データベース・システム 57－2
（１９８７．１．１９）

LISP BASE－S式による知識表現を扱うDBMS

小林 哲雄　田中 譲
北海道大学 工学部

現在までに数多くの知識表現言語が提案されており一長一短がある。知識表現言語はほとんどがS式によって記述することができる。知識を利用するシステム自身も多くがS式によって記述されている。S式という共通の構文に着目し、異なる知識表現であってもS式を単位として統一的に知識を管理するDBMS－Lisp Baseを設計した。Lisp Baseは柔軟なデータ構造を持つS式を二次記憶上のDBに登録し、知識表現からの各種の検索要求に対応できる。S式を終端ノードとその位置情報の組の集合で表現することでDBにデータ構造を導入した。DB中のデータ、DDL、DML、ホスト言語をS式で統合したため、DBと推論機構との結合が容易である。Lisp Baseは各種の知識表現を管理する知識ベースの核になるデータベース管理システムである。

LISP BASE SYSTEM

A DBMS storing knowledge represented in S-expressions

Tetsuo KOBAYASHI and Yuzuru TANAKA
Electrical Engineering Department, Hokkaido University,
Sapporo, Hokkaido, 060, Japan

Various kinds of knowledge representation languages(KRLs) and knowledge processing systems(KPSs) use their own ways of knowledge descriptions with the merits and demerits. However most of KRLs and KPSs can be described in S-expressions. We propose in this paper "Lisp Base system" as a DBMS to manage different kinds of KRLs based on S-expressions. In this system, S-expressions can be stored in secondary storage in a form suitable for various KPS queries.

In "Lisp base system", its host language, DDL and DML are all represented in S-expressions. Therefore it is easy to introduce an inference mechanism, and hence, the system provides a basis for knowledge base systems.

1. はじめに

従来、LISPで記述された知識情報を扱うシステムは、その推論などに必要なルールやファクトなどの知識が、全て記憶空間に存在することを前提にしていた。

近年、より多数の知識を扱うエキスパート・システム(ES)などの実用化や、より小さな計算機上での構築の要求が、日増しに増加している。このため、知識を二次記憶上のデータ・ベース(DB)へ格納し、必要なものだけを取り出しうるようなシステムを構築するほうが望ましい。

現在までに数多くの知識表現言語が提案されているが、各々に一長一短があり万能のものはない。これら知識表現言語はそのほとんどがS式によって記述される。また、知識を利用するアプリケーション自身も多くがS式によって記述されている。S式という共通のデータ構造に着目することで、異なる知識表現であってもS式を単位として統一的に知識を管理することができる。

本研究では、はじめに各種のS式によって記述される知識表現についてなにを手掛かり(キー)に検索を行っているかを考察し、それらのなかで検索要求の共通するプリミティブを抽出した。次に、共通する検索要求を記述出来るようにDBアクセス関数の構文を定義した。必要な各種の検索要求に対応するのに適当な二次記憶でのS式集合の蓄積構造を導入した。

次のようなことを本研究では目指している。

1) 大量の異種の知識表現であっても一つのシステムによって統一的に管理すること。

例えば一つのESにおいて、知識表現としてフレームとルールとユーザー定義関数をすべて使いたい場合でも、LISP BASEを利用すれば、それぞれの知識表現ごとに異なるDBMSを準備する必要がない。

2) プログラミング言語とデータベースとの融合

このシステムの最大の特徴は、DDL、DML、DBに登録されている内容、検索結果、ホスト言語、すべてがS式であることである。データであっても関数であってもS式としてDBに入るので、検索結果をEVALすることで、モデルを書き換えたり、DBへの新たな操作を行うことも可能である。

S式によってプロダクション・ルール、フレーム、CDなど各種の知識表現や関数・手続き的知識を記述できる。LISP BASEは幅広い応用が考えられる。

2. S式集合の検索

この章では、S式による代表的ないくつかの知識表現において、S式で記述された知識の集合を検索するとはどのような操作であるか、また、何をキーに検索をしているかについて述べる。その操作に共通するアクセスのプリミティブは何かについて考察する。[1]

