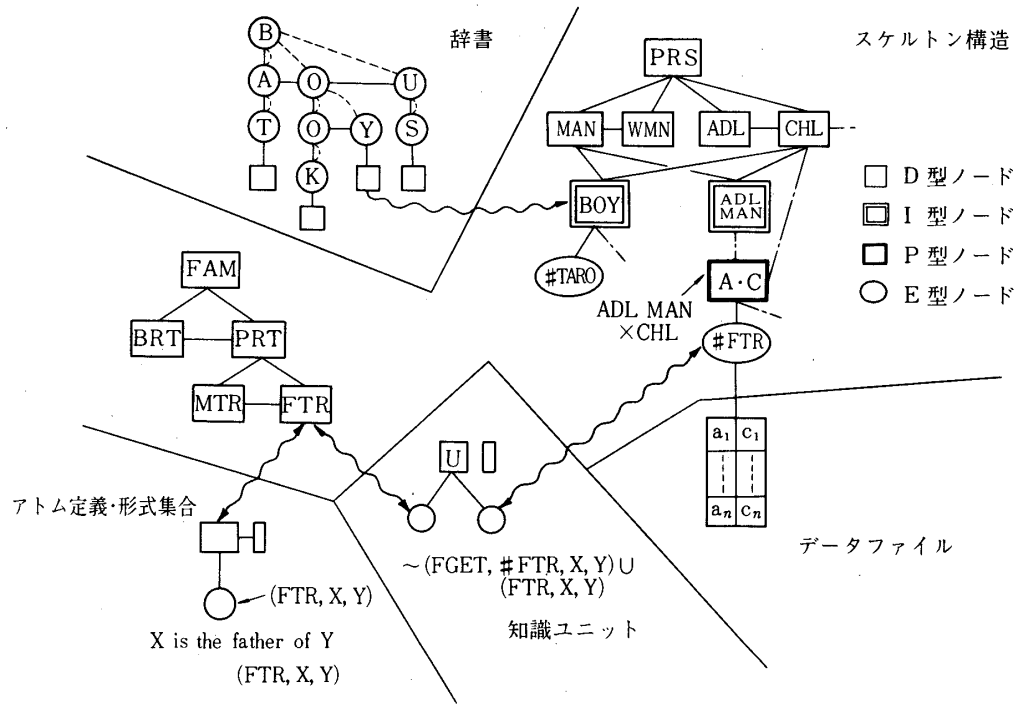


図2　情　報　構　造

的証明可能なノード（LCC）を求め〔B〕，これを含意する候補リテラル（LCP）のセット（SLCP）を知識ベース内に見出す〔D〕．ここで，前もってCのプリテストで置換の無限ループに入るか，LCC は既的なノードかのチェックが与えられた置換木の内部のみで行われる．次に，SLCPが求まると，この中から一つLCPを抽出し〔E〕，次の含意判定のところで LCC を LCPが含意するかの判定がなされる．この含意判定はLCP，LCC の対応する変数の定義域の包含関係によってチェックされ，すべての対応する変数の組が定められた条件を満足したとき次の代入操作が行われる．また，対応する組がともにネストされたAND・OR 表現のとき，推論は再帰的に行われる．このようにして，すべての LCC が処理され，最後に直接的証明可能な手続きノードの処理過程（DPFプロセス）に入る［I］．

このような推論は，最初，質問の否定形を作成して推論を開始し（～Cモード），答が得られなかったとき，次は与えられた質問形そのものから推論をやり直す（Cモード）．両モ

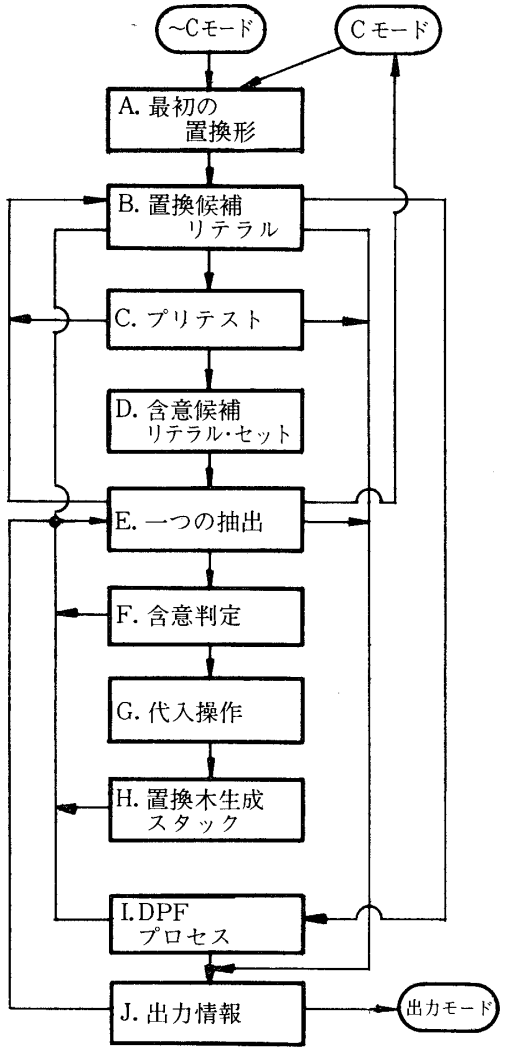


図3　推論機構図

ードを実行しても答が得られなかったとき，システムは“I DON'T KNOW”を出力する．

推論の流れは大略以上のとおりであるが，知識の探索および含意判定は頻繁に行われ推論過程の中心部となっている．そこでこれらについて少し具体的に記しておく．

2.2.1　知識の探索

知識の探索の対象となるものは置換候補リテラル（LCC）と同じデータをもつリテラル（含意候補リテラル—LCP）を含む知識ユニットである．ここで，LCCは置換木(中間手続き表現）内の間接的証明可能なノード（NTA）のうちで，それが少くとも一つの∃限量化されている束縛変数あるいは定数をもつものをいう．また，リテラルの型データとはリテラルを変数（定数）の組とみたときのその組の大きさとそのリテラル内の∃変数または定数がリテラル内の何番目のものかをペアで表わしたものである．

スケルトン構造の各ノードにはそれを定義域とする変数をもつ知識ユニット内のリテラルへのポインタおよびそのリテラル内の何番目の変数の定義域として使われているかの情報がテーブル形式（EFT）で記憶されているので，知識の探索はこれを利用して行われる．すなわち，

- LCCの型データLPAT（I）と一致するEFT内のリテラルを，LCCの変数V(I)の定義域を表わすスケルトン構造内のノードおよびその上位ノードのEFTを参照してすべてとり出す．
- 上記をLPAT全体についてくり返し，それらの共通部分のみを残す．

によって含意候補リテラル・セット（SLCP）を求める．たとえば，

LCC：$(\forall x)(\exists y)(\exists z)(f, x, a, y, z)$

を考えると，f，a，y，z が∃変数もしくは定数である．これらの定義域が図4のようなスケルトン構造内のノードであるとすると，・ノードのEFTが SLCP を求める対象となる．

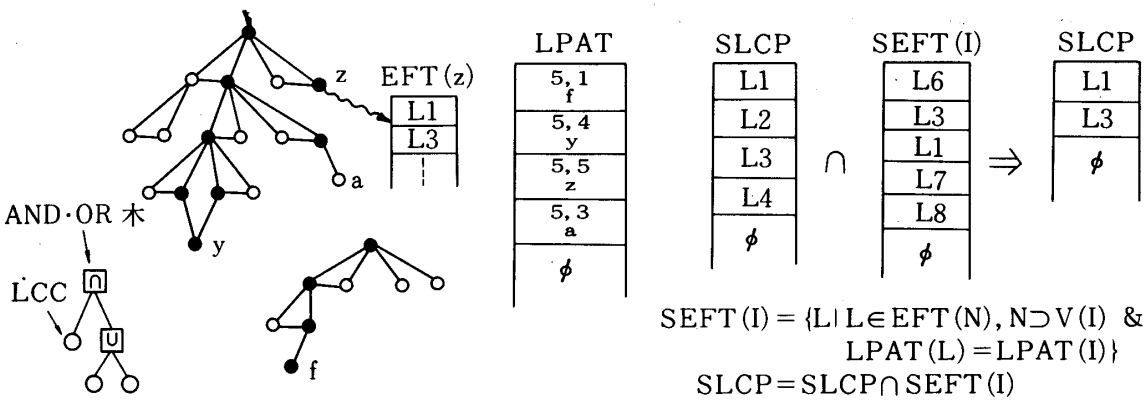


図4　知識の探索

2.2.2 含意判定

含意候補リテラル（LCP）と置換候補リテラル（LCC）との含意関係を調べる.テスト方法は，LCPおよびLCCの対応する変数の組（マッチング・ペア）に対し，その限量詞の組合わせ，限量詞の順序情報（JINF）および定義域の包含関係について調べる．含意成立の条件を表1に示す[1].

