フレーム型知識表現における論理型推論メカニズムの検討†

島 健一††

本論文は，フレーム型知識表現と融合した論理型の推論メカニズムPRIMEについて，そのねらいと処理方式を述べる．エキスパートシステムを構築する場合，個々の対象の構造，対象間の階層関係の表現にはフレーム型知識表現が適している．一方，細かな動作を意識せず推論手続きを明確に表現するには論理型表現が向く．フレーム型知識表現と述語論理を組み合わせて知識を表現する方法もあるが，(1)フレーム構造のマッチング，フレーム構造の変更を統一的に扱う簡便な記述，(2)対象の構造に基づいた推論規則の適用範囲の記述，(3)フレーム操作の試行錯誤的な適用の制御，に関しては従来十分な配慮が成されていない．これらの問題に対して，(1)フレーム構造のマッチング，およびフレーム構造の変更を論理の枠組みのなかで統一的に扱う機能，(2)推論規則の適用範囲の限定機能，(3)試行錯誤的操作で矛盾無く，フレーム操作を行うための変更フレームの自動回復機能，を具備するメカニズムを開発することで解決した．これにより，フレーム型知識表現において，推論手続きの記述，制御を容易にできる．また，試行錯誤が多く，探索範囲が膨大となる設計型の問題に対しても，上記の機能を取り入れることが有効であることを示す．

1. まえがき

エキスパートシステムは，特定の問題領域の複雑な問題に対して，専門家の知識を用いて推論を行い，問題解決をはかるシステムである．その構築のためには，知識ベースと推論機構が重要である．

知識ベースは，従来のデータベースとは基本的に異なり，以下の点を考慮する必要がある．

(1) 知識は一般に階層構造を持つと考えられるので，知識ベースの表現においても対象の持つ複雑な構造を階層化できること．また，階層化された知識の修正が動的にでき，知識ベース全体にわたり矛盾なく，その操作は自動的に行われること．

(2) 対象の記述のみならず，付随する手続きも対象と関連するものとして表現でき，その定義，更新操作が容易であること．

知識の表現法としては，従来から各種のものが提案されている[1]．上記の点を満たすものとしてはフレーム理論[3]に基づく表現（フレーム型知識表現）[4)~6)]，オブジェクト指向概念に基づく表現（オブジェクト指向型知識表現）[7]がある．(1)については，階層的なデータ構造による表現，およびインヘリタンス(属性遺伝)機能により実現される．(2)については，手続き付加(attached procedure)，あるいはメソッド(method)により実現される．しかし，オブジェクト指向型知識表現では，階層関係を動的に変えること（クラスの追加，削除などの変更）ができず操作面で問題がある．

フレーム型知識表現を用いて，エキスパートシステムを作成する場合，そこでの推論手続きとしては，(1)対象の特定の構造のパターンをサーチし，それに対して変換，合成などの操作を行うこと，(2)与えられた条件を満足しない場合には，別の候補について再度試行を繰り返すこと，が必要である．従来のフレーム型表現を用いたシステム[4),5)]では，特定の推論機構を持たず，利用者は自ら付加手続きと呼ばれる方法で推論機構を逐一作成する必要があり，非常な負担となっていた．この負担を軽減させるためには，対象の持つ機能，動作などを明確に表現でき，試行錯誤に適したメカニズムを推論機構が基本機能として持つ必要がある．これらの基本機能の表現には述語論理などの論理型表現が向いている．

1階述語論理を基礎にした言語 Prolog[10] は，それ自身で証明過程を用いた推論機構を持ち，統一化(unification)の機能，自動バックトラッキングなどの機能を持つ．しかしながら，Prologだけでは，知識表現の面において不十分であることが指摘されている[12]．これに対し，フレーム型知識表現と述語論理を結合したシステム[6),12),16)~18)]，オブジェクト指向型知識表現にルールシステムを組み込んだシステム[7]，知識表現を Prolog を用いて作成するシステム[11),19)]が報告されている．しかし，それらはいずれも推論機構の観点から見た場合，以下の点に関して問題がある．

(1) フレーム型表現での推論は，頻繁に対象中の特定の構造のパターンに対して変換，合成などの操作を行う．そのため，(a)フレームのデータ構造を容易

† A New Inference Mechanism for Frame-based Knowledge Representation by KENICHI SHIMA (Research Laboratories, NTT Electrical Communications Laboratories).

