自然言語処理 81－6
（1991. 1. 17）

日本語データベース処理システムの試作

パーソナル・コンピュータ上での実現

笠　晃一　　小林修二　　横田将生
（㈱日本データベースネットワーク研究所）　（福岡工業大学）

ＳＱＬなどの形式的な問合せ言語を用いてデータベースにアクセスするのは、初心者にとって容易なことではないため、自然言語を用いてデータベースに問合せできるようなシステムが数多く開発されてきた。しかしながら、これらの自然言語問合せシステムは、特定のデータベースにしかアクセス可能でなかったり、受理可能な文に対する制限の緩和が容易でなかったり、あるいは、語彙辞書をユーザが作成しなければならなかったりして、本当に実用的なものとは言えなかった。我々は、独自に開発したＳＡＸ－Ｃトランスレータと呼ばれるツール等を用いて、これらの問題を解決し、パーソナル・コンピュータ上で動作する実用的なシステムを試作した。本稿では、ＳＡＸ－Ｃトランスレータおよびシステムについて報告する。

An Experimental Database Processing System with a Japanese Interface

Implemention on a Personal Computer

Kouichi Ryu　Shuji Kobayashi　Masao Yokota*
Japan Database Network Laboratory Co., Ltd.
*Fukuoka Institute of Technology

It is not easy for elementary users to have access to a database by a formal query language such as SQL. There have been developed a lot of systems that allow such users access to a database by a natural language. Most of them were only for a special database, and, even if for general use, it was not practical because it could not extend the range of acceptable sentences or it forced users to make a lexical dictionary. We have recently developed a tool called SAX-C translator which make it easy to solve these problems and constructed an experimental system on a personal computer using the tool. This paper describes the outlines of SAX-C translator and the system.

1．はじめに

関係データベース（以下で単にデータベースといえば関係データベースを指すものとする）にアクセスするには、ＳＱＬ[1]などの形式的問合せ言語を利用するのが一般的であるが、形式的問合せ言語は抽象的かつ複雑であるので、修得するのに時間がかかるし、また、仮に修得できたとしても、検索内容によっては、かなり複雑な問合せ文になる可能性がある（図１参照）。このため、自然言語によってデータベースにアクセスできるようなシステムが、いくつも考えられてきた[2]-[4]。しかしながら、今までのシステムは、次のような点に問題があり、実用に耐えられるようなものはなかった。

①受理可能な文に対する制限の緩和が容易でない

実用的なレベルの自然言語問合せシステムを作ろうとすると、少なくとも１０００ルールくらいの構文規則が必要と思われる。しかしながら、今までのシステムの中には、実用的な構文解析手法について考慮したと思われるものはなく、したがって、受理可能な文の範囲が狭いときには、うまく動いていたシステムでも、受理可能な文の範囲を広くしたとたんに、解析時間がかかりすぎたり、作業領域が足りなくなったりするようなシステムがほとんどであった。

②構文解析用の辞書をユーザが作成する必要がある。

「ヤチマタ」[2]のような例外はあるが、いままでの自然言語問合せシステムは、特定のデータベースを対象に開発されたものが多かった。しかしこのようなやり方だと、データベースごとに一つのシステムが必要になり、あまり実用的ではない。したがって、データベースの内容からは独立したシステムを作成する必要があるが、「ヤチマタ」を含め、そのようなシステムでも、使用される名詞や動詞はデータベースに依存するので、ユーザがシステムに教えてやる必要があった。

③漢字かな混じりの日本語べた書き文が扱えるものが少ない。

日本語問合せシステムに限って言えば、実験的なシステムが多いせいもあって、漢字かな混じり文を受けつけるようなものは少ない。また、かなり実用性を意識して作られたシステムでも、漢字かな混じり文は受けつけるものの、分かち書きをしなければならないようなシステムがほとんどである。このような、漢字かな混じり文を使えなかったり、分かち書きをしなければならなかったりするような制約は、ユーザにとっての負担が大きく、実用的でない。