JINFテスト

JINFテストは X_p（LCP変数）が∃変数で，対応する X_c^-（LCC変数）が∃変数のばあいのみ行われる．そして，二つのリテラルが unify 可能かどうかを調べるものである.

一般に，述語論理において，

（1） $(\exists x)(\forall y)P(x, y)$

（2） $(\forall y)(\exists x)P(x, y)$

の2つの論理式があったばあい，（1）から（2）を証明できるが，（2）から（1）は証明できない．これは skölem 関数を用いて，（2）と～（1）が unify 可能かどうかを見ればよいが，（2）は $P(f(y), y)$ で～（1）は $P(x, g(x))$ となり，most significant unifierが存在しない[6]．つまり，

LCP： $(\forall y)(\exists x)P(x, y)$

LCC： $\sim(\exists x)(\forall y)P(x, y)=(\forall x)(\exists y)\sim P(x, y)$

としたとき，限量詞の組合わせは x，y について共に表1を満足するが，含意は成立しない．JINF テストはこれをマトリックス形式で表わされた限量詞順序情報を用いて行う．その方法は，LCP およびLCCの JINF マトリックスの行と列をそろえ，

$JINF_{LCP}(i, j) = JINF_{LCC}(j, i) = 1$

が成り立っているかをみればよい．ここで，$JINF(i, j)=1$ はフォーミュラの限量化部においてi番目の変数が∃変数でそれに先行する j 番目の変数が∀変数であることを示す.

定義域の包含関係テスト

表2の定義域の包含関係，$X_P \supset X_C^-$（$X_P \subset X_C^-$）あるいは $X_P \cap X_C^- \neq \phi$ の条件テストは

表1. 含意成立条件

Q_P	Q_C^-	Q_R	X_R	CONDITON
∀	∀	∀	$X_P \cap X_C^-$	$X_P \cap X_C^- \neq \phi$
∀	∃	∃	X_C^-	$X_P \supset X_C^-$
∃	∀	∃	X_P	$X_P \subset X_C^-$, JINF
—	∀	∀	X_C^-	—
—	∃	∃	X_C^-	—
∀	—	∀	X_P	—
∃	—	∃	X_P	—

LCP: $Q_P(X_{P1}\ X_{P2}\ X_{P3})$
↕ ↕ ↕
LCP: $Q_C^-(X_{C1}^-\ X_{C2}^-\ X_{C3}^-)$

Q_R, X_R は unify されたときの情報を表す.

スケルトン構造内の X_P および $X_{\bar{C}}$ のノードの包含関係を調べることによってなされる．

スケルトン構造内の各ノードは集合論的に順序づけられており，下位集合は上位集合の互いに交わらない部分集合に分けられたものか，2つの上位集合の共通部分として定義されたものを基本としている．そこで，この性質を利用した次のテストが行われる．

（1）　$X_P \subset X_{\bar{C}}$ テスト

X_P を含む上位ノードのセットを SXN とし（図5－(a)の・ノード），

（a）　$X_{\bar{C}}$ が D 型ノード(図5－(a)の $X_{\bar{C}2}$ ノード）のとき，

$X_{\bar{C}} \in S \times N$ なら　$X_P \subset X_{\bar{C}}$

（b）　$X_{\bar{C}}$ が　I 型ノード(図5－(a)の $X_{\bar{C}}$ ノード）のとき，

$X_{\bar{C}}$ の直上親集合 $\subset SXN$　なら　$X_P \subset X_{\bar{C}}$

が成り立つ．$X_P \supset X_{\bar{C}}$ のテストは，上記を X_P および $X_{\bar{C}}$ を入れかえて考えればよい．図5－(a)では $X_P \not\subset X_{\bar{C}}$，$X_P \subset X_{\bar{C}2}$ となる．

（2）　$X_P \cap X_{\bar{C}} = \phi$ テスト

X_P または $X_{\bar{C}}$ のどちらかのノードについて，それを含む上位ノードのセットを SXN とし（図5－(b)の・ノード），他方のノードについてはそれと交わらない上位集合のセットを SYN としたとき（図5－(b)の⊗ノード）$SXN \cap SYN \neq \phi$ なら，$X_P \cap X_{\bar{C}} = \phi$

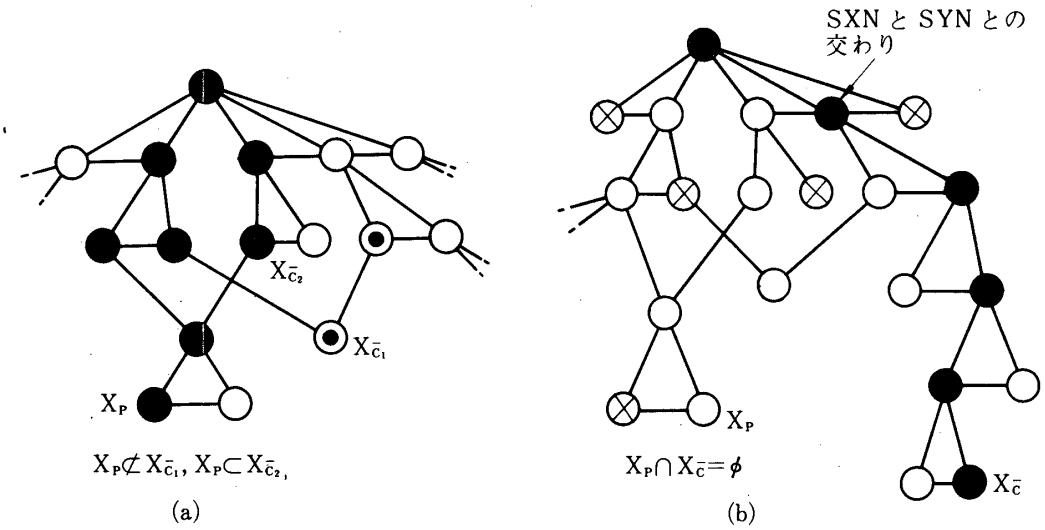


図5　定義域の包含関係

が成り立つ，図5－(b)では，$X_P \cap \bar{X_C} = \phi$ である．

3. 入力言語

入力言語のもつべき機能は，知識ベース内のスケルトン表現および知識ユニット表現，実験用データベースの作成，システムへの質問表現が可能でなければならない．特に，知識ユニットは拡張された述語で表わされたアトムのAND・OR表現であり，通常の一階述語におけるアトムと違って，

（1） 変数の変域を陽に含む
（2） 述語記号自身が変数として扱い得る
（3） 再帰的表現が可能
（4） 手続き型アトム（関数）を陽に含む

である．このような特徴をもった知識ユニットの形で一般的な事実関係，個別的事実関係，自然語表現とのインターフエース（自然語表現とシステム固有のアトム表現との対応関係）およびデータ・ファイルのセマンティックスが表現される．そこで，入力言語のシンタックスは

$$(V_0, V_1, V_2, \cdots, V_r)$$

なる組表現を基礎とし，そのネスト表現が可能とした．ここで，V_0 はシステム・コマンド名か述語名またはそれを変域とする変数，$V_i (i \neq 0)$ は実体語，文字列，数値，スケルトン構造内のあるノードを変域とする変数および任意の組表現を基本とする．システム・コマンドには以下のものがある．

（1） SK　　スケルトン構造の一般語の定義
（2） EL　　スケルトン構造の固有語（要素）の定義
（3） AND　論理積（Λ）
（4） OR　　論理和（V）
（5） TRNS　自然語表現とシステム固有のアトム表現との対応づけ
（6） FCRE　関係データ・ファイルの定義
（7） FDAT　関係データのそう入

知識ユニットは，AND・ORコマンドと接頭部表現とを用いて表現される．

3.1 入力言語の使用例

1)．スケルトン・ノードの定義

スケルトン・ノードは，実体語の概念集合あるいは関係語の概念集合およびそれらの要素(固有語)を表わし，その意味的な包含関係により順序づけて構造化されるものと，これらの積集合を表わすものとがある．これらをD型，I型，P型，E型の四つの型のノードで表現する．これらのノードの構造的表現と入力表現との対応関係を示せば図6のとおりになる．