†† NTT 電気通信研究所基礎研究所

にパターンマッチングする機能，(b)知識ベース内のデータに対する操作，計算などを行う関数等を適宜起動する機能，が要求される．しかし，これらを同一の推論メカニズムのもとで統一的に扱う簡便な記述がない．

(2) 対象の規模が大きくなるに従い，特定の構造のマッチングの際，関係のないフレームのアクセスが増加する．そのため，探索範囲を絞るための注視点の制御 (focus of attention) 機能が要求される．しかし，その点に対する配慮がない．

(3) 推論規則は，ある条件が満足されるまで試行錯誤的に繰り返し適用される．もし，個々の試行で，対象が変更されていれば，これを元の状態に戻し，再度別の推論規則を適用しなければならない．そのため，変更された対象を自動的に以前の状態に戻す機能 (自動回復機能) が必要である．しかし，この機能がない．

これらの問題を解決するため，以下の機能を取り入れた論理型推論メカニズム PRIME[8] (PRolog wIth fraME unifier) を作成した．

(1) フレームの構造のマッチングと変更の統一的記述．

(2) 推論規則の適用の際の注視点の制御．

(3) 変更フレーム情報の自動回復機能．

これにより，フレーム型表現に対する柔軟な推論規則の記述，制御が可能となる．また，試行錯誤が多く，探索範囲が膨大となる設計問題に対しても，上記の機能を取り入れたことで設計操作に適用できる．

現在，PRIME は Maclisp (DEC 2060), および Zetalisp (SYMBOLICS 3600) 上にインプリメントされており，知識表現推論環境 KRINE[2] (Knowledge Representation and INference Environment) の推論機構として位置づけられている．また，計算機の論理設計問題を扱うエキスパートシステム DE[9](Design Expert system) で設計知識 (合成，検証の手続き) の記述，制御に用いられている．

以下では，論理型推論メカニズム PRIME の基本的考え方と処理のメカニズムについて述べる．2章では，基本的考え方について述べる．3章では，PRIME による推論規則の記述について述べる．4章では，その実現法について述べる．5章で，PRIME を用いた応用例，考察を行う．

2. 基本的考え方

2.1 KRINE での知識表現

KRINE は，設計問題を扱うエキスパートシステム (設計型エキスパートシステム) の構築用ツールであり，以下の特徴を持つ．

(1) フレーム理論に基づく知識表現を採用し，設計対象の階層的表現が容易で，複雑かつ巨大な知識ベースをモジュール性良く構築できる．

(2) 知識の使用目的，方法に応じた複数の知識表現パラダイムを提供する．本論文で述べる PRIME による論理型表現をはじめ，手続き表現，操作指向表現，ルール表現，オブジェクト指向表現，およびグラフィックス表現パラダイムを持つ．詳しくは文献 2) を参照のこと．以下では，主に KRINE での推論手続きに関するフレームの構造，操作について述べる．

2.1.1 フレームの構造

フレームシステムは，階層的な構造と遺伝属性を持つフレームを単位として構成される．各フレームは，システム中でユニークな名前を持ち，任意個数のスロットから構成される．各スロットは，スロットの値，このスロットが持つべきデータの値の性質を指示するデータタイプ，親フレームから子フレームへ受け継ぐべき属性を指示するインヘリタンス・ロールを持つ．また，フィールドは，スロットの付加情報で，スロットに対する操作 (参照，更新) が行われた場合に起動される手続きを書くことができ，デーモンとして利用できる．さらに，個々のフレームには，グループを指定することができ，推論範囲の制限などに用いる．論理回路設計で用いる算術論理回路 ALU のフレーム表現を図1に示す．このフレームは GR−CONN というグループが指定され，親フレームは $$ BINARY_ALU で，スロットとして，TYPE, A, B, C, FUNCTION を持つ．

```
ALU1
<Group>:GR-CONN

 Slot
TYPE:
A:
B:
C:
FUNCTION:

 Prototype is $$BINARY_ALU
Top             Role  Datatype     Data
$$DATA-PART     (U)   <ATOM>
$$BINARY_ALU    (U)   <FRAME>      L14
$$BINARY_ALU    (U)   <FRAME>      L22
$$BINARY_ALU    (U)   <FRAME>      L23
$$DATA-PART     (U)   <S-EXPR>      One of:
((& (ALU1 C) (ALU1 A) (ALU1 B)) (ALU1 AND))
((OR (ALU1 C) (ALU1 A) (ALU1 B)) (ALU1 OR))
.....
```

図1 フレームの構造 (ALU1) の例
Fig. 1 Frame structure example (ALU1).

2.1.2 フレーム操作

フレームに関する操作は以下に示すものがある.

(1) フレームおよびスロットの構造の参照, 変更

(2) スロット値の参照, 格納, 消去

(3) 階層関係を利用した構造の変更 (フレームの生成, 削除など)

これらのフレーム操作はシステム関数 (Lisp 関数) として作成されている.

2.2 簡便な推論のための記述

PRIME ではフレーム構造のマッチング, およびフレーム構造の変更の両者を論理の枠組みのなかで扱うため, 統一化の対象として Prolog の節ばかりでなく, フレームのアクセス, Lisp 関数もその対象とするように拡張した.

2.2.1 フレーム表現との融合

フレーム表現との融合のためには, 以下のことが必要である.

(1) PRIME の節をフレーム中に定義でき, フレーム操作の一環として節の変更ができること.

(2) PRIME からフレームをアクセスするための記述ができること.

(1)については, データタイプが Prolog であるフレーム中のスロットの値として節を格納できるようにした. これにより, 節の定義の変更もフレーム操作として実現できる.