2.1. LISP関数

計算機上のセルとポインタによる表現ではなく、S式で表現されたLISP関数をDBから検索する場合、文字アトムである関数名を用いる。この関数名という値の出現する位置はS式のCADRの位置だけであると一意に決まっている。このように、検索キーの出現するべきS式上の位置が決まっていることを、『位置の強い指定』と呼ぶことにする。 DBからLISP関数を検索することは、『位置の強い指定』による検索であり、かつ、関数名という値の指定による検索である。

2.2. CDネット、意味ネット

Conceptual Dependencyネットや意味ネットなどは、ノードとリンクと呼ばれる有向線分によって概念を表現する。このような表現を行うためのS式による構文はいろいろ考えられ、定まったものはない。たとえば、有向線分の始点ノードに着目して、そのノードからのリンク・タイプと終点ノードをリストにして、次のようにネットワークを表現することができる。

```
(始点ノード
      (リンクタイプ1 終点ノード1)
      (リンクタイプ2 終点ノード2)
                :
      (リンクタイプn 終点ノードn) )
```

リンクタイプ-1とリンクタイプ-2のリストは位置が入れ替わってもよい。図1で示す様に、リンク・タイプ名の出現位置はある程度決まっている。このように、検索キーのあるべきS式上の位置が一意には決まらず、ある条件によって制限されていることを『位置の弱い指定』と呼ぶことにする。

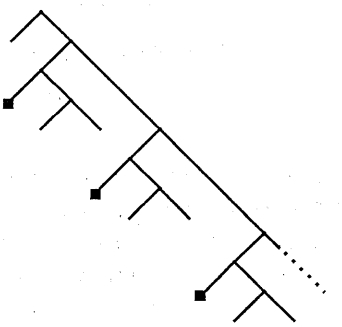

図1 リンクタイプ名の出現位置

始点ノード名からリンクタイプとそれに対応する終点ノード名を検索することは、始点ノード名と言う値の指定による検索であり、かつ、その『位置の強い指定』による検索である。

また、ある終点ノードにリンクしている始点ノード名を検索することは、検索キーである終点ノードという値の指定による検索であり、かつ、その検索キーの『位置の弱い指定』による検索である。

2.3. フレーム

フレームをS式で記述したFRLは、次のような構造をしている。

```
(<frame>
    (<slot-1> (<facet-1> <data-1>)
              (<facet-2>  <data-2>)
                    :
              (<facet-n>  <data-n>) )
    (<slot-2> (<facet-1> <data-1>) ...)
                :
)
```

ここで、フレーム名、スロット名、ファセット名を指定して、対応するデータを取り出す検索を考えることにする。

1) フレーム名をキーに一つのS式を選ぶ。
2) 1)のS式の中で、スロット名をキーにしてスロットを選ぶ。
3) さらに、2)のスロットの中で、ファセット名をキーにしてファセットを選ぶ。
4) 3)のファセットの中のデータを取り出す。

1)は、『位置の強い指定』と値の指定による検索である。

2)と3)は、『位置の弱い指定』と値の指定による検索である。

4)は、値の指定のない、『位置の強い指定』のみによる検索である。

フレームの場合、前の検索の結果を利用して次の検索を行う副次検索をする必要がある。

2.4. ルール・ベース

プロダクション・システム(PS)は、ルール・ベースよりif部の条件の成立するルールを探し、そのルールのthen部を実行することを繰り返す。二次記憶上のルール・ベースの中からルールを選び出す場合、真にif部の条件が成立するかを判定するためには検索の度ごとにすべてのルールを各々解釈し、条件式を評価することが必要であり、高速化は望めない。そこで二次記憶上のルール・ベースの検索はif条件の成立する可能性があるかないかによって行うことにする。検索結果得られたif条件の成立する可能性があるルールについてその各々を真にif条件が成立するかをアプリケーションが判定すればよい。

条件が成立する可能性があるルールとはなにかについて考えてみる。

if-then型知識を登録したルール・ベースをプロダクション・システムが検索するのは、プロダクション・システムが起動され、初期の問題状態(Problem State)が与えられた時、または、ルールの適用によって、新しい問題状態になった時のいずれかである。