本システムでは、ＤＣＧ規則をＳＡＸアルゴリズムのＣプログラムへと変換するＳＡＸ－Ｃトランスレータと呼ぶツールを開発することにより、構文・意味解析が高速に、しかも、少ない作業領域で行なえるようになった。このことにより、受理可能な文の種類を豊富にすることができ、ひいては、流暢な日本語で質問することができるようになった。また、名詞類とその意味をデータベースから直接学習させ、動詞類の意味を入力文から学習させることにより、問合せ文の中で使用する単語を、ユーザが登録しなくてもよいようにした。さらに、形態素解析を工夫することにより、漢字かな混じりのべた書き文を受けつけるようにした。

2．ＳＡＸ－Ｃトランスレータ

2．1．処理の組合せ的爆発とその抑制

従来の自然言語問合せシステムは、受理可能な文の範囲を広げようとすると、極端に処理時間がかかったり、

納入業者

業者番号	業者名	所在地
S1 …	大崎商会 …	福岡 …

部品

部品番号	部品名	色
P1 …	ボルト …	青 …

納入表

業者番号	部品番号	個数
S1 …	P2 …	2000 …

日本語問合せ文：

すべての部品を納入している業者は？

ＳＱＬ問合せ文：

```
select 業者名
from 納入業者
where not exists
  (select *
   from 部品
   where not exists
     (select *
      from 納入表
      where 業者番号＝納入業者．業者番号
            and 部品番号＝部品．部品番号))
```

図１．自然言語検索文と対応するＳＱＬ文の例

作業領域が不足して文の処理ができなかったりするものが、ほとんどであった。これは、受理可能な文の種類を増やすと、形態素解析や構文解析の段階での曖昧性が増大し、これが処理の組合せ的爆発を引き起こしたためと考えられる。この組合せ的爆発を防ぐには、形態素解析、構文解析、意味解析を同時並行的に行ない、より上位の情報を用いて、早い段階で曖昧性の解消をはかるしかないと思われる。ＤＣＧ[5]と呼ばれる記法を用いれば、構文規則と意味規則をまとめて記述することができるので、この処理系を作れば、構文解析と意味解析を同時に行なうことができるようになる。形態素解析と構文解析の間のインタフェースをうまく作れば、この二つも同時並行的に行なうことができるから、ＤＣＧ記法によって、処理の組合せ的爆発をある程度まで抑えることができるのが分かる。

このＤＣＧを実行可能なプログラムに変換するものとして、ＤＣＧトランスレータ、ＢＵＰトランスレータ[6]、あるいは、ＳＡＸトランスレータ[7]などが知られている。このうち、ＤＣＧトランスレータは、左再帰規則を使えない、得られる実行可能プログラムの実行速度が遅いなどの欠点を持ち、実用的でない。ＢＵＰトランスレータは左再帰規則が使えるし、得られるプログラムの実行速度もＤＣＧトランスレータによるものと比べて、かなり速い。ただし、実行可能プログラムは大量にメモリを消費する。ＳＡＸトランスレータは左再帰規則が使えるし、得られるプログラムの実行速度も、ＢＵＰトランスレータによるものと比較しても、さらに３０～３００倍程度は速いし、メモリ消費量も少ない。

このように、上記三つのうちではＳＡＸトランスレータがもっとも優れていると考えられるが、このトランスレータもＰｒｏｌｏｇプログラムを生成するので、この点だけは問題である。Ｐｒｏｌｏｇは、Ｃなどの言語に比べて、処理速度やメモリ効率が、かなり落ちるからである。そこで、我々は、ＳＡＸアルゴリズムが後戻りを行なわないことに着目し、Ｃのプログラムを生成するようなトランスレータを作成してみた。

３．２．ＳＡＸアルゴリズムのＣプログラム

ＤＣＧからＳＡＸアルゴリズムのＣプログラムを生成するトランスレータを、ここではＳＡＸ－Ｃトランスレータと呼んでいる。このトランスレータの生成するＣプログラムについて簡単に説明する。まず、文脈自由文法で記述された次のような構文規則があったとする。

```
s --> np, id1 vp.
np --> det, id2 noun.
np --> det, id3 noun, id4 relc.
relc --> comp, id5 vp.
vp --> verb.
vp --> verb, id6 np.
```