(2)については, 知識ベースのアクセスを柔軟に記述するためフレーム一個を一つの事実節 (fact clause) に対応させ, 統一化の対象とすることで対処した. 従来のシステム[6),17),18)]では, マッチングの対象がスロットの値だけというような制約がある. PRIME ではこの種の制約を受けない形とし, スロットの値だけでなく, フレーム名, スロット名も統一化の対象とした.

2.2.2 手続き表現との融合

Lisp 関数の呼出しを可能とする機能を持たせ, 手続き表現との融合をはかる. ただし, Prolog 述語から Lisp 関数を呼び出す際, 従来のシステム[13)]では, ある関数を実行した結果の値により, バックトラックをするか, しないかを決める機能は有していない. これは, 試行錯誤的な推論の記述に不向きである. そのため, Lisp 関数の実行結果 (return value) が失敗の情報を示す場合, 自動的にバックトラックを起こさせる機能を追加し, Lisp 関数呼出し時の試行錯誤的な推論の記述を可能とした.

2.3 設計操作知識の処理強化機能

設計型エキスパートシステムにおける設計操作の記述, 推論の制御には以下に示す機能が必要である.

2.3.1 注視点の制御機能

大規模な設計対象については, 特定の構造のパターンマッチングの際, 盲目的探索を避けるため探索範囲を絞るための注視点の制御機能が必要である. ブラックボードモデルでは, 選択すべきルールの集合を制限できるが, 推論の過程で探索すべきフレームを動的に制限する機能はない.

PRIME では利用者が推論の探索範囲を指定することで, 注視点の制御を行うこととした. そのため以下の二つの方法による探索範囲の限定の指定法を設定した.

(1) フレームの階層関係を利用して, ある階層のフレームの下位/上位部分だけをアクセスする.

(2) フレームをいくつかのグループに分割し, この分割されたグループのみをアクセスする.

2.3.2 自動回復機能

Prolog では, 節の定義, 削除のための述語 (ASSERT, RETRACT) による操作はバックトラックが起きても元には戻らない. また, 回復機能を持つシステムでも, 利用者がその都度, 意識して操作を行わなければならず非常に煩雑である. たとえば, いったん処理が終わった後でなければ回復操作ができない, 回復操作のための手続きを利用者が実行しなければならないなど.

PRIME では, フレームの変更操作が行われ, その後バックトラックが起きた場合は, 自動的に変更前のフレームの状態に戻す機能を推論時のバックトラックと同期させることで実現した.

3. PRIME による推論規則の記述

3.1 推論規則の記述

一般の Prolog の節の表記法を図 2(a) に示す. PRIME では, 節は図 2(b) のように各リテラルを+, −で結び表現する. また, 変数は＊で始まるシンボルで表す. リストの結合 (Append) の節の例を図 2(c) に示す.

節の各項には 2 章で述べた基本的考え方に基づき, 以下の 3 種類の操作が記述できる.

(1) Prolog 述語の呼び出し

論理的な操作の記述のための Prolog 述語の項は, 以下のように記述する.

```
(1) P←.
(2) P←Q1∧Q2∧...QN.
(3) ←Q1∧Q2∧...∧QN.
```

図 2(a) Prolog のプログラム
Fig. 2(a) Prolog program.

```
(1) +(P):
(2) +(P)-(Q1)-(Q2)-...-(QN):
(3) -(Q1)-(Q2)-...-(QN):
```

図 2(b) PRIME のプログラム
Fig. 2(b) PRIME program.

```
+(Append nil *x *x):
+(Append (*a . *x) *y (*a . *z))
     -(Append *x *y *z):
```

図 2(c) PRIME の節の例 (Append 述語)
Fig. 2(c) PRIME clause example(Append predicate).

(〈述語名〉〈項の並び〉)

たとえば，APPEND 述語の呼び出しは −(APPEND (a b) (1 2)): と記述する．

(2) Lisp 関数の呼び出し

フレームの変更，操作などの Lisp 関数の呼び出しの項は以下のように記述する．

(〈関数名〉〈パラメタの並び〉)

これにより，関数をその引数に適用する．ただし，引数は入力パラメタとして扱い，その値は実行時に定まっている必要がある．たとえば，−(DELETE−FRAME *X): は *X に束縛されているフレームの消去を行う．

(3) フレームのアクセス，マッチング操作の記述

図1に示したフレームの構造を基本として，フレーム名，スロット名，データの値（各データタイプに応じた値）の並びから成る三つの組を項の記述とする．推論においては，この三つ組で十分であり，データタイプ，インヘリタンスロールなどは統一化の際に自動的にチェックされる．以下に三つ組の表現を示す．

(〈フレーム名〉〈スロット名〉〈データの値〉)

たとえば，スロット名が FUNCTION であるようなスロットを持つフレーム名とそのスロットの値を得るには，−(*X FUNCTION *Y): と記述する．

3.2 推論範囲の限定の記述

推論の探索範囲を絞るための述語を以下に示す．利用者が，これらを適宜用いることで，アクセスすべきフレームの範囲を絞ることができ，効率良い推論を行うことが可能となる．また，これらの述語はバックトラック時に，以前の値に戻るので制限すべき状況に応じた制御ができる．