問題状態の表現方法として、次のような方法がある。

① 広域変数の値による方法
② 事実をアトムのリストとして表現し、問題状態を既知の事実のリストを集めたリストとして表現する方法

if-then型知識のS式による表現にはいろいろなものがある。例えば、ここでは次のように表現こととする。

```
(RULE     <rule-id>
          IF      <起動条件>
          THEN    <行動>)
```

問題状態の表現として①の方法を使う場合、<起動条件>は、広域変数を含む命題として表現される。

```
(RULE     空調1
          IF      (真夏日か? 気温)
          THEN    (SET 冷房 5) )
```

この場合、if条件の成立する可能性のあるルールの検索は、広域変数の値が書き換わった時に、その変数を含む<起動条件>を持つルールを探すことである。この検索は、変数名という値の指定と『位置の弱い指定』による検索である。

一方、②の方法を使う場合、一つの事実を一つのリストとして表現し、AND条件で結合される事実を書き並べて<起動条件>とする。

```
(RULE  判別3
       IF     ((ITS COLOR IS WHITE)
               (IT IS IN HARD CASE))
       THEN (check  rule5) )
```

この場合、if条件の成立する可能性のあるルールとは、新たに既知となった事実を<起動条件>に含むルールである。この検索は、新事実を表すS式が<起動条件>をしめすS式の部分木になっているルールを見つけることである。

真にif条件が成立するルールが複数存在する場合に、その中から一つのルールを選ぶ戦略の記述はアプリケーション側に委ねられている。

2.5. 述語

二次記憶上に多量にあるファクトを検索したい。この検索はゴールとファクトのunificationによって行うことができる。しかし、厳密なunificationによる検索は計算コストが大きい。ここで、ゴールには変数を含んでもよいが、二次記憶上のファクトには変数は含まれない、という制約を加えても実用上問題はないと思われる。この制約のもとで、二次記憶上のファクトの検索をunificationによる検索ではなく、unificationの可能性による検索によって行うことにする。検索結果として返されたファクトに対

して本当にunificationができるかの判定は、アプリケーション側で行うことにする。

unification可能性をどのようにしたら容易に判定できるかを考察する。項をS式によって表現すると、値(atomまたはvariableまたはnil)を持つノードが、木構造の終端ノードに限られる。このことに着目すると、S式とS式のunificationにおいては、各々のS式において比較すべきノードを簡単に選び出せることが必要であることがわかる。では、どのようなノードが比較すべきものなのかを次の例で考える。

ゴール: L1
(EAT *AGENT *OBJ)
ファクト: L2
(EAT MAN (肉 野菜))

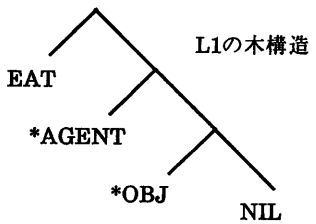

図2 L1の木構造

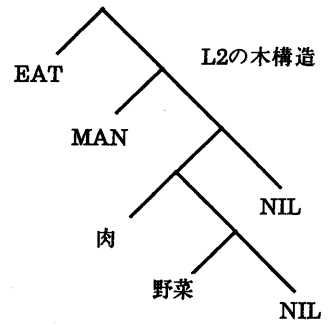

図3 L2の木構造

ゴール中の各アトムについての判断方法を述べる。L1のCARの位置のEATとL2のEATや、CADRの位置の*AGENTとMANのように、L1とL2のそれぞれS式上の対応する位置に終端ノードがある場合は、それらノードを比較すればunificationの可能性を判断できる。

また、ゴール中の各変数について判断する方法は次の通りである。L1上の変数*OBJに対応するL2上の位置には終端ノードはない。しかし、対応する位置の子孫には終端ノードがあるので、unification可能性があるとわかる。もし、L2上で対応する位置の祖先に終端ノードがあったなら、直ちにunification不可能とわかる。

このように、ゴールのアトムの値と位置情報から、『位置の強い指定』と値の指定による検索条件が得られる。また、ゴールの変数ノードの位置情報より、ファクトのS式上の対応する位置をrootにする部分木のノードを検索するという条件が得られる。ゴールと変数を含まないファクトとのunificationは、この二つの条件を同時に満足するS式の検索に対応する。