ここに、ｉｄ１とかｉｄ２などは識別子と呼ばれるもので、処理がどこまで進んだかを表している。この識別子に関して、カテゴリの処理を二つのタイプに分ける。一つは、構文規則の左辺の最も左にあるカテゴリの処理で、これはタイプ１の処理と呼ばれる。この種の処理は、原則として識別子をはき出すようなものになる。これ以外のところにあるカテゴリの処理は、タイプ２である。この種の処理は、原則として識別子を受け取って別の識別子をはき出すようなものになる。たとえば、カテゴリｄｅｔは、タイプ１の処理だけしかなく、識別子ｉｄ２またはｉｄ３をはき出すので、次のような処理をすればよい。

```
void det(x)
stream x;
{
        C(2,x);
        C(3,x);
}
```

ただし、関数Ｃ（ｎ，ｘ）は、ストリームｘを下位ストリームとして、識別子ｎを現ストリームに加えるような働きをする。次に、カテゴリｎｏｕｎを考えてみると、これはタイプ２の処理だけしかない。最初のものは、識別子ｉｄ２を受け取って、カテゴリｎｐの処理を呼ぶようなものであり、もう一つは、識別子ｉｄ３を受け取って、別の識別子ｉｄ４をはき出すようなものである。したがって、カテゴリｎｏｕｎの処理を記述すると次のようになる。

```
void noun(x)
stream x;
{
        do {
                switch (x->id) {
                case 2:
                        np(x->ss);
                        break;
                case 3:
                        C(4,x->ss);
                        break;
                }
        } while ((x = x->next) != NULL);
}
```

ここに、ｘ－＞ｉｄ、ｘ－＞ｓｓ、ｘ－＞ｎｅｘｔはそれぞれ、ストリームｘの最初の要素における識別子、下位ストリーム、および、次の要素を表している。

３．３．解析速度の比較実験

ＳＡＸ－Ｃの有効性を確認するために、ＳＡＸトランスレータとＳＡＸ－Ｃトランスレータの生成するプログラムについて、その解析速度を比較した。作業領域についての比較は、今回は行なわなかった。比較実験は、Ｍｉｃｒｏ　ＶＡＸ　Ⅱ上で行ない、ＳＡＸトランスレータの生成するＰｒｏｌｏｇプログラムに対しては、Ｑｕｉｎｔｕｓ　Ｐｒｏｌｏｇコンパイラを、ＳＡＸ－

番号	語数	解釈数	処理時間		速度比
			SAX	SAX−C	
1	6	1	130	12.2	10.7
2	10	1	410	41.3	9.9
3	12	5	1110	116.3	9.5
4	18	20	2360	254.9	9.3

（処理時間の単位：msec）

図2．　SAXとSAX−Cの比較

Cトランスレータの生成するCプログラムに対しては、VAX　Cコンパイラを、それぞれ用いた。また、実験に用いた構文規則は日本語に対するもので、DCGにして50ルール程度の簡単なものである。実験結果を図2に示すが、SAX−Cトランスレータによって生成されたCプログラムは、SAXトランスレータによって生成されたPrologプログラムに比べて、約10倍の解析速度を持っていることが分かる。

2．システムの全体構成

試作したシステムは、図3に示すような構成になっている。構文解析と意味解析を同時に行なっているため、処理の組合せ的爆発を抑えることができ、したがって、処理時間を短く、作業領域を少なくすることができる。

①形態素解析部

日本語問合せ文として許されるのは、漢字かな混じりのべた書き文であるため、普通にワード・プロセッサを使っているような感覚で入力できる。また、ヒストリ機能を持っているので、以前に入力した文をいつでも呼び出すことができ、タイピングの手間を最小限に抑えることができる。形態素解析部では、この入力された日本語問合せ文を最長一致法を用いて単語に分解し、各単語の構文的・意味的情報を次の構文・意味解析部に引き渡している。解析用辞書の構造として、TRIE構造[8]に改良を加えたS−TRIE（Sorted　TRIE）と呼ばれるものを使っている。

②構文・意味解析部

構文・意味規則の記述にはDCG記法を用いている。このため、構文解析と意味解析は同時並行的に行なわれる。したがって、構文解析木を経ずにいきなり意味表現を生成でき、この点も、処理時間の短縮と作業領域の縮小に寄与している。

ここで使われている意味表現は、SRT（SQL Representation Tree）と呼ばれるものである。SRTというのは基本的には、形式的問合せ言語SQLと同じものであるが、多少の拡張を施してあり、また、生成や解釈がしやすいように木構造の形をしている。ただし、意味解析部によって直接作り出されるものは、深層SRTと呼ばれるもので、これは、SQLを圧縮したような構造をしている。これを、意味トランスレータによって、表層SRT（あるいは単にSRT）に変換している。