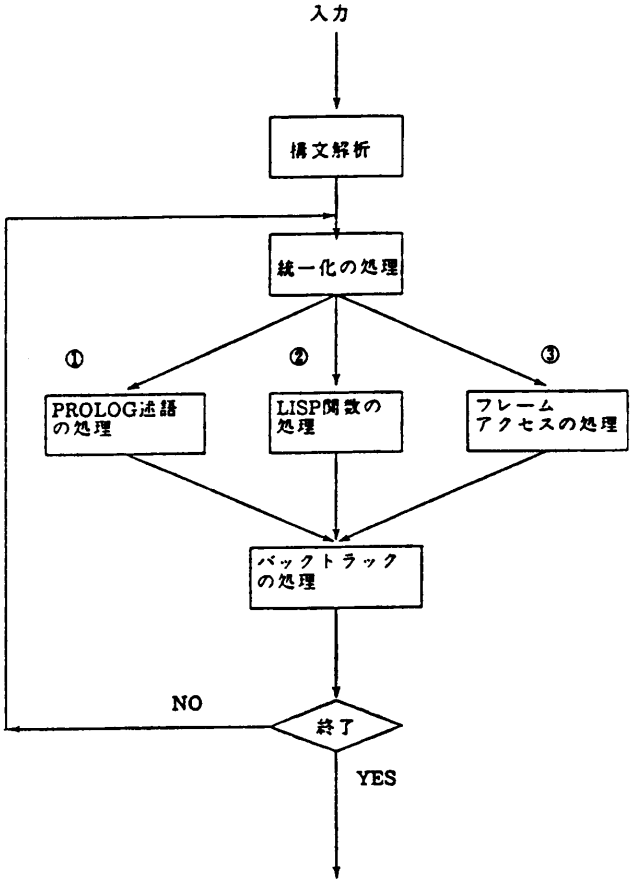

図 3 論理型推論メカニズム PRIME の処理概要
Fig. 3 Process flowchart of PRIME processing.

(1) フレームの階層関係を指定する述語

PARENT−OF/PROGENY−OF などの述語．たとえば，−(PROGENY−OF *X ALU): とすると，以後 *X は ALU の下位の階層のフレームだけに探索範囲が限定される．

(2) フレームのグループを指定する述語

& LIMITS 述語．たとえば，−(& LIMITS (A)): は，グループAだけにフレームのアクセスを限定し，他のグループとの分離を行う．

4. PRIME の実現法

処理の概要を図3に示す．まず，システムは入力された文に対して，構文解析を行う．次にフレームのアクセス，Lisp 関数の起動を含んだ統一化の処理を行う．統一化に失敗した節はバックトラックを引き起こす．この時，フレームに対する変更があれば，自動的にフレームの回復処理を行い，次の候補に対して，再度統一化の処理を行う．

4.1 構文解析

入力文についてシステムは，(1) Prolog 述語の呼び出し，(2) Lisp 関数の呼び出し，(3)フレームのアクセスの三種類を各リテラルの先頭要素の属性を取り出すことで判定し，それぞれに応じた述語呼出しの形式に置き換える．(2)の場合は Lisp 関数の呼び出し述語 CALL の形式 (CALL 〈関数名〉〈パラメタの並び〉)，(3)の場合はフレームのアクセス述語 FCALL の形式 (FCALL 〈フレーム名〉〈スロット名〉〈データの値〉)にそれぞれ置き換える．もし，同じ名前が重複して定義された場合は(1)，(2)，(3)の優先順位で扱う．ただし，利用者が(2)を優先したければ CALL を，(3)を優先したければ FCALL を付けることで，(1)の Prolog 述語として扱われるので，それが優先的に処理される．

4.2 統一化の処理

通常の Prolog 述語はそのまま統一化の対象となる．以下では，Lisp 関数の呼び出し述語とフレームのアクセス述語について述べる．

(1) CALL 述語の場合

CALL の次のパラメタを関数名，残りを引数として，関数を適用する．もし，引数のなかに変数があれば，その変数に束縛されている値を取り出してから適用する．

(2) FCALL 述語の場合

統一化の実行時に，フレームアクセスの候補となる(〈フレーム名〉〈スロット名〉〈データの値〉) から成る三つ組を作り出す．ただし，フレーム名が束縛されている場合は，そのフレーム内だけで三つ組の候補を絞る．また，スロット名が束縛されている場合は，スロット名からフレーム名をポイントしているインデックスを利用して候補フレームを絞る．さらに，データの値が束縛されている場合は，そのデータの値だけをもつ候補をチェックし，三つ組を作成する．このように，各パラメタの束縛状態に応じて，無駄なフレームのアクセスを避ける工夫を施し，効率化を図った．

4.3 バックトラックの処理

実行に失敗した節は，バックトラックを起こし，その過程で生じた変数への代入 (substitution) を解除し，自動的に次の候補となる節 (Prolog 述語，あるいはフレームの候補) を探す．節の実行に失敗する原因には，次の三つがある．