2.6. パターン・マッチング

パターン・マッチングとは、例えば、
(A *V B)というパターンが与えられたときに、CARにはアトムのA、CADRにはアトムのBが来るようなS式を探すことである。これは、あきらかに、『位置の強い指定』と値の指定による検索である。

また、*Vにはリストが来なければならないと、条件が加わることもある。

パターン・マッチングによる検索は、『位置の強い指定』であり、かつ、値の指定による検索と、マッチング変数と対応する位置に出現するS式の構造を指定する『構造による検索』という二つの検索の組み合わせに変換することができる。

2.7 検索要求のまとめ

検索要求のプリミティブをサポートするようにDBMSのインターフェースを設計すれば、S式による知識表現を一つのDBMSによって統一的に管理することができる。

S式によって記述された知識表現の集合に対する検索はどのような操作なのか、S式のどの部分を取り出すのか、また、検索条件とその条件の指定の組み合わせにはどのような性質があるか検討してきたことを要約する。

S式の集合に対する検索条件の指定には、次のものがあることがわかった。

- 値の指定
- 位置の指定

位置の指定には、一意に決まる『強い指定』と、ある条件で示される『弱い指定』がある。これらの条件は、単独で、または、組み合わせて指定される。組み合わせかたを表1にまとめた。

値の指定と位置の指定が、検索要求のプリミティブである。位置の指定による検索、値の指定による検索のほかに、

- S式の構造の指定による検索
- 部分木であるかどうかによる検索
- unification可能性による検索

が、あった。しかし、これらはすべて値の指定と位置の指定の組み合わせによって検索要求を記述することができる。

位置	値の指定	
	なし	あり
弱	部分木	スロット ファセット
強	構造の指定	LISP関数 リンクタイプ

表1 検索要求のプリミティブの組み合わせ

例えば、構造による検索は、終端ノードの値を指定せずに、終端ノードのあるべき位置を指定して検索することで実現できる。

S式の検索条件のプリミティブは以上の通りである。ところで、検索条件を満足するS式のうち取り出す部分は、

- S式全体
- S式の部分木

の場合がある。DMLによって取り出す部分を指定する必要がある。

また、ある検索の結果を利用して次の検索を行うこと(以下、副次検索という)が必要な場合もある。

したがって、これらのすべての検索の方法を記述できるようにDMLの構文を決めなければならない。

3. S式の関係DBモデルによる表現

以上の考察より、S式の集合を検索する時には、値の指定による検索と同様に、位置の指定による検索やS式のデータ構造の指定による検索が、実現できなければならないことがわかった。

終端ノードごとに位置情報をつけ、それを検索のキーにすれば、位置の指定による検索を実現することができる。

前章の考察より、この位置情報の表現方法の備えるべき性質として、次のことをあげることができる。

① ノードの親子関係が容易に判断できること
② S式上の、あるノードからの相対位置を表現できること
③ 部分木を構成するノードが他のノードから簡単に区別できること。

3.1 S式データの変換の問題点

S式を関係データベースのタプルにどのようにして対応させるかは重要な問題である。

もし、一つのS式の印字表現を文字列として一つのタプルに対応させる方法を採用すると、「位置の弱い指定」による検索(SELECT操作)をする場合、各タプルごとに、属性値である文字列と、検索キーである文字列化したアトムとのストリング・マッチをしなければならない。

また、検索のキーにする終端ノード各々と、属性を対応させる方法は、一つの関係に登録されるS式がすべて同じ木構造をしていなければならないという制約が生じ、柔軟な構造なS式に対応できないことがある。

S式においては、アトムやストリングは終端ノードにのみ存在することに着目し、終端ノードの位置情報と、ノードの値の組によって、もとのS式と等価な情報を表現する方法を導入する。その表現を変換の要として、S式を関係データベースのタプルに対応させる方法を提案する。

3.2. Positional Notationの導入

各終端ノードのS式中での位置情報を下の図のように表現する。木の左方向へ進むことに"0"を、また、右方向に進むことに"1"を対応させる。更に、右または左へ進むことは、それぞれ"0"または"1"を書き連ねて表す。

((E1 E2)(E3.E4))…… [1]