すなわち，図ではD型ノードとして男性（ML）および女性（FML）が人間の集合（PS）を性別で分割して定義したものであり（図のA2は分割クラスを表わすが具体的に性別を意味するものと定義する必要はなく，単なる識別子と考えてよい），ML∩FML＝ϕ なる

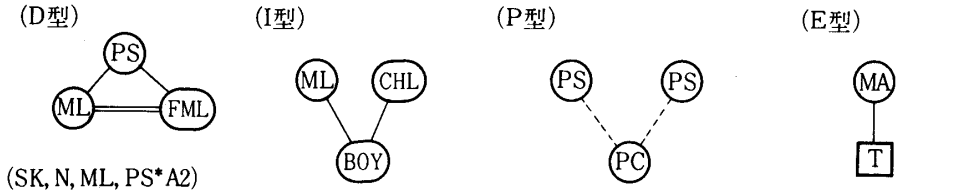


図6　スケルトン・ノードの定義

関係を示す．また，I型ノードとして少年（BOY）をMLとCHL（子供）の共通部分として定義し，PCノードはPSとPSとの積集合として定義されている．また，MAの要素として#TARŌが定義されている．Nは品詞情報を表わすことが細かなことは省略する．

2）．知識ユニットの定義

一般に知識ユニットは，"If A，then B"なる表現を表わし，これは論理式で書けば，～AUBと等しい．すなわち，EFIに関しては「もしZがXの親でかつZがYの親でありXとYが等しくない」ならば「XとYは兄弟である」を表現している．

EFI ：〔AX/PS〕〔AY/PS〕〔AZ/PS〕(ŌR, −(PRT, $Z, $X), −(PRT, $Z, $Y) −′(NONEQ, $X′, $Y′), (BOS, $X′, $Y′)).

ここで，〔AX/PS〕〔AY/PS〕〔AZ/PS〕はEFIの接頭部（限量化部）で，〔AX/PS〕は，「すべての人Xに対して」を意味し，PS（人）はXの変域を表わす，Aの代りにEを使えば「ある人Xに対して」となる．また，アトム表現の中で$つきの英文字は変数を表わし，手続き型アトムは左かっこの前にアポストロフをつける．また，アトム変数の中で交換可能（順序不定）のものは変数名の右肩にアポストロフをつける．

3）．データベースの定義

データベースの定義は，まず，関係データ名（積集合要素名）とその軸要素を< >で与え，次にFDATコマンドで関係データのそう入を行なう（図7参照）．図のTの部分は入力ディバイスを指定する．

4）．アトム定義形式の定義

自然語表現とシステム固有のアトム表現との対応づけを行なうもので，自然語入力および出力時に参照される．自然語による入出力は，現在検討中であり具体的な記述形式はここでは省略したい．

3.2　入力言語の解析

この入力言語の解析は，図8のようにOスタックとAスタックの2つのスタックを使って左から右へスキャンして行われる．Oスタックには，v_0 および符号，アトムの型（手続き型，非手続き型の別）およびAスタックへのポインタがスタックされ，Aスタックには，v_i（$i \neq o$）の入力文内の文字列へのポインタがスタックされる．そして，右閉じカッコに出合ったとき，Oスタックから v_0 をアンスタックし，v_0 のタイプによって定められた処理を v_i ストリングに施す．スタックが空になったら，エンドマークの種類により作業場所内に作成さ

(FCRE, // PC, <PS, PS>).
(FDAT, // PC, T).
<#YOSHIO, #ICHIRO>
<#YOSHIO, #JIRO>
<#YOSHIO, #SABURO>
<#ICHIRO, #AKIO>
<#JIRO, // TARO>

PC

PS	PS
# Y	# I
# Y	# J
# Y	# S
# I	# A
# J	# T

図7　データベースの定義

れた処理結果をシステム内に組み込むかあるいは質問とみなして以後の処理を続行する.

4. ファイル操作プログラムの生成

推論過程で中間非手続き表現内の間接的証明可能なノードはすべてそれと等価な意味をもつ知識ユニットで置き換えられていき,最後には直接的証明可能なノードである手続き型ノードのみよりなる中間手続き表現に変換される. ファイル操作プログラムの生成は, この中間手続き表現を入力データとし, これをもとにファイル操作型プログラムを生成する. これは, 図3における DPF プロセスで行われる.

実験システムの中間手続き表現は図9のような形式になっている. すなわち, 真または偽または不明と評価されたノード（図の・, ■ノード）を AND・OR 木から削除せず, どういう知識ユニットが推論に使用されたかがわかるようにしてある. 本来, 木の評価のためのみには, このような評価済のノードを残しておくことは得策ではないが, ここで, こうした表現形式にした理由を少しのべておこう.

実験システムは, 現在, yes, no 型および value を求める what 型の質問に答えることができるが, さらに, 計算機のある質問に関する知識レベルを問うばあいに有効であろうと思われるからである. すなわち, 計算機から "I DON'T KNOW" が返えされたとき, どこが解らないのかとか, また否定的な答が返えされたとき, どこが矛盾しているのかをそのときの木の状態から答を導き出すようなシステムに拡張できるようにしておくためである.

4.1 プログラム形式

実験システムが内部にもつデータベースは実験用の関係データベースであり, これに対するファイル操作プログラムの形式はコマンド形式で次のような一般形をなす列からなる.

(cmd arg1 arg2……argk wrki)

ここで, cmd はコマンド名, argi はそのアーギュメント, wrki は演算結果の格納ワークエリヤ名である. このワークエリヤ名はシステムが自動的に割り当てる.

コマンドには, ファイル型, マッピング型, ディシジョン型の三つのタイプのものがある. フィイル型コマンドには, FGET, MULT, EXPD, AND, OR や PRJ, DIV などの関係

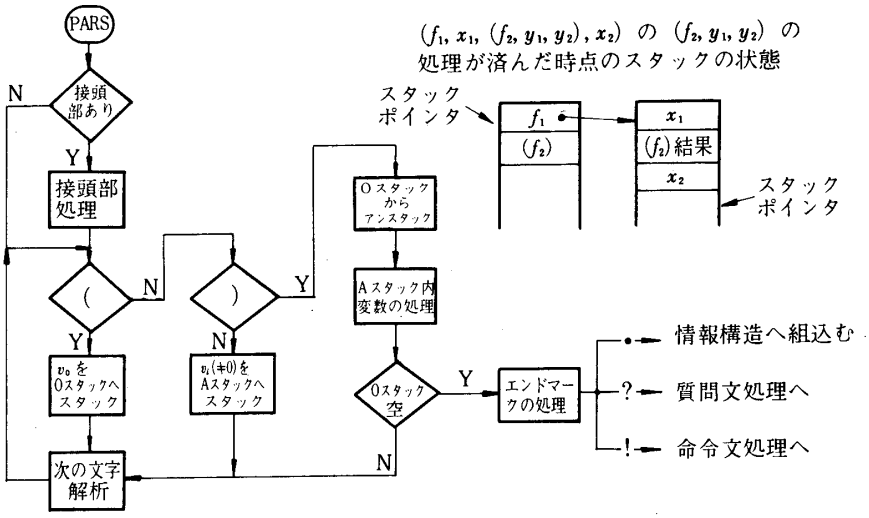


図8　入力言語の解析

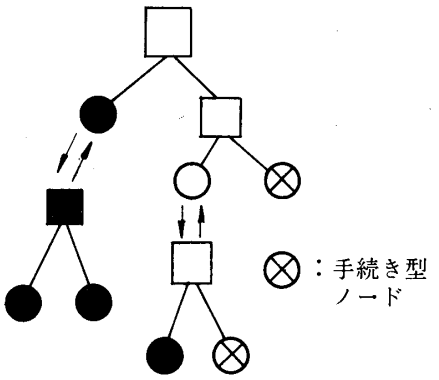


図9　中間手続き表現

代数演算を行うものがあり，マッピング型コマンドには，たとえば，（＃DIV Z X Y W)：ワークファイルWにある関係データのX列をY列で割りその結果をZ列に格納する，のような一般の代数関数などがある．また，ディシジョン型コマンドは，たとえば，（＃GT X Y W)：ワークファイルWにある関係データのX列の値がY列の値よりも大きなレコードを選択しその結果をWワークに格納する，のような特定の条件をみたすレコードをとり出すコマンドである．