(1) Prolog の述語名が未定義，または FAIL 述語を実行した場合．

(2) Lisp 関数を適用した結果，失敗を示す値が返された場合．

(3) フレームに該当するパターンがなくアクセスに失敗した場合．

(2)の場合，必ずしも NIL が失敗の値とは限らない (たとえば，Maclisp の関数 TERPRI は常に NIL を返す) ので，2種類の値* を設けることで対処した．

4.4 自動回復機構 (UNDO メカニズム)

試行錯誤的な推論を矛盾無く行うため，フレームの変更操作が行われた後で，バックトラックが生じた場合，以前のフレームの状態に戻す制御が自動的に適用される．推論の過程で，フレームを変更する関数が起動されると，フレームの変更前の情報を必要最小限のものだけ保存する**．バックトラックが生じると，この情報をもとに以前のフレームの情報を復元する．PRIME では，フレームを変更する可能性のある関数 (ユーザの作成した関数はすべて) を含む節を事前にチェックし，この節の実行の前に UNDO メカニズムを行うためのシステム関数を呼び出し，フレームの変更操作が行われた後，バックトラックに同期させて，フレームの情報を回復する．

5. 応用例

高度な設計問題であるディジタル回路の自動設計を行う設計型エキスパートシステム DE[9] の中で PRIME は，機能ブロックレベルの合成，検証の手続きの記述[14),15)] に用いられている．ここでは，その一部を示し，設計操作知識の表現，適用の制御について考察する．

5.1 設計操作知識の記述

機能ブロックレベルの検証手続きの例として，ある設計対象が与えられた時，そこでのデータパス系の検証を行い，もし検証不成功の場合は，転送路の欠落などの原因を捜し，それを補正 (適当な転送路につなげる)，アドバイスする例を示す (詳細は文献14)を参照のこと)．検証対象のブロック図を図 4，そこで用いられているフレームの例を図 5 に示す．推論範囲の限定のため，転送路を表すフレームは GR−LINE グループ (図5の L1)，また Function スロットに，SET,