式[1]の木構造を例に説明する。

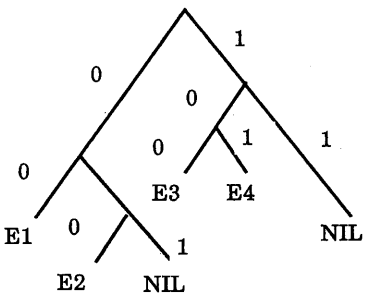

図4 式[1]の木構造

S式のrootから終端ノードへのパスを表すビット列と、終端ノードの値であるアトムまたはストリングからなる組の集合は、

(("00" E1)("010" E2)("011" NIL)("100" E3)
("101" E4)("111" NIL)("1011" NIL)) …… [2]

と、なる。ここで、リスト終端を示しているNILの組ではビット列のLSBが必ず"1"である。この組がなくても他の組の存在から、その位置にリストの終わりを示すNILが来ることが明らかなので省略する。ただし、リストの終わりがドット・ペアの場合は省略できない。その結果、式[3]が得られる。

(("00" E1)("010" E2)("100" E3)("101" E4))…… [3]

木構造に着目してS式の終端ノードの、rootからの位置を示すビット列とそのノードの値の組の集合によって、S式と等価な情報を表現することができる。このような記法をPositional Notation(以下PN)とよぶ。
式[3]は、式[1]のPNである。また、ノードへのパスを表すビット列のことを"PNビット列"と呼ぶこととする。

3.3. 次数4の関係による表現

3.2で導入したPNの組(<PNビット列> <終端ノード>)を、関係データベースのタプルに対応させる。

TID	SID	PNBSTR	NDVAL
1	1	"00"	E1
2	1	"010"	E2
3	1	"100"	E3
4	1	"101"	E4

表2 式[1]の関係による表現

一般には、関係データベースの一つのタプルは、一つの独立したオブジェクトを表現する。しかし、ここで用いられる方法では、一つの独立したオブジェクトであるS式を複数のタプルによって表している。タプルどうしが一つのS式を構成する要素であることを示すのに、S式のID(SID)をタプルに付加する必要がある。[2]

関係には、TID(タプルID)、SID(S式のID)、PNBSTR (Positional Notation Bit String) 、NDVAL(ノードの値)の4つの属性がある。PNでの各々の組が、この関係ではタプルに対応している。TIDは関係の中でユニークなキーである。SIDは重複を許すキーで、同じS式を構成しているタプルには同じキーが割当てられている。PNBSTRにはPNビット列が、NDVALにはその終端ノードの値がS式上のデータ型を表すフラグと共にそれぞれ入っている。

NDVALのフラグによって、そのノードがもとのS式では文字アトム、数字アトム、ストリングのいずれだったかを判別できるようにしている。(表2ではフラグを省略している。)

3.4. PNによる位置情報表現の性質

位置情報の表現方法にもとめられる性質は3.において述べた。PNによる表現ではこれらの求められている性質は満足しているかを考察する。

あるノードの親の位置は、PNビット列のLSBを1ビット落とすことで得られる。

相対位置は文字列連結によって得ることができる。例えば、rootからの、ある非終端ノードへの位置をあらわすPNビット列と、そのノードから子孫への木のたどりかたをあらわすPNビット列の文字列連結によってrootからみた子孫ノードのPNビット列を得る。

部分木の構成要素の検索は、その部分木のrootの位置を示すPNビット列とタプルのPNビット列の前方一致によって実行できる。

このように、PNビット列によって位置情報を表すと、位置に関する必要な操作は、ビット列の連結や抽出に写像される。PNは位置情報の表現において必要な性質を満足している。

4. システムの概略

アプリケーションからの検索要求はLB:QUERY関数によってLISP BASEへ渡される。LISP BASEは、これを解釈して次数4の表に対する操作を行う。表から得られたタプルは、いったんPositional Notationにされたのち、S式の集合としてアプリケーション・プログラムへ渡される。

LISP BASEへの知識の登録・追加を行う場合は、S式をPostional Notationを経て、次数4の表のタプルに変換して格納する。この表への操作は、SQLを基に作られた検索言語IQL/Sによっておこなう。