③エコーバック生成部

ここでは、構文・意味解析部で作られたSRTを曖昧性のない日本語になおして表示する。エコーバック生成部の本来の働きは、問合せ文がユーザの意図通りに解釈されたかどうかを、ユーザに確認してもらうというものである。理想的なシステムであれば、問合せ文はユーザの意図通りに解釈されるはずであるが、現在の技術レベルにおいては、どうしても微妙な点で解釈の食い違いが出てくるので、エコーバック生成部を省略することはできない。エコーバック生成部のもう一つの働きは、構文的曖昧性の解消である。構文的曖昧性が生じたとき、係り受け関係を示して、ユーザに正しい方を選択させるというやり方もあるが、多少とも文法知識が必要になり、あまり現実的ではない。ここでは、生成された複数個のSRTをそれぞれ曖昧性のない日本語で表示して、その中から正しいものを選ばせるというやり方をとっている。

④SRTインタープリタ部

データベース管理部が扱えるのは、関係データベースの関係表が一つに限定された形で、いわゆるカード型データベースといわれるものである。したがって、データベース管理部が受けつけるSRTも単純な条件式を持つものに限られ、たとえば、異なる条件で二度以上検索しなければならないものなどは、受けつけることができない。したがって、ここでは、SRTを単純な条件式に分解して、データベース管理部へ送っている。検索結果を保持する機能を持っているので、一

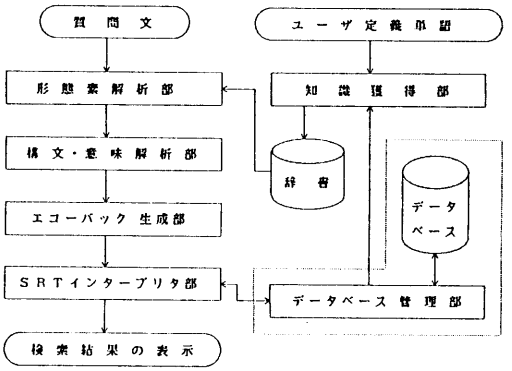

図3．システムの全体構成

度検索して、その結果を使って再び検索しなければならないような場合にも対処できる。

⑤知識獲得部

問合せ文の中で使われる名詞のうち、属性名と属性値の情報は自動的に対応するデータベースから取り込まれるので、ユーザが入力してやる必要はない。また、問合せ文の中で使われる動詞のうち、「住む」とか「生まれる」のような属性名に関連のある動詞は、学習によって自動的に属性名と関連づけられる。したがって、システムに何も教えないでも、最初からかなりの範囲の文を使用することができる。なお、システムに同義語やキー・フィールドなどを教えたい場合を考えて、問合せシステムをユーザが教育することもできるようになっている。たとえば、「少年」を「１５才以下の男性」のように定義しておけば、「福岡に住んでいる１５才以下の男性は誰か」と質問する代わりに、「福岡に住んでいる少年は誰か」と聞くことができるし、「名前」フィールドと「所属」フィールドをキー・フィールドに指定しておけば、「３０才以下の人の趣味は何か」のように聞いたときでも、「趣味」フィールドだけでなく、「名前」フィールドと「所属」フィールドも表示してくれるようになる。

４．システムが受理可能な文

４．１．組集合と制約

現在、試作システムが受理可能なのは検索文に限られる。検索文の代表的な形は次のようなものである。

検索文　－－＞　組集合，　［は］．

ここに、［］は終端記号を表している。また、組集合というのは、制約を満たす組（あるいはカード）の集まりのことであるが、統語範疇の名前としてもこれを用いている。それは、そのような句が、組集合という実体に対応しているように考えられるからである。組集合は、次のような形をしている。

組集合　－－＞　制約，主題．
組集合　－－＞　制約，属性値．

たとえば、「趣味が読書である社員」は組集合であるが、このうち「趣味が読書である」の部分が制約、「社員」の部分が主題である。制約は検索条件を表す部分であり、主題は検索対象のデータベースが何についてのものかを表している部分と考えることができる。なお、「趣味が読書である山川さん」のように、主題の代わりに「山川さん」のような属性値が来ることもある。制約は、次の二つに大きく分けられる。