* CALL 述語の場合は 'FAIL, EVAL-CALL 述語の場合は NIL をバックトラックを引き起こすために返す値と設定する．

** 実際には，フレームの情報をそのまま保存せず，フレーム操作に対する逆関数を記憶させる．たとえば，フレームの削除 (DELETE-FRAME 関数) に対しては，フレームの作成 (MAKEF 関数) を対応させ，その情報を格納する．

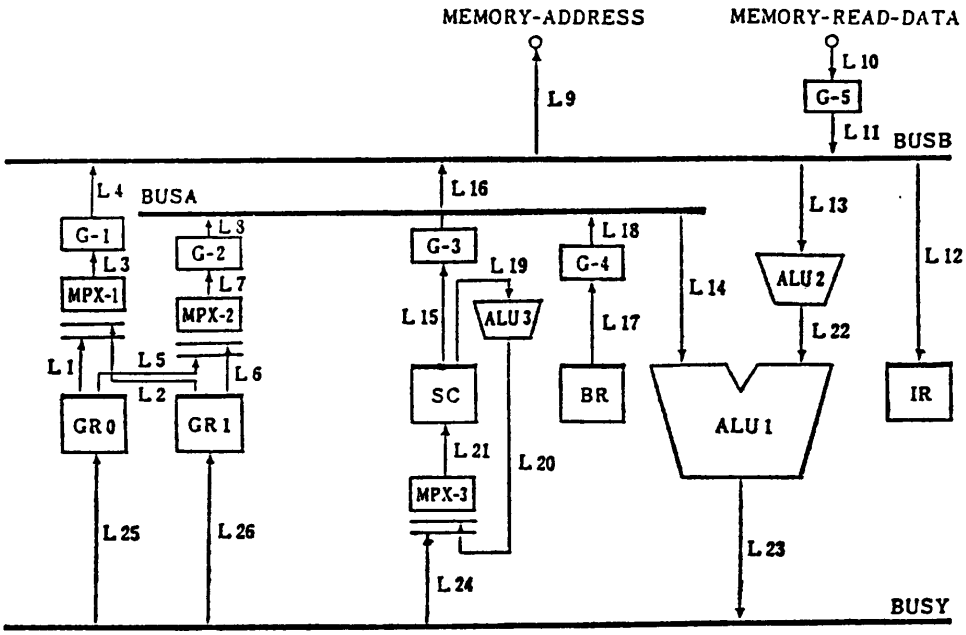

図 4 データパス検証例のブロック図
Fig. 4 Block diagram of data path verification example.

```
    L1
  <Group>:GR-LINE
                    Prototype is $$LINE
    Line 1
 Slot                Top              Role  Datatype     Data
TYPE:                $$DATA-PART      (U)   <ATOM>
FROM:                $$LINE           (U)   <S-EXPR>     (GR0 OUT)
TO:                  $$LINE           (U)   <S-EXPR>     (MPX_1 IN1)
FUNCTION:            $$DATA-PART      (U)   <S-EXPR>

    GR0
  <Group>:GR-CONN
                    Prototype is $$REGISTER
 Slot                Top              Role  Datatype     Data
TYPE:                $$DATA-PART      (U)   <ATOM>
OUT:                 $$REGISTER       (U)   <FRAME>      One of: L1 L5
IN:                  $$REGISTER       (U)   <FRAME>      L25.....
FUNCTION:            $$DATA-PART      (U)   <S-EXPR>      One of:
                     ((SET GR0 (GR0 IN)) (GR0 SET))
                     ((CONN (GR0 OUT) GR0) (GR0 T))

    BUSB
  <Group>:GR-CONN
                    Prototype is $$BUS
 Slot                Top              Role  Datatype     Data
TYPE:                $$DATA-PART      (U)   <ATOM>
OUT:                 $$BUS            (U)   <FRAME>      One of: L9 L13 L12
IN:                  $$BUS            (U)   <FRAME>      One of: L4 L16 L11
FUNCTION:            $$DATA-PART      (U)   <S-EXPR>
                     ((CONN (BUSB OUT) (BUSB IN)) (BUSB T))
```

図 5 設計対象のフレーム表現の例
Fig. 5 Design object frame structure example.

```
    VERIFY
                    Prototype is PLPROC
 Slot               Top              Role  Datatype     Data
CLAUSE:             PLPROC           (U)   <PROLOG>
  +(VERIFY *D <- *S)-(VERIFY-E (*D <- CONN *S) *PATH *E):
  +(VERIFY-E (*D <- *OP *S) *PATH *E)
       -(VERIFY1 (*OP *D *S) *PATH)
       -(EXPLAIN (*OP *D *S) *PATH *E):
  +(VERIFY-E (*D <- *OP *S) *PATH *R)
       -(WHY-NOT1? *R (*OP *D *S) *PATH)
       -(EXPLAIN1 *R (*OP *D *S) *PATH *REASON)
       -(SUGGEST1 *R (*OP *D *S) *PATH *SUGGESTION)
       -(REVISE-DATA-PATH *NEW-LINE *SUGGESTION *D)
       -(ASK-IF-ACCEPT *NEW-LINE *SUGGESTION *D):
  .....
REFUTE:             PLPROC           (U)   <PROLOG>
  -(verify *x <- *y):

    FIND-FUNCTION
                    Prototype is PLPROC
 Slot               Top              Role  Datatype     Data
CLAUSE:             PLPROC           (U)   <PROLOG>
  +(FIND-FUNCTION SET *A *B)-(INST&PROGENY-OF *X $$REGISTER)
                            -(*X FUNCTION ((SET . *A) *B)):
  +(FIND-FUNCTION *OP *A *B)-(&LIMITS (GR-CONN))
                            -(*X FUNCTION ((*OP . *A) *B)):
  .....


    FIND-PATH
                    Prototype is PLPROC
 Slot               Top              Role  Datatype     Data
CLAUSE:             PLPROC           (U)   <PROLOG>
  +(PATH *FROM *TO *LINE)-(&LIMITS (GR-LINE))-(*LINE FROM *FROM)
   .                        -(*LINE TO *TO):
  +(A-PATH *FROM *TO *NET)-(PATH *FROM *TO *NET):
  +(A-PATH *FROM *TO *CTL)-(FIND-FUNCTION SET (*TO *FROM) *CTL):
  +(A-PATH *FROM *TO *MPX)-(FIND-FUNCTION CONN (*TO *FROM) *MPX):
  .....
    REVISE-DATA-PATH
                    Prototype is PLPROC
 Slot               Top              Role  Datatype     Data
CLAUSE:             PLPROC           (U)   <PROLOG>
  +(REVISE-DATA-PATH *NEW-LINE *FROM *TO)
       -(MAKE-LINE *NEW-LINE)
       -(PUTVALUE *NEW-LINE FROM *FROM)
       -(PUTVALUE *NEW-LINE TO *TO):
  +(ASK-IF-ACCEPT *NEW-LINE *FROM *TO)
       -(MPR* LINE  *NEW-LINE  from  *FROM  to  *TO  is acceptable?)
       -(IF (EQ (FUNC READ*) Y) (TRUE) (FAIL)):
  .....
```

図 6 設計ルールの例 (一部)
Fig. 6 Design rule example.