以下，これらのコマンドからなるファイル操作プログラムの生成とプログラムの最適化についてのべる．

4.2　コマンド列生成

アクセス・コマンド列の生成は，まず，中間手続き表現内の部分木を depth first で抽出し，抽出した部分木の中の手続きノードをファイル型，ディシジョン型，マッピング型に分類する．そして FGET（ファイル・フエッチ・ノード）のセットとそれらの変数セット（**X**）を求め，FGET コマンドの生成をする．次に，このフエッチされるファイルは各々の共通ドメインをもつように normalize され，部分木の結合子の種類によって AND/OR コマンドを生成しファイル・マージを行う．そのあと，ディシジョン型ノードを調べ，マージされたファイルのドメイン・セットすなわち変数セット**X**の中にその変数がすべて含まれているものを取り出し，そのディシジョン型コマンドを生成する．ただし，マージされる前に適用できる FGET ノードがあればその時点で生成する．マッピング型ノードについては，そのうち，変数セット**X**にその定義域がすべて含まれているものを取り出し，その値域セットと**X**とを併合して**X**とし，マージされたファイルを column expand する．そしてそのあとマッピング型コマンドを生成する．以上をくり返えして部分木内のノードの処理を済ませ，すべての部分木の処理が終了したら次に接頭部の処理に入る．

接頭部には，AND・OR 木内に現われる変数のドメインおよびその限量化およびその部分順序情報などが格納されており，これらを関係代数演算などを表わすコマンドに変換する．すなわち，∃限量化された変数および定数は PRJ（projection）コマンドに，∀限量化された変数は DIV（division）コマンドに，そして値要求変数については OUT（output）コマンドが生成される．これらのコマンドの生成順序は，接頭部内変数の順序情報に従って後ろから前に向って行ない，コマンド列の最後に OUT コマンドを入れる．

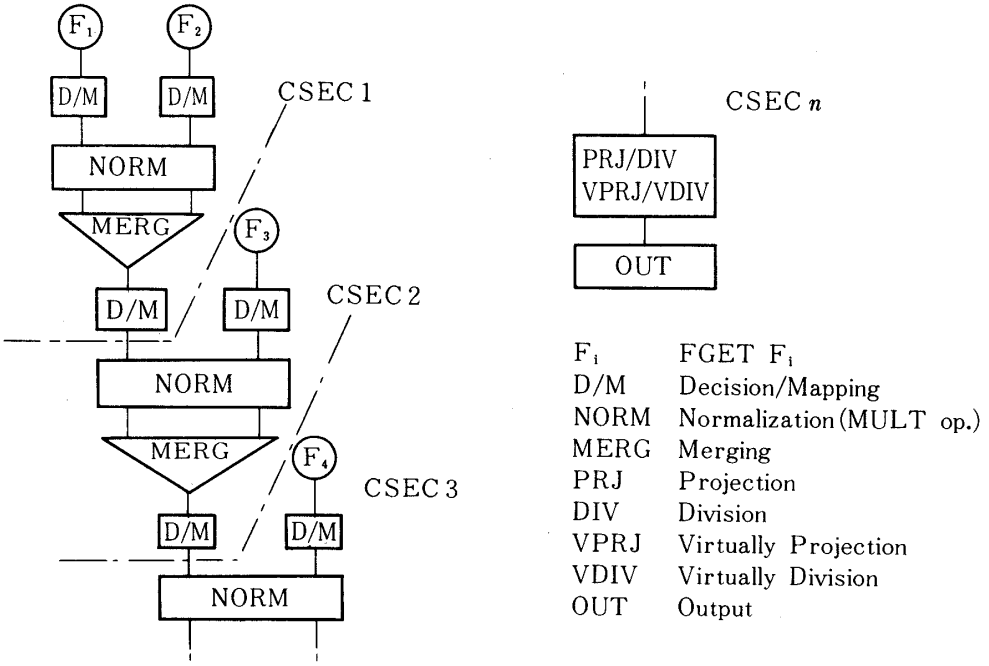


図10　コマンド列の構成

図10は，このような方法で生成されるコマンド列の構成を示したものである．図において，実行の順序を示すと，まず，CSECI（コマンド・セクション1）で F_1 および F_2 の2つのファイルがFGETされ，それぞれに適用可能なディシジョン型およびマッピング型のコマンド（D/Mコマンド）があればそれらが実行される．そしてその結果をノーマライズしたあと，ファイル・マージ（AND/ORマージ）を行ない，その結果に対して適用可能なD/Mコマンドがあればそれらが実行される．CSEC2は，CSEC1で得られたファイルと F_3 ファイルに対して同様なことが行われ，以下，同様にCSEC3，……CSECn と続き，最後にPRJ/DIV/VPRJ/VDIV のコマンド列が続きコマンド列の最後はOUTコマンドとなりこれが実行される．

4.3　コマンド列の最適化

4.2で生成されたコマンド列では，ファイル・マージのためのノーマライゼーション（MULTオペレーション）が数多く行われるが，この操作を行うと実行時のファイル・サイズが途端に大きくなる．そこで，この操作をできるだけ後に行なうようにしたり，あるドメインについてはMULT操作を行わず．AND/OR 操作する前に PRJ（projeetion）またはDIV（division）操作を施してもよいばあいはこれを先に行ない，実行時における効率のよいプログラムとする．このための最適化ルール[1]を用意し，これに従って最適化を行う（表2に最適化ルールを示す）．

適用可能な最適化ルールは，AND/OR 操作されるファイルのドメイン・セット内の各ドメインのタイプを調べて選択する．このドメイン・タイプには，抹消可能MULT アーギュメントとなっている，限量化の種類（∀/∃）および定数の別があり，抹消不可のドメイン

表2.　最適化ルール

E	M_1	M_2	処　理　内　容
0	0	0	1），2)チエック
0	0	1	} 3），4），5），6）チエック
0	1	0	
0	1	1	MULT コマンド削除
1	0	0	1），2）チエック，X^*処理
1	0	1	} 3），4），5），6）チエック，X^*処理
1	1	0	
1	1	1	7），8)適用

E：erazable/nonerazable（0：erazable）
M：MULT argument/not MULT arg.（1：MULT arg.）

1). $DIV(AND(F_1, F_2), X) = AND(DIV(F_1, X), DIV(F_2, X))$
2). $PRJ(OR(F_1, F_2), X) = OR(PRJ(F_1, X), PRJ(F_2, X))$
3). $DIV(AND(F_1, MULT(X, F_2)), X) = AND(DIV(F_1, X), F_2)$
4). $PRJ(AND(F_1, MULT(X, F_2)), X) = AND(PRJ(F_1, X), F_2)$
5). $DIV(OR(F_1, MULT(X, F_2)), X) = OR(DIV(F_1, X), F_2)$
6). $PRJ(OR(F_1, MULT(X, F_2)), X) = OR(PRJ(F_1, X), F_2)$
7). $AND(MULT(X, F_1), MULT(X, F_2)) = MULT(X, AND(F_1, F_2))$
8). $OR(MULT(X, F_1), MULT(X, F_2)) = MULT(X, OR(F_1, F_2))$

はそれがマッピング型またはディシジョン型のアーギュメントとなっているとき，および質問の値要求ドメインとなっているときである．これらの抹消不可のドメインは，最適化プロセスの過程で仮りにPRJ（VPRJと呼ぶ）またはDIV（VDIVと呼ぶ）されることを表示し，そのあとに続くAND/ORコマンドの直接の対象ドメインでないことを示す(図のX*処理)．これらのドメインは必要でなくなった時点で抹消される．

各ドメインの処理順序は，4.2で生成されたPRJ/DIV/VPRJ/VDIVの順序に従い，AND/ORセクションからなるコマンド列を逆にたどって各ドメインの最適化処理が行われるよう最適化する．

5. 実験例

実際に行われた実験例を3つ示す．これらに関する計算機への入力例および生成されたプログラムの例をまとめて巻末に掲げてあるので参照されたい．

例1. 物理の問題

物理に関する簡単な例として，関係データベースに数種の固体の質量と体積がファイル化されており（それぞれ，#WTおよび#VOLファイルとする），また，別に数種の液体の比重がファイル化（#SWファイルとする）されているものとする．このとき，「ファイル化されているすべての液体に浮く固体をデータ・ベースから求めよ」という問題を考えてみよう．

計算機がこの質問に答えられるためには「浮く」ということばの意味を理解できなければならない．知識ベースに，このようなことばの意味関係やデータベースとの関係が知識ユニットとして予め与えられているが，実験システムでは，これらの知識ユニットを参照してことばの意味の理解をし，データベース・アクセスのためのプログラムを自動的に生成することができる．図11は，この例に必要な知識ユニットのみを示したもので，実際の表現は巻末のEF5～EF9でなされる．