アプリケーション・プログラムとLISP BASEとのやり取りはどのようにして行うかについて概略を説明する。

4.1 LISP BASEのデータ定義

LB:DEFSCHEMA関数はLISP BASEシステムでのDDLである。使用する知識表現言語は、S式で記述した時にどのようなデータ構造をしているのか、S式のどの位置にシンボルがあるのかを定義する。さらに、PNビット列をユーザが直接記述することはを間違いを起こし易いので、ビット列の代名詞として位置を示す名前をつける。この代名詞をPNタグと呼ぶ。この関数の実行によって、S式上の位置をPNタグによって指定することができようになる。

フレームの場合は次のように記述する。(構文は付録を参照)

```
(LB:DEFSCHEMA 'FRL
      '(frame名 *
        (slot名 * (facet名 data-list))))...[4]
```

この式でアスタリスクは、その印の次に現れる構造が繰り返すことを示している。PNタグによってノードの位置を表すことで、タグがついたS式の部分を、一つの属性のように扱うことができるようになる。この定義によってフレームについての検索は、あたかも、

frame名	slot名	facet名	data-list

という関係に対する検索のように記述できる。

また、Lisp関数の定義は式[5]のように記述する。

(LB:DEFSCHEMA 'LISP
'(F-DEF-F FUN-NAME ARGS BODY))..[5]

この定義によってLisp関数に対する検索は、あたかも、

F-DEF-F	FUN-NAME	ARGS	BODY

という関係に対する検索のように記述できる。

4.2 LISP BASEのデータ操作

LISP BASEのデータ操作言語(DML)であるQL/S(Query Language in S-expression)は、SQLが持つSELECT-FROM-WHENの構文を基本にしている。(付録を参照)

副次検索を記述する必要があるため、DMLの構文を決めるのと同時に、検索結果の返し方も定めなければならない。

LISP BASEから返される検索結果はNILまたはS式の集合である。
条件を満たすS式が存在しない場合、検索結果としてNILが返される。
条件を満たすものが存在する場合で、かつ、<検索対象>がnilつまり条件を満足するS式全体を取り出す場合は検索結果はS式の集合として渡される。一般形を式[6]に示す。

(S式1 S式2 S式3 . .). ... [6]

条件を満たすものが存在する場合で、かつ、<検索対象>が指定され、条件を満足するS式のある部分を取り出す場合の検索結果は式[7]に示すS式の集合として渡される。

((S式1の部分1 S式1の部分2 ・・)
(S式2の部分1 S式2の部分2 ・・・) ・).... [7]

式[7]の特別な場合として、((NIL))は、検索の結果としてNILという値が得られたことを示す。PROLOGの内部データ・ベースでは、検索の結果としてNILという値が得られると、assertionがないことなのか、値としてNILを持っているのか、その区別はできない。しかし、LISP BASEではこの区別が可能である。

副次検索を記述するには、<検索対象>中に<LB:QUERY関数>を入れ子にし、検索結果であるS式の集合を、検索条件のLB:LAMBDA式にAPPLYすればよい。lambda変数によって検索結果の受け渡しができるよう構文が決められている。

検索式の記述例を挙げる。フレームにおいてframe名がtetsuo、slot名がheight、facet名がvalueであるようなdata-listの検索は、

(LB:QUERY '(data-list)
'(AND (frame名 'tetsuo) (slot名 'height)
(facet名 'value)) 'FRL) ...[8]

と記述できる。

また、関数名がFACTであるようなLisp関数の関数定義式を取り出す検索は、

(LB:QUERY NIL
'(FUN-NAME 'FACT) 'LISP)) ...[9]

と記述できる。

4.3 データ構造の表現方法

<構造表現リスト>によって、データ構造によるアクセスが可能になった。データ構造によるアクセスとは、S式のある指定された位置に、ある特定のデータ構造が格納されているかどうかによって検索を行うことである。

構造のrootから各終端アトムまでの位置を表すPNビット列の集合によってデータ構造をユニークに表現することができる。このリストを構造表現リストと呼ぶこととする。ビット列の集合はリスト中でソートされているため、同一のデータ構造は同一の構造表現リストによって表現される。

さらに、例で説明する。

【例1】 (A B C)のような長さ3のリスト.....("0" "10" "110")

【例2】 ((A B) (D E))
("00" "010" "100" "1010")