CONN のデータを持つものは，GR−CONN グループ (図5の GR 0, BUSB) に分割されている．ルールの例を**図 6** に示す．

このルールを実際に起動するためには，データパスの検証を行う二つの機能ブロックをパラメタとして，VERIFY フレームの REFUTE スロットにメッセージを送る．システムはそれを実行文 (Goal 節) と解釈し，検証のルール (検証が成功する場合は VERIFY 述語，失敗する場合は VERIFY-E 述語) を働かせ，もし必要とあれば，補正を行う．

まず，検証すべき対象の直接の転送路を捜す (PATH 述語)．もしなければ，Function スロットに代入 (SET)，または参照 (CONN) のデータを持つものを捜し (FIND-FUNCTION 述語)，つながりを見つけ，順次データパスの検証を行う．もし，検証の条件を満足しないデータパスがあると，その原因を探索し

(WHY-NOT?, SUGGEST 1 述語), それぞれの原因に応じた補正ルールを起動する. ここでは, 候補となるデータパスの補正を行い, その結線関係を定義する (REVISE-DATA-PATH 述語により, LINE の各端子 FROM, TO に BUS をつなげる). その結果をユーザに問い合わせて (この部分は他の評価関数を用いることもできるが, ここでは, 簡単のために問い合わせにした), それが妥当かどうかの判断を求める (ASK-IF-ACCEPT 述語). もし, 妥当でないと判断された場合 (条件を満足しない場合) は, 定義された結線関係を解除し, 元の状態に復元し, 違うデータパスを見つけだす操作を行う.

表 1 推論範囲指定の効果
Table 1 Effect of frame access limitation.

実行時間	検証成功の例 -(verify sc <- br):	検証不成功の例 -(verify br <- gr0):
(1)探索範囲の指定なしの場合	100	100
(2)グループの指定ありの場合	74.1	76.6
(3)グループと親子関係の指定ありの場合	66.9	76.5

(1)を100とした場合の実行時間の比率
表中の(1), (2), (3)は以下のプログラムにより実行される.

```
(1) +(PATH *FROM *TO *LINE)
      -(*LINE FROM *FROM)
      -(*LINE TO *TO):
    +(FIND-FUNCTION *OP *A *B)
      -(*X FUNCTION ((*OP . *A) *B)):

(2). +(PATH *FROM *TO *LINE)
       -(&LIMITS (GR-LINE))
       -(*LINE FROM *FROM)
       -(*LINE TO *TO):
     +(FIND-FUNCTION *OP *A *B)
       -(&LIMITS (GR-CONN))
       -(*X FUNCTION ((*OP . *A) *B)):

(3). +(PATH *FROM *TO *LINE)
       -(&LIMITS (GR-LINE))
       -(*LINE FROM *FROM)
       -(*LINE TO *TO):
     +(FIND-FUNCTION SET *A *B)
       -(INST&PROGENY-OF *X $$REGISTER)
      -(*X FUNCTION ((SET . *A) *B)):
     +(FIND-FUNCTION *OP *A *B)
      -(&LIMITS (GR-CONN))
      -(*X FUNCTION ((*OP . *A) *B)):
```

5.2 考　　察

(1) 設計操作知識の表現についての考察

推論の記述は, 従来の Lisp 関数だけで行っていたシステム[4),5)] と比較した場合, フレームアクセスの記述, 適用範囲の指定のためのシステム関数の呼び出しの記述が不要である. また, 変数を用いることでマッチングの対象, 推論範囲の指定に対する記述に柔軟性が増している. 一方, 述語論理を組み合わせたシステム[6),12),16)~18)] と比較した場合, PRIME は, フレームのパターンマッチングの記述, フレームの探索範囲の絞り込み制御が簡便に記述できる (PATH, FIND-FUNCTION 述語). さらに, 試行錯誤的な知識の適用のための環境設定 (UNDO 機能) が自動的に行われ (REVISE-DATA-PATH 述語), バックトラック時 (ASK-IF 述語で FAIL する時)には変更されたフレームを元に戻して次の推論を行うので, 矛盾無く設計が行える.

(2) 探索範囲の制限の効果についての考察

この例では, データパス, 機能ブロックの検索時 (PATH, FIND-FUNCTION 述語) に, フレームの階層関係などを用いた探索範囲の指定が入っている. 全く範囲を指定しない場合と比較した効果を表 1 に示す. 範囲を指定することで, 2 割から 3 割の実行時間の減少がみられる. 全体のフレーム数 (この例では, 137 フレーム) が増加し, グループ範囲の指定 (この例では 2 グループ) が細かくなれば, さらに実行時間の比率は下がると考えられる.

6. むすび

フレーム型知識表現の構成要素に対する構造の認識のためのパターンマッチング機能と, 設計操作適用時のバックトラック制御機能を付加した論理型推論メカニズム PRIME の基本的考え方, 実現方式, 具体例および考察について述べた. PRIME の特徴は以下のとおりである.

(1) フレーム構造のパターンマッチングと Lisp 関数による知識ベース内のデータに対する演算操作の両者を相互に呼び出すことができ, 推論手続きの簡便な記述を可能にした.

(2) 推論の適用範囲の限定をフレームの階層関係, グループ関係に基づき指定できる記述を可能にした.

(3) 変更されたフレームの情報回復機能を内蔵し, 試行錯誤的推論を可能にした.