上記の質問がシステムに与えられると，システムは推論過程でP1～P5の知識ユニットを探索し，図12のAND・OR木（中間手続き表現）を生成する．このAND・OR木には，#WT，#VOLおよび#SWのファイル・フエッチ型の手続きノード，#DIV（divide）というマッピング型手続きノード，それに#GT（greater than)というディシジョン型手続きノードがある．そして，このAND・OR木はDPFプロセスによって巻末の図B－(g)，(h)のファイル操作プログラムに変換される．

図B－(h)のプログラムに表われるコマンドの意味は次のとおりである．

〔1〕 PO×RNUMで定義された#WTファイルをWORK1へフエッチし，そのワーク・ファイルのドメイン名はPO1およびRNUM9である．

〔2〕 WORK1内のコラム・データRNUM9を仮りにPRJしたファイルをWORK1ファイルとする．

〔5〕 WORK1ファイルとWORK2ファイルをANDマージし，WORK3ファイルとする．

〔6〕 WORK3ファイルのファイル・ドメインとしてRNUM4を付加する(コラム付加)．

〔7〕 WORK3ファイルのRNUM9，RNUM10のコラム・データについてRNUM9／

P1. (∀x/PO)(∀y/LIQ)(∀r/RNUM)(∀s/RNUM)
〔(SW, x, r)∩(SW, y, s)∩(GT, s, r)⇒(FLT, x, y)〕
物体xの比重がrで液体yの比重がsでsがrよりも大きければxはyに浮ぶ.

P2. (∀x/PO)(∀u/RNUM)(∀v/RNUM)(∀w/RNUM)
〔(WT, x, u)∩(VOL, x, v)∩(DIV, w, u, v)⇒(SW, x, w)〕
物体xの重さがuで体積がvであり,uをvで割ったものをwとすればwは物体xの比重である.

P3. (∀x/PO)(∀r/RNUM)
〔(FGET, #WT, x, r)⇒(WT, x, r)〕

P4. (∀x/PO)(∀r/RNUM)
〔(FGET, #VOL, x, r)⇒(VOL, x, r)〕

P5. (∀x/LIQ)(∀r/RNUM)
〔(FGET, #SW, x, r)⇒(SW, x, r)〕

P3. P4. P5 はデータベースとのリンケージを表わし，データベースアクセスを定義する.

C. (∃x/PO)(∀y/LIQ)(FLT, x, y)？
すべての液体に浮く物体は何か？

図11　例1に関する知識ユニット

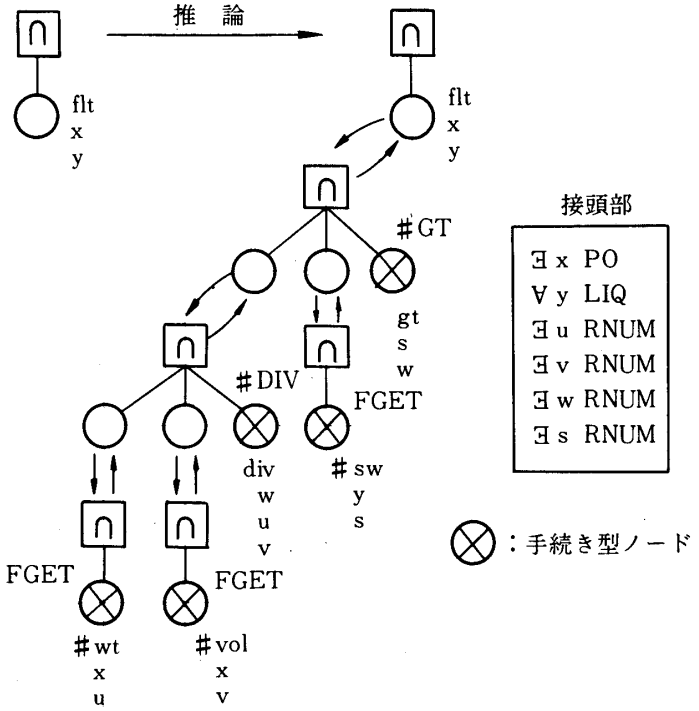


図12　例1の中間手続き表現

RNUM10 を行い，その結果を RNUM4 コラム・データとする．

〔8〕 WORK3 ファイルのコラム・データ RNUM9 を PRJ する．

〔13〕 WORK1 ファイルをノーマライズする．すなわち，WORK1 ファイルのタプル・データと PO1 データとのデカルト積をつくる．

〔16〕 WORK2 ファイルの RNUM5 および RNUM4 のコラム・データを比較し前者が後者よりも大きいレコード（タプル）を抜き出し，WORK2 ファイルとする．

〔21〕 WORK2 ファイルをテレタイプに出力する．

例2. 家族関係に関する問題

データベースには，親子関係のみを表わす1つのファイル（＃PCファイル）しかないとき，これから，兄弟，おじ，おい，いとこの関係などを導く．この例は，とり扱うファイルが1つで推論によっていろいろな関係を導く例である．巻末の図B－(a)～(f)に生成プログラムの例を示す．

例3. 所属関係に関する問題

データベースには，人とその所属するクラブ名の関係を表わす関係データ（＃PSCIC），部はどのクラブからなっているかを示す構成表（要素テーブル），それに，人の身長，体重を示す＃PSATTR データがあるものとする．このとき，ある人があるクラブに属しているとき，その人はどの部に属しているかとか，ある部に属している人の体重と身長の割り合いを求めるとかの質問ができる．巻末の図B－(k),(l) に生成プログラムの例を示す．

6. おわりに

実験システムの概要とこれによる各種知識表現のしかたおよびデータベース・アクセスのためのプログラムの自動生成およびその実験例について述べた．ここに記した例だけに関していえば，質問の入力が終ってから高々1秒以内の時間でプログラムの生成がなされた．この生成時間は，ディスク・アクセスの回数，ディスクI／Oバッファの大きさ，推論の深さに依存して主に変化する．

システムへの入力は，目下，システム固有の形式言語を用いて行われているが，自然語によるシステムとのコミュニケーションを可能とする問題に関しては，現在，進行中であり別の機会としたい．自然語理解の能力をシステムがもつことになれば，一般の casual user が簡単な自然語入力によって，システムの詳しい内容を知らなくても，各種データベースにアクセスできることになろう．

なお，本報告によるシステムとは別個に，大学院生，宇田川佳久君によってインプリメントされたシステムがあり，別に研究が進められている．

1979年12月21日 航空力学部

参 考 文 献

[1] 大須賀節雄，「知的マン・マシン・システム」，本特集号.

[2] S. Ohsuga, "Semantic Information Processing in Man-Machine Systems", Proc. 1977 IEEE Conf. on Decision & Control, pp. 1351−1358.

[3] 宇田川佳久，大須賀節雄「知識システムの設計問題への応用について」，情報処理学会第20回全国大会予稿集.

[4] C.J. Date, "An Introduction to Database Systems", Addison-Wesley, 1975.

[5] 大須賀節雄，山内平行，「推論能力を備えた情報検索方式について」，情報処理，Vol. 18, No. 8, pp. 789〜798 (1977).

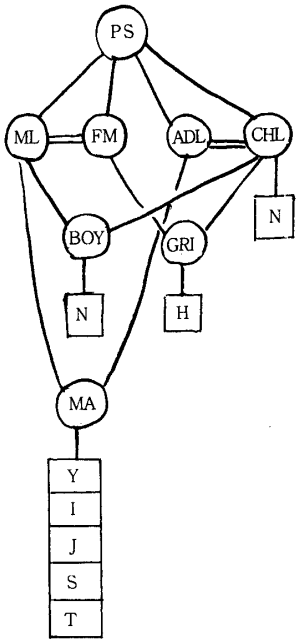
[6] C.L. Chang & R.C.T. Lee, "Symbolic Logic and Mechanical Theorem Proving," Academic Press, 1973.

[7] Herve Gallaire & Jack Minker (edited), "Logic and Databases" Plenum Press, New York, 1978.