①属性値を述部に含むような制約

制約　－－＞
【属性名，［が］，】属性値述部．

②属性値を格助詞句に含むような制約

制約　－－＞　格助詞句$^{+}$，一般動詞；

制約の例としては、「年齢が２０才以上である」とか「福岡から熊本に旅行した」などが挙げられる。ここで、「年齢」が属性名、「２０才以上である」が属性値述部であり、また、「福岡から」と「熊本に」が格助詞句、「旅行した」が一般動詞である。

なお、格助詞句$^{+}$というのは、格助詞句が一つ以上並んだものを表す。また、上の規則で、【】は省略可能な部分を表している。したがって、制約に関する第一の規則は、属性名が助詞「が」とともに省略可能であることをいっているし、また、制約に関する第二の規則も属性名が省略されていると見なすことができる。このような省略された要素を持つ検索文[9]に対しては、その要素を推定してやる必要があるが、その方法については次節で述べる。

４．２．制約の分類

制約は、その意味によって、以下のように分類することができる。

①一つの属性を一つの属性値と比較するもの

「趣味が読書である」、「年齢が２０以上の」、「福岡に住んでいる」などは、ある一つの属性を一つの属性値と比較するもので、最も単純な制約と考えられる。なお、属性値と比較する代わりに、「残業手当が基本給より少ない」とか「ボーナスが給料の５倍に３０万たしたものより多い」のように、属性やそれを含んだ式と比較することもできる。

②一つの属性を複数の属性値と比較するもの

「趣味が読書とピアノである」、「年齢が２０以上３０以下の」、「福岡か熊本に住んでいる」などは、ある一つの属性を複数の属性値と比較している。条件は、連言的に結合される場合と選言的に結合される場合がある。

③一つの属性を組集合の属性と比較するもの

これは、ＳＱＬでいえばｓｅｌｅｃｔ文が入れ子になったもので、「山川さんの年齢よりも若い」とか「山川さんより若い」などがそうである。

④以上の３つが組み合わさったもの

①から③までを、連言的もしくは選言的に組み合わせ

て、「趣味が読書とピアノで、福岡に住んでいる」とか「山川さんより若いか、あるいは年齢が２０以上の」のような制約を作ることができる。

４．３．受理可能な文の種類

①カードを表示する

このときの検索文は、次のような形をしている。

検索文　－－＞　組集合，［は］．
検索文　－－＞　組集合，［を］，表示述語．

表示述語というのは、［表示しなさい］、［表示して下さい］とか［見せてくれ］のようなものである。この部分は、ＳＲＴに変換するときに何の意味も持たないので、単に無視されるだけである。

［例］福岡に住んでいる人を表示しなさい

②カードを並べ換えてから表示する

検索文　－－＞
　組集合，［を］，ソート指定，表示述語．

ここに、ソート指定としては、「若い順に」、「年齢の順に」、「年齢の若い順に」あるいは「年齢の順に並べ換えてから」など、いろいろなものが許される。

［例］福岡に住んでいる人を若い順に表示しなさい

③いくつかの属性を表の形で表示する

このときの検索文は、次のような形をしているものとする。

検索文　－－＞
　組集合，［の］，属性並び，［は］．
検索文　－－＞
　組集合，［の］，属性並び，［を］，表示述語．

ここに、属性並びというのは属性名もしくはこれを含む式の並びであって、「趣味」とか「ボーナスと給料の和および年齢の平均」のようなものを表している。

［例］趣味が読書である社員のボーナスと給料の和
　および年齢の平均は

④カードを並べ換えてから、いくつかの属性を表の形で表示する

検索文　－－＞
　組集合，［の］，属性並び，［を］，
　ソート指定，表示述語．

これは、②と③を合わせたような形をしている。

［例］年齢が２０才以上の社員の年齢と趣味を若い
　順に表示しなさい

⑤いくつかの属性をグループごとに表示する

検索文　－－＞
　組集合，［の］，属性並び，［を］，
　グループ指定，表示述語．

これは、④のソート指定の代わりにグループ指定がくる形をしている。グループ指定というのは、「部署ごとに」のようなものであって、平均や合計などをある条件を満たすグループごとに表示することを指定する。