具体例について検討した結果, 設計操作知識の表現については, 簡便な操作の記述, 試行錯誤的な知識の適用が柔軟に行えた. また, 効率良い推論のためには, フレームの探索範囲を指定し, マッチングの対象を減らすことが有効であることを示した. ただし, 設計問題に求められる機能は各種のものがある. 今後はさらに推論範囲の指定などの機能面, およびバックトラックの方法などの実行メカニズムの面で考察を深める必要があると考える. また, フレーム数が増加するに従い, パターンマッチングの対象が増加し, 実行時

間が問題となると予想されるので，さらに効率良い推論を可能にするため，PRIME の高速化をはかる予定である．

謝辞 日頃御指導いただく山下紘一第一研究室室長，および，有益な討論，助言をしていただいた，小川裕氏，尾内理紀夫氏，高木茂氏，斉藤宗昭氏，および後藤滋樹氏に謝意を表します．また，システム関数，および UNDO メカニズムについては，菅原俊治氏の貢献によるところが多く感謝します．

参考文献

1) Barr, A. and Feigenbaum, E. A.: *The Handbook of Artificial Intelligence*, Vol. 1, pp. 140-222, William Kaufmann, Inc., Los Altos, California (1981).
2) Ogawa, Y., Shima, K., Sugawara, T. and Takagi, S.: Knowledge Representation and Inference Environment—KRINE, *Proc. International Conference on FGCS*, pp. 643-651 (1984).
3) Minsky, M.: A Framework for Representing Knowledge, in Winston, P. (ed.), *The Psychology of Computer Vision*, pp. 211-277, McGraw-Hill, New York (1975).
4) Bobrow, D. G. and Winograd, T.: An Overview of KRL-0: A Knowledge Representation Language, *Cognitive Sci.*, Vol. 1, No. 1, pp. 3-46 (1977).
5) Smith, R. G. and Friedland, P.: UNIT Package User's Guide, Stanford Heuristic Programming Project Memo, Memo HPP-80-28 (1980).
6) Kehler, T. P. and Clemenson, G. D.: KEE, The Knowledge Engineering Environment for Industry, Intelligenetics (1983).
7) Bobrow, D. G. and Stefik, M.: The LOOPS Manual, XEROX PARC Knowledge-Based VLSI Design Group Memo, KB-VLSI Design Group Memo, KB-VLSI-81-13 (1981).
8) 島 健一：知識ベースシステムにおける論理的推論メカニズムの検討，Proc. Logic Programming Conference (1984).
9) 高木 茂，小川 裕，斉藤宗昭：設計型エクスパートシステム DE-0 の検討，情報処理学会第26回全国大会講演論文集，pp. 961-962 (1983).
10) Clocksin, W. F. and Mellish, C. S.: *Programming in Prolog*, p. 279, Springer-Verlag, Berlin, Heidelberg (1981).
11) Mizoguchi, F., Ohwada, H. and Katayama, Y.: LOOKS: Knowledge Representation System for Designing Expert Systems in a Logic Programming Framework, *Proc. International Conference on FGCS*, pp. 606-612 (1984).
12) 中島秀之：知識表現用言語としての Prolog/KR 情報処理学会論文誌，Vol. 25, No. 2, pp. 180-186 (1984).
13) Robinson, J. A. and Sibert, E. E.: LOGLISP: An Alternative to PROLOG, in Hayes, J. E., Michie, D. and Y-H. Pao (eds.), *Machine Intelligence 10*, pp. 399-419, Ellis Horwood Ltd., Chichester, West Sussex (1982).
14) 高木 茂：データパスの検証・補正への一アプローチ，情報処理学会論文誌，Vol. 26, No. 1, pp. 9-18 (1985).
15) 島 健一，高木 茂：知識エディタ KE-0 における手続き付加について，情報処理学会第26回全国大会講演論文集，pp. 965-966 (1983).
16) 堀 浩一，斉藤忠夫，猪瀬 博：帰納的推論の記述に有用な知識表現の一提案，情報処理学会論文誌，Vol. 24, No. 1, pp. 72-79 (1983).
17) Charniak, E.: A Common Representation for Problem-Solving and Language-Comprehension Information, *Artif. Intell.*, Vol. 16, No. 3, pp. 225-255 (1981).
18) 伊藤秀昭，上野晴樹：フレーム・モデルと述語論理の結合について，情報処理学会第29回全国大会講演論文集，pp. 1211-1212 (1984).
19) Furukawa, K., Takeuchi, A., Kunifuji, S., Yasukawa, H., Ohki, M. and Ueda, K.: Mandala: A Logic Based Knowledge Programming System, *Proc. International Conference on FGCS*, pp. 613-622 (1984).

(昭和61年1月 9 日受付)
(昭和61年6月18日採録)

島 健一（正会員）

1954年生．1976年北海道大学工学部電気工学科卒業．1978年同大学院情報工学専攻修士課程修了．同年 NTT 武蔵野電気通信研究所入所．以来，関係データベースマシン，知識ベース構築用システムの研究開発に従事．主に，フレーム理論を基礎とした知識表現と述語論理を融合した推論システムの基礎研究を行う．知識獲得，学習システムなどの研究に興味を持つ．現在，NTT 基礎研究所情報通信基礎研究部研究主任．電子通信学会会員．