付 録

〔A〕知識ベース，データベースの定義

```
***SKELETON STRUCTURE***
SK1:    UNI      (SK,N,UNI).
SK2:    PS       (SK,N,PS,UNI*A1).
SK3:    FAM      (SK,N,FAM,UNI*B1).
SK4:    DPF      (SK,N,DPF,UNI*C1).
SK5:    ML       (SK,N,ML,PS*A2).
SK6:    FML      (SK,N,FML,PS*A2).
SK7:    ADL      (SK,N,ADL,PS*B2).
SK8:    CHL      (SK,N,CHL,PS*B2).
SK9:    BOY      (SK,N,BOY,ML&CHL).
SK10:   GRL      (SK,N,GRL,FML&CHL).
SK11:   MA       (SK,N,MA,ML&ADL).
SK12:   FA       (SK,N,FA,FML&ADL).
SK13:   PRT      (SK,N,PRT,FAM*A2).
SK14:   BOS      (SK,N,BOS,FAM*B2).
SK15:   UOA      (SK,N,UOA,FAM*C2).
SK16:   CSN      (SK,N,CSN,FAM*D2).
SK17:   SUM      (SK,N,SUM,DPF*A2).
SK18:   SUBT     (SK,N,SUBT,DPF*A2).
SK19:   MULT     (SK,N,MULT,DPF*A2).
SK20:   DIV      (SK,N,DIV,DPF*A2).
SK21:   GT       (SK,N,GT,DPF*B2).
SK22:   LT       (SK,N,LT,DPF*B2).
SK23:   GE       (SK,N,GE,DPF*B2).
SK24:   LE       (SK,N,LE,DPF*B2).
SK25:   EQL      (SK,N,EQL,DPF*B2).
SK26:   NONEQ    (SK,N,NONEQ,DPF*B2).
SK27:   FPRO     (SK,N,FPRO,DPF*C2).
SK28:   FJOI     (SK,N,FJOI,DPF*C2).
SK29:   FDIV     (SK,N,FDIV,DPF*C2).
SK30:   FGET     (SK,N,FGET,DPF*C2).
```




```
***ELEMENT NODES***
EL1:    #YOSHIO    (EL,#YOSHIO,MA).
EL2:    #ICHIRO    (EL,#ICHIRO,MA).
EL3:    #JIRO      (EL,#JIRO,MA).
EL4:    #SABURO    (EL,#SABURO,MA).
EL5:    #TARO      (EL,#TARO,MA).
EL6:    #AKIO      (EL,#AKIO,CHL).
EL7:    #NATSUO    (EL,#NATSUO,BOY).
EL8:    #HANAKO    (EL,#HANAKO,GRL).

***FILE DATA***
FL1:    #PC FILE CREATION    (FCRE,#PC,<PS,PS>).
FL2:    #PC FILE DATA          (FDAT,#PC,T).
                               <#YOSHIO,#ICHIRO>
                                <#YOSHIO,#JIRO>
                               <#YOSHIO,#SABURO>
                              <#ICHIRO,#AKIO>
                               <#ICHIRO,#NATSUO>
                               <#JIRO,#TARO>
                               <#SABURO,#HANAKO>
```

#PC

PS	PS
#Y	#I
#Y	#J
#Y	#S
#I	#A
#I	#N
#J	#T
#S	#H

```
***EXTENDED FORMULA***
EF1:    [AX/PS][AY/PS][AZ/PS]
         (OR,-(PRT,$Z,$X),-(PRT,$Z,$Y),-'(NONEQ,$X',$Y'),
         (BOS,$X',$Y')).
EF2:    [AU/PS][AV/PS][AW/PS]
         (OR,-(PRT,$W,$U),-(BOS,$W',$V'),(UOA,$U,$V)).
EF3:    [AR/PS][AV/PS][AT/PS]
         (OR,-(UOA,$R,$T),-(PRT,$T,$V),(CSN,$R',$V')).
EF4:    [AX/PS][AY/PS]
         (OR,-'(FGET,#PC,$X,$Y),(PRT,$X,$Y)).
```

```
***SKELETON STRUCTURE***
SK44:  MFAT       (SK,X,MFAT,DPF*M1)・
SK45:  CARDNO     (SK,X,CARDNO,DPF*M2)・
SK46:  RATIO      (SK,X,RATIO,DPF*M3)・
SK47:  SCIRC      (SK,N,SCIRC,UNI*S1)・
SK48:  ACIRC      (SK,N,ACIRC,SCIRC*A1)・
SK49:  CCIRC      (SK,N,CCIRC,SCIRC*A1)・
SK50:  ECIRC      (SK,N,ECIRC,UNI*S1)・
SK51:  APOPU      (SK,N,APOPU,PS*P1)・
SK52:  ASTD       (SK,N,ASTD,APOPU*A1)・
SK53:  ATCH       (SK,N,ATCH,APOPU*A1)・
SK54:  ASRV       (SK,N,ASRV,APOPU*A1)・
SK55:  PATTR      (SK,N,PATTR,UNI*P2)・
SK56:  PHEI       (SK,N,PHEI,PATTR*A1)・
SK57:  PWEI       (SK,N,PWEI,PATTR*A1)・
SK58:  PSIG       (SK,N,PSIG,PATTR*A1)・
SK59:  HWRATIO    (SK,N,HWRATIO,PATTR*A1)・
SK60:  MEMBER     (SK,N,MEMBER)・
SK61:  BE         (SK,V,BE)・
SK62:  FAT        (SK,M1,FAT)・
SK63:  #MF        (SK,N1,#MF,RNUM*N1)・

***ELEMENT NODES***
EL17:  #ACIRC     (EL,#ACIRC,ACIRC)・
EL18:  #CCIRC     (EL,#CCIRC,CCIRC)・
EL19:  #CIRCA     (EL,#CIRCA,ECIRC)・
EL20:  #CIRCB     (EL,#CIRCB,ECIRC)・
EL21:  #CIRCC     (EL,#CIRCC,ECIRC)・
EL22:  #AAA       (EL,#AAA,ASTD)・
EL23:  #BBB       (EL,#BBB,ATCH)・
EL24:  #CCC       (EL,#CCC,ASRV)・

***FILE DEFINITION***
FL9:   #ACIRC    (FCRE,#ACIRC,*)・
FL10:  #CCIRC     (FCRE,#CCIRC,*)・
FL11:  #PSCIRC    (FCRE,#PSCIRC,<APOPU,ECIRC>)・
FL12:  #PSATTR    (FCRE,#PSATTR,<APOPU,PHEI,PWEI>)・

***KNOWLEDGE UNITS***
EF10:   [AX/APOPU][AY/ECIRC][AZ/SCIRC]
        (OR,-(MEMBER,$X,$Y),-(MEMBER,$Y,$Z),(MEMBER,$X,$Z))・
EF11:   [AX/APOPU][AY/ECIRC]
        (OR,-'(FGET,#PSCIRC,$X,$Y),(MEMBER,$X,$Y))・
EF12:   [AX/ECIRC]
        (OR,-'(FGET,#ACIRC,$X),(MEMBER,$X,#ACIRC))・
EF13:   [AX/ECIRC]
        (OR,-'(FGET,#CCIRC,$X),(MEMBER,$X,#CCIRC))・
EF14:   [AP/APOPU][AH/PHEI][AW/PWEI][AM/RNUM]
        (OR,-(PATTR,$P,$H,$W),-'(MFAT,$M,$H,$W),
       (HWRATIO,$P,$M))・
EF15:   [AP/APOPU][AM/RNUM]
       (OR,-(HWRATIO,$P,$M),-'(GT,$M,#MF),(BE,FAT,$P))・
EF16:   [AP/APOPU][AH/PHEI][AW/PWEI]
        (OR,-'(FGET,#PSATTR,$P,$H,$W),(PATTR,$P,$H,$W))・
```

```
***SKELETON STRUCTURE***
SK31:   P0      (SK,N,P0,UNI*D1).
SK32:   SOL     (SK,N,SOL,P0*D2).
SK33:   LIQ     (SK,N,LIQ,P0*D2).
SK34:   BOL     (SK,N,BOL,P0*D2).
SK35:   NUM     (SK,N,NUM,UNI*E1).
SK36:   INUM    (SK,N,INUM,NUM*E2).
SK37:   RNUM    (SK,N,RNUM,NUM*E2).
SK38:   CNUM    (SK,N,CNUM,NUM*E2).
SK39:   ATRPO   (SK,N,ATRPO,UNI*F1).
SK40:   SW      (SK,N,SW,ATRPO*F2).
SK41:   WT      (SK,N,WT,ATRPO*F2).
SK42:   VOL     (SK,N,VOL,ATRPO*F2).
SK43:   FLT     (SK,N,FLT,ATRPO*F2).

***ELEMENT NODES***
EL8:    #P01    (EL,#P01,P0).
EL9:    #P02    (EL,#P02,P0).
EL10:   #P03    (EL,#P03,P0).
EL11:   #P04    (EL,#P04,P0).
EL12:   #P05    (EL,#P05,P0).
EL13:   #LIQ1   (EL,#LIQ1,LIQ).
EL14:   #LIQ2   (EL,#LIQ2,LIQ).
EL15:   #LIQ3   (EL,#LIQ3,LIQ).
EL16:   #LIQ4   (EL,#LIQ4,LIQ).

***FILE DATA***
FL3:    #WT FILE CREATION   (FCRE,#WT,<P0,RNUM>).
FL4:    #WT FILE DATA       (FDAT,#WT,T).
                    <#P01,8.00>
                     <#P02,3.76>
                     <#P03,7.24>
                     <#P04,2.44>
                    <#P05,3.28>
 FL5:    #VOL FILE CREATION   (FCRE,#VOL,<P0,RNUM>).
FL6:    #VOL FILE DATA       (FDAT,#VOL,T).
                     <#P01,5.00>
                     <#P02,4.82>
                     <#P03,6.35>
                     <#P04,0.92>
                     <#P05,4.80>
 FL7:    #SW FILE CREATION   (FCRE,#SW,<LIQ,RNUM>).
FL8:    #SW FILE DATA       (FDAT,#SW,T).
                    <#LIQ1,0.80>
                     <#LIQ2,1.36>
                    <#LIQ3,1.04>
                     <#LIQ4,1.28>

***EXTENDED FORMULA***
EF5:    [AX/P0][AY/LIQ][AR/RNUM][AS/RNUM]
         '(OR,-(SW,$X,$R),-(SW,$Y,$S),-'(GT,$S,$R),
         (FLT,$X,$Y)).
EF6:    [AX/P0][AU/RNUM][AV/RNUM][AW/RNUM]
         (OR,-(WT,$X,$U),-(VOL,$X,$V),-'(DIV,$W,$U,$V),
         (SW,$X,$W)).
EF7:    [AX/P0][AR/RNUM]
         (OR,-'(FGET,#WT,$X,$R),(WT,$X,$R)).
EF8:    [AX/P0][AR/RNUM]
         (OR,-'(FGET,#VOL,$X,$R),(VOL,$X,$R)).
EF9:    [AX/LIQ][AR/RNUM]
         (OR,-'(FGET,#SW,$X,$R),(SW,$X,$R)).
```