データ構造中の各要素の実際の位置を表すPNビット列は、S式のrootから見た構造のrootの位置を表すPNビット列と、構造表現リストの各要素ビット列の連結によって求めることができる。

利用者が"構造表現リスト"を定義するのに都合がよいようにLB:DEFSTRUCT関数が用意されている。例1の場合なら、

(LB:DEFSTRUCT '*LIST3* '(A B C))

とすれば、*LIST3*に長さ3のリストの構造表現リストが束縛される。

頻繁に使用するデータ構造であるリストについては、リスト長1から10までをそれぞれ*LIST1*から*LIST10*として、また、有限任意長のリストを*LISTN*としてシステムがあらかじめ定義している。

5. 操作の変換

4.で述べたように、ユーザー・プログラムからLISP BASEへの検索、登録、更新などの操作は、LB:DEFSCHEMA関数で定義した見かけ上の関係に対しての操作として、QL/Sによって記述される。

ところが、LISP BASE内部においては、S式はそのデータ構造に依存せず、次数4の関係によって統一的に表現されている。

この次数4の関係への操作を記述するものがIQL/S (Intermediate QL/S)である。LB:QUERY等のQL/Sによる操作は解釈され、システム内部では、IQL/Sとして実行される。

式[8]で示した検索式は図5のように実行される。

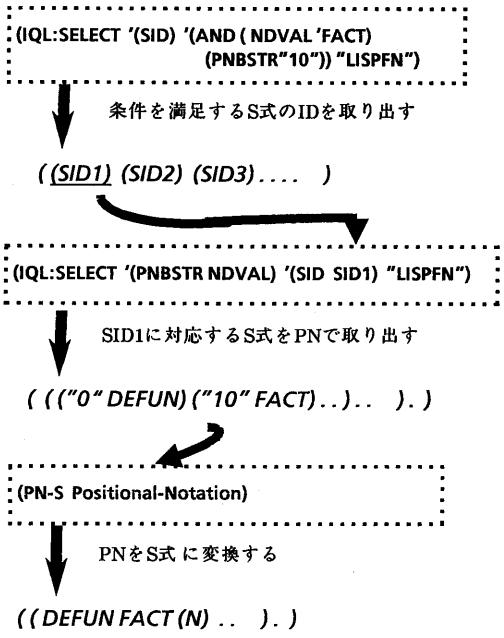

図5 式[8]の解釈と実行の様子

フレームの例にあった『位置の弱い指定』がPNビット列によってどのように表されるかを考える。スロット名の出現可能なノードの位置は図6の■印のところである。

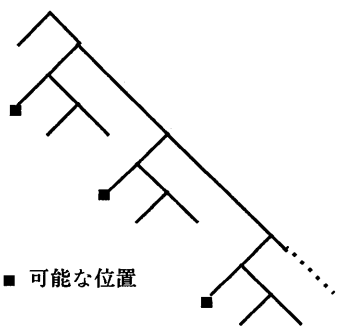

図6 スロット名の出現可能位置

出現可能なノードの位置のPNビット列は、

"100" "1100" "11100" "111100".....

である。そのビット列のなかに共通なビット列とN回の"1"を繰り返しから出来ていることがわかる。『位置の弱い指定』による検索はビット列の部分一致によって実現できる。

6. まとめ

Positional NotationをS式と関係DBのタプルとの変換の要として導入したことにより、この二つを対応させることができ、S式で書かれている知識やデータをそのままDBに格納することができることになった。LISP BASEでは、値による検索、位置による検索、または、それらの組み合わせによるデータ構造による検索などが可能である。S式は柔軟なデータ構造をとりうるが、Positional Notationを導入したおかげで、次数4の単一な構造の関係として扱うことができる。S式によって記述された知識への検索、追加、書き換え、消去などの操作はすべてこの関係への操作に写像され、実行される。

S式からPositional Notationへの変換や逆変換、検索式の解釈にかかるCPU時間は一つの検索につき数百ミリ秒(ALPHAのインタープリタを使用)である。S式とPNとの変換はほとんどシステムの負荷となっていないことが既に確かめられている。

今後の課題として、本システムでは、フレーム、ルール・ベース、ユーザー定義関数などを統一的に扱うことができるので、エキスパートシステム構築ツールと融合させて本システムの有用性を示したい。また、複数のユーザからの知識の更新要求によっても知識の一貫性を保つことのできるロック機構の実現方法について考察する。

参考文献

[1] R. Lorie, W. Kim, D. McNabb, W. Plouffe, and A. Meier: "Supporting Complex Object in a Relational System for Engineering Databases", Query Processing in Database System, Springer-Verlag, pp145-155

[2] Y. Vassilliou, J. Clifford, and M.Jarke: "Data Access Requirements of Knowledge-Based System", Query Processing in Database System, Springer-Verlag, pp156-170

付録

(1) LB:DEFSCHEMAの構文

(LB:DEFSCHEMA
　　<スキーマ名> <定義リスト>)
<スキーマ名>　::= 文字アトム
<定義リスト>
　::=PNタグ
　| (<定義リスト> . <定義リスト>)

(2)QL/Sの構文

● LB:QUERY関数

LISP BASEへの検索を行う。

<LB:QUERY関数> ::=
(LB:QUERY <検索対象> <検索条件式>
<スキーマ名>[<検索対象のWorld名>])
<検索対象> ::= NIL | < list >
nilの場合…. <検索条件式>の条件に合うS式をそのまま返す。
listの場合…. 検索の結果として欲しいもののPNタグのリスト
<検索条件式>
::= φ
| ([AND| OR]<条件> . <検索条件式>)
| (LB:LAMBDA <lambda変数>
<検索条件式> <LB:QUERY関数>)
<条件> ::=(PNタグ [<値>| <構造表現リスト>])
<構造表現リスト> 4.3で述べる。
<検索対象のWorld名> ::= <ストリング>
どのWorldについて検索を行うかを指定できる。
指定しないとDefault値のWorldについて検索を行う。

● LB:ADD関数

LISP BASEへの知識の追加をする。完全に同一のS式がない場合、追加を行う。

(LB:ADD <S式> [<World名>])

● LB:UPDATE関数

LISP BASE中の知識から<検索条件>に合うS式を選び出し、そのようなS式について更新操作を行う。

(LB:UPDATE <更新操作リスト> <更新条件式>
<スキーマ名> [<World名>])

<更新操作リスト>
::=φ
| (<操作> . <更新操作リスト>)
<操作>::=(PNタグ <値>)
<検索条件式>
::=φ
| ([AND| OR]
<条件> . <検索条件式>)
| (LB:LAMBDA<lambda変数>
<検索条件式>
<LB:QUERY関数>)
<条件>
::=(PNタグ
[<値>| <構造表現リスト>])
<World名> ::= 文字アトム
指定しないとDefault値のWorldに対して更新を行う。

● LB:DELETE

LISP BASE中の知識から<検索条件>に合うS式を選び出し、そのようなS式を消去する。

(LB:DELETE <更新条件式>
<スキーマ名> [<World名>])

● LB:CREATE関数

LISP BASEのWORLD(DBでの関係に相当)を創成する。

(LB:CREATE <World名>)

● LB:KILL関数

LISP BASEのWORLDを消去する。

(LB:KILL <World名>)

(3) IQL/Sの構文

IQLの構文はSQLをもとに作られている。操作内容はSQLのものと対応している。

● IQL:SELECT関数

(IQL:SELECT <抽出する属性名のリスト>
<検索条件式> [<表の名前>])
<属性名> ::= TID|SID|PNBSTR| NDVAL
<表の名前> ::=<ストリング>
<検索条件式>
::= φ
| ([AND| OR]
<条件> . <検索条件式>)
| (LAMBDA <lambda変数>
<検索条件式><IQL:SELECT関数>)
<条件> ::= (<属性名> <値>)

● IQL:UPDATE関数

(IQL:UPDATE <更新操作リスト>
<表の名前> <検索条件式>)
<更新操作リスト>
::=(<代入操作>.<更新操作リスト>)
<代入操作> ::= (<属性名> <値>)
<検索条件式> ….IQL:SELECTに同じ

● IQL:ADD関数

(IQL:ADD <値リスト> <表の名前>)
<値リスト>::=
(TIDの値 SIDの値 PNBSTRの値 NDVALの値)

● IQL:DELETE関数

(IQL:DELETE <表の名前> <検索条件式>)