〔B〕ファイル操作プログラムの生成例

```
# 1602G

(a) [EX/PS](CSN,#TARO,$X)?
[1]   (#FGET  #PC  PS7  #TARO  *WRK1)
[2]   (#FGET  #PC  PS12  PS7  *WRK2)
[3]   (MULT  *WRK1  PS12  *WRK1)
[4]   (MULT  *WRK2  #TARO  *WRK2)
[5]   (AND  *WRK1  *WRK2  *WRK3)
[6]   (#FGET  #PC  PS12  PS4  *WRK1)
[7]   (MULT  *WRK1  PS7  *WRK1)
[8]   (MULT  *WRK1  #TARO  *WRK1)
[9]   (MULT  *WRK3  PS4  *WRK3)
[10]  (AND  *WRK1  *WRK3  *WRK2)
[11]  (#NONEQ  PS7  PS4  *WRK2)
[12]  (#FGET  #PC  PS4  PS1  *WRK1)
[13]  (MULT  *WRK1  PS7  *WRK1)
[14]  (MULT  *WRK1  #TARO  *WRK1)
[15]  (MULT  *WRK1  PS12  *WRK1)
[16]  (MULT  *WRK2  PS1  *WRK2)
[17]  (AND  *WRK1  *WRK2  *WRK3)
[18]  (PRJ  *WRK3  #TARO  *WRK3)
[19]  (PRJ  *WRK3  PS7  *WRK3)
[20]  (PRJ  *WRK3  PS12  *WRK3)
[21]  (PRJ  *WRK3  PS4  *WRK3)
[22]  (VPRJ  *WRK3  PS1  *WRK3)
[23]  (OUT  *WRK3  *TTY)
```

```
(b) [EX/PS](CSN,#TARO,$X)?
[1]   (#FGET  #PC  PS7  #TARO  *WRK1)
[2]   (PRJ  *WRK1  #TARO  *WRK1)
[3]   (#FGET  #PC  PS12  PS7  *WRK2)
[4]   (MULT  *WRK1  PS12  *WRK1)
[5]   (AND  *WRK1  *WRK2  *WRK3)
[6]   (#FGET  #PC  PS12  PS4  *WRK1)
[7]   (VPRJ  *WRK3  PS7  *WRK3)
[8]   (MULT  *WRK3  PS4  *WRK3)
[9]   (AND  *WRK1  *WRK3  *WRK2)
[10]  (#NONEQ  PS7  PS4  *WRK2)
[11]  (PRJ  *WRK2  PS7  *WRK2)
[12]  (PRJ  *WRK2  PS12  *WRK2)
[13]  (#FGET  #PC  PS4  PS1  *WRK1)
[14]  (VPRJ  *WRK1  PS1  *WRK1)
[15]  (AND  *WRK1  *WRK2  *WRK3)
[16]  (PRJ  *WRK3  PS4  *WRK3)
[17]  (OUT  *WRK3  *TTY)
```

(a) 最適化前　　(b) 最適化後

```
# 1602G

(c) [EX/PS](UUA,$X,#HANAKO)?
[1]   (#FGET  #PC  PS4  PS1  *WRK1)
[2]   (#FGET  #PC  PS9  PS4  *WRK2)
[3]   (MULT  *WRK1  PS9  *WRK1)
[4]   (MULT  *WRK2  PS1  *WRK2)
[5]   (AND  *WRK1  *WRK2  *WRK3)
[6]   (#FGET  #PC  PS9  #HANAKO  *WRK1)
[7]   (MULT  *WRK1  PS4  *WRK1)
[8]   (MULT  *WRK1  PS1  *WRK1)
[9]   (MULT  *WRK3  #HANAKO  *WRK3)
[10]  (AND  *WRK1  *WRK3  *WRK2)
[11]  (#NONEQ  PS4  #HANAKO  *WRK2)
[12]  (PRJ  *WRK2  #HANAKO  *WRK2)
[13]  (PRJ  *WRK2  PS4  *WRK2)
[14]  (VPRJ  *WRK2  PS1  *WRK2)
[15]  (PRJ  *WRK2  PS9  *WRK2)
[16]  (OUT  *WRK2  *TTY)
```

```
(d) [EX/PS](UUA,$X,#HANAKO)?
[1]   (#FGET  #PC  PS4  PS1  *WRK1)
[2]   (VPRJ  *WRK1  PS1  *WRK1)
[3]   (#FGET  #PC  PS9  PS4  *WRK2)
[4]   (MULT  *WRK1  PS9  *WRK1)
[5]   (AND  *WRK1  *WRK2  *WRK3)
[6]   (#FGET  #PC  PS9  #HANAKO  *WRK1)
[7]   (VPRJ  *WRK1  #HANAKO  *WRK1)
[8]   (VPRJ  *WRK3  PS4  *WRK3)
[9]   (AND  *WRK1  *WRK3  *WRK2)
[10]  (#NONEQ  PS4  #HANAKO  *WRK2)
[11]  (PRJ  *WRK2  #HANAKO  *WRK2)
[12]  (PRJ  *WRK2  PS4  *WRK2)
[13]  (PRJ  *WRK2  PS9  *WRK2)
[14]  (OUT  *WRK2  *TTY)
```

(c) 最適化前　　(d) 最適化後

```
      1602G

(e)  [EX/PS](BOS,#JIRO,$X)?
     [1]   (#FGET  #PC  PS4  #JIRO  *WRK1)
     [2]   (#FGET  #PC  PS4  PS1  *WRK2)
     [3]   (MULT  *WRK1  PS1  *WRK1)
     [4]   (MULT  *WRK2  #JIRO  *WRK2)
     [5]   (AND  *WRK1  *WRK2  *WRK3)
     [6]   (#NONEQ  #JIRO  PS1  *WRK3)
     [7]   (PRJ  *WRK3  #JIRO  *WRK3)
     [8]   (PRJ  *WRK3  PS4  *WRK3)
     [9]   (VPRJ  *WRK3  PS1  *WRK3)
     [10]  (OUT  *WRK3  *TTY)



(f)  [EX/PS](BOS,#JIRO,$X)?
     [1]   (#FGET  #PC  PS4  #JIRO  *WRK1)
     [2]   (VPRJ  *WRK1  #JIRO  *WRK1)
     [3]   (#FGET  #PC  PS4  PS1  *WRK2)
     [4]   (VPRJ  *WRK2  PS1  *WRK2)
     [5]   (AND  *WRK1  *WRK2  *WRK3)
     [6]   (#NONEQ  #JIRO  PS1  *WRK3)
     [7]   (PRJ  *WRK3  #JIRO  *WRK3)
     [8]   (PRJ  *WRK3  PS4  *WRK3)
     [9]   (OUT  *WRK3  *TTY)
```

(e) 最適化前　　(f) 最適化後

```
      # 1602G

(g)  [EX/PO][AY/LIQ](FLT,$X,$Y)?
     [1]   (#FGET  #WT  PO1  RNUM9  *WRK1)
     [2]   (#FGET  #VOL  PO1  RNUM10  *WRK2)
     [3]   (MULT  *WRK1  RNUM10  *WRK1)
     [4]   (MULT  *WRK2  RNUM9  *WRK2)
     [5]   (AND  *WRK1  *WRK2  *WRK3)
     [6]   (EXPD  *WRK3  RNUM4  *WRK3)
     [7]   (#DIV  RNUM4  RNUM9  RNUM10  *WRK3)
     [8]   (#FGET  #SW  LIQ2  RNUM5  *WRK1)
     [9]   (MULT  *WRK1  PO1  *WRK1)
     [10]  (MULT  *WRK1  RNUM9  *WRK1)
     [11]  (MULT  *WRK1  RNUM10  *WRK1)
     [12]  (MULT  *WRK1  RNUM4  *WRK1)
     [13]  (MULT  *WRK3  LIQ2  *WRK3)
     [14]  (MULT  *WRK3  RNUM5  *WRK3)
     [15]  (AND  *WRK1  *WRK3  *WRK2)
     [16]  (#GT  RNUM5  RNUM4  *WRK2)
     [17]  (PRJ  *WRK2  RNUM9  *WRK2)
     [18]  (PRJ  *WRK2  RNUM10  *WRK2)
     [19]  (PRJ  *WRK2  RNUM4  *WRK2)
     [20]  (PRJ  *WRK2  RNUM5  *WRK2)
     [21]  (DIV  *WRK2  LIQ2  *WRK2)
     [22]  (VPRJ  *WRK2  PO1  *WRK2)
     [23]  (OUT  *WRK2  *TTY)
```

(h)
```
[EX/PO][AY/LIQ](FLT,$X,$Y)?
[1]    (#FGET   #WT   PO1   RNUM9   *WRK1)
[2]    (VPRJ   *WRK1   RNUM9   *WRK1)
[3]    (#FGET   #VOL   PO1   RNUM10   *WRK2)
[4]    (VPRJ   *WRK2   RNUM10   *WRK2)
[5]    (AND   *WRK1   *WRK2   *WRK3)
[6]    (EXPD   *WRK3   RNUM4   *WRK3)
[7]    (#DIV   RNUM4   RNUM9   RNUM10   *WRK3)
[8]    (PRJ   *WRK3   RNUM9   *WRK3)
[9]    (PRJ   *WRK3   RNUM10   *WRK3)
[10]   (#FGET   #SW   LIQ2   RNUM5   *WRK1)
[11]   (VPRJ   *WRK1   RNUM5   *WRK1)
[12]   (VPRJ   *WRK3   RNUM4   *WRK3)
[13]   (MULT   *WRK1   PO1   *WRK1)
[14]   (MULT   *WRK3   LIQ2   *WRK3)
[15]   (AND   *WRK1   *WRK3   *WRK2)
[16]   (#GT   RNUM5   RNUM4   *WRK2)
[17]   (PRJ   *WRK2   RNUM4   *WRK2)
[18]   (PRJ   *WRK2   RNUM5   *WRK2)
[19]   (DIV   *WRK2   LIQ2   *WRK2)
[20]   (VPRJ   *WRK2   PO1   *WRK2)
[21]   (OUT   *WRK2   *TTY)
```

(g) 最適化前　　(h) 最適化後

```
# 1602G
```

(i)
```
[EX/RNUM](SW,#PO1,$X)?
[1]    (#FGET   #WT   #PO   RNUM4   *WRK1)
[2]    (#FGET   #VOL   #PO   RNUM5   *WRK2)
[3]    (MULT   *WRK1   RNUM5   *WRK1)
[4]    (MULT   *WRK2   RNUM4   *WRK2)
[5]    (AND   *WRK1   *WRK2   *WRK3)
[6]    (EXPD   *WRK3   RNUM1   *WRK3)
[7]    (#DIV   RNUM1   RNUM4   RNUM5   *WRK3)
[8]    (PRJ   *WRK3   #PO   *WRK3)
[9]    (PRJ   *WRK3   RNUM4   *WRK3)
[10]   (PRJ   *WRK3   RNUM5   *WRK3)
[11]   (VPRJ   *WRK3   RNUM1   *WRK3)
[12]   (OUT   *WRK3   *TTY)
```

(j)
```
[EX/RNUM](SW,#PO1,$X)?
[1]    (#FGET   #WT   #PO   RNUM4   *WRK1)
[2]    (VPRJ   *WRK1   RNUM4   *WRK1)
[3]    (#FGET   #VOL   #PO   RNUM5   *WRK2)
[4]    (VPRJ   *WRK2   RNUM5   *WRK2)
[5]    (AND   *WRK1   *WRK2   *WRK3)
[6]    (EXPD   *WRK3   RNUM1   *WRK3)
[7]    (#DIV   RNUM1   RNUM4   RNUM5   *WRK3)
[8]    (PRJ   *WRK3   #PO   *WRK3)
[9]    (PRJ   *WRK3   RNUM4   *WRK3)
[10]   (PRJ   *WRK3   RNUM5   *WRK3)
[11]   (VPRJ   *WRK3   RNUM1   *WRK3)
[12]   (OUT   *WRK3   *TTY)
```

(i) 最適化前　　(j) 最適化後

```
# 1602G
```

(k)
```
[AX/ASTD][AY/RNUM](AND,(MEMBER,$X,#ACIRC),(HWRATIO,$X,$Y))?
[1]   (#FGET  #PSCIRC  ASTD1  ECIRC6  *WRK1)
[2]   (#FGET  #ACIRC  ECIRC6  *WRK2)
[3]   (MULT  *WRK2  ASTD1  *WRK2)
[4]   (AND  *WRK1  *WRK2  *WRK3)
[5]   (#FGET  #PSATTR  ASTD1  PHEI13  PWEI14  *WRK1)
[6]   (MULT  *WRK1  ECIRC6  *WRK1)
[7]   (MULT  *WRK3  PHEI13  *WRK3)
[8]   (MULT  *WRK3  PWEI14  *WRK3)
[9]   (AND  *WRK1  *WRK3  *WRK2)
[10]  (EXPD  *WRK2  RNUM2  *WRK2)
[11]  (#MFAT  RNUM2  PHEI13  PWEI14  *WRK2)
[12]  (PRJ  *WRK2  ECIRC6  *WRK2)
[13]  (PRJ  *WRK2  PHEI13  *WRK2)
[14]  (PRJ  *WRK2  PWEI14  *WRK2)
[15]  (OUT  *WRK2  *TTY)
```

(l)
```
[AX/ASTD][AY/RNUM](AND,(MEMBER,$X,#ACIRC),(HWRATIO,$X,$Y))?
[1]   (#FGET  #PSCIRC  ASTD1  ECIRC6  *WRK1)
[2]   (#FGET  #ACIRC  ECIRC6  *WRK2)
[3]   (MULT  *WRK2  ASTD1  *WRK2)
[4]   (AND  *WRK1  *WRK2  *WRK3)
[5]   (PRJ  *WRK3  ECIRC6  *WRK3)
[6]   (#FGET  #PSATTR  ASTD1  PHEI13  PWEI14  *WRK1)
[7]   (VPRJ  *WRK1  PHEI13  *WRK1)
[8]   (VPRJ  *WRK1  PWEI14  *WRK1)
[9]   (AND  *WRK1  *WRK3  *WRK2)
[10]  (EXPD  *WRK2  RNUM2  *WRK2)
[11]  (#MFAT  RNUM2  PHEI13  PWEI14  *WRK2)
[12]  (PRJ  *WRK2  PHEI13  *WRK2)
[13]  (PRJ  *WRK2  PWEI14  *WRK2)
[14]  (OUT  *WRK2  *TTY)
```

(k) 最適化前　　(l) 最適化後