［例］２０才以上の社員の給料の平均を部署ごとに
　表示しなさい

５．省略された要素の推定

５．１．自然言語における属性名の省略

自然言語による問合せ文では、データあるいは属性値が属している属性の名前、つまり属性名が省略されることが多い。たとえば、ＳＱＬを使うと、

```
select 名前 from 社員 where
    住所＝"福岡"  and  性別＝"男"
```

と言うべきところを、日本語であれば、

福岡に住んでいる男の人の名前は？

のように、「住所」や「性別」などの属性名を省略してしまう。したがって、関係データベースを対象とした自然言語問合せシステムを作ろうとすると、省略された属性名を推定してやる必要性が生ずる。我々は、属性名というものをデータあるいは属性値に対する制約としてとらえ、属性名リストに制約階層[10]を適用することにより、この問題を解決している。

５．２．属性名リストと制約階層

あるデータがどのような属性に属する可能性があるのかを表したものが、属性名リストである。これは、一般に次のような形をしている。

［属性１，属性２，．．．．，属性ｎ］

たとえば、「３０万円」というデータに、

［給料，ボーナス，年収］

のような属性名リストが付随していた場合、「３０万円」というデータは、給料、ボーナス、あるいは、年収のどれかに属する可能性があることを示している。

属性名リストは、データに課せられた制約であると考えることもできる。そこで、制約階層の概念を部分的に適用してみることにする。ここでは、属性名リストが次のような二つのレベルを持っているものとする。

レベル０：これは、必須制約である。あるデータに、レベル０の属性名リストが付随していた場合、そのデータは、そのリストに記述されている属性のどれかに必ず属さなければならないということを意味する。

レベル１：これは、デフォルト制約である。もし、レベル０の属性名リストによってデータの属する属性が一意に決定できないときには、レベル１の属性名リストも併用する。

この二つの属性名リストからデータの属する属性を一意に決めたいときは、次のようなアルゴリズムを用いる。ただし、これは関係表がただ一つしかない関係データベースに対してのみ有効である。

①レベル０の属性名リストがＮＩＬのときは、入力文のエラーとみなす。

②レベル０の属性名リストがただ一つの要素を持つときは、これがデータの属する属性である。

③レベル０の属性名リストが二つ以上の要素を持つときは、レベル１の属性名リストと積集合をとる集合演算を行なう。この結果できる属性名リストを、仮に積リストと呼ぶことにする。

④積リストがＮＩＬのときは、レベル０の属性名リストの中から適切な属性をユーザに選ばせる。

⑤積リストがただ一つの要素を持つときは、これがデータの属する属性である。

⑥積リストが二つ以上の要素を持つときは、積リストの中から適切な属性をユーザに選ばせる。

５．３．属性名を推定するための情報

データの属する属性を推定するのに、今は次のような情報を用いている。

①データの型
②陽に指定された属性名
③用言と格助詞の対
④接尾語
⑤もともとのデータが属していた属性

社員

名前	性別	住所	給料
山崎	男	長崎	２５，０００
大磯	女	熊本	１８，０００
福岡	女	大分	２１，０００
型	単語	単語	単語	お金
「住む」の格助詞	が	ー	に	ー
接尾語	さん	性	県	ー

図４．社員データベース

このうち⑤以外がレベル０の属性名リストに変換され、⑤はレベル１の属性名リストに変換される。以下、図４のデータベースを例にとって、推定情報を用いて属性名を推定するやり方を説明する。

［例］福岡に住んでいる男の人の名前は？

Ⓐ「福岡」のデータ型は単語型である。データベースから、単語型の属性は［名前，性別，住所］の３つがあることが分かる。これは、レベル０の制約である。

Ⓑ助詞「に」と動詞「住む」の組合せによって、「福岡」は［住所］に関係することが分かる。これも、レベル０の制約である。

Ⓒ「福岡」がもともと属していた属性は、［名前］である。これは、レベル１の制約である。

Ⓓここで、ⒶとⒷの積集合をとると、レベル０の制約として［住所］が得られる。これは、ただ一つの要素を含むので、「福岡」の属する属性として「住所」が推定される。

Ⓔ「男」のデータ型も単語型である。したがって、［名前，性別，住所］のような属性リストが得られる。これは、レベル０の制約である。

Ⓕ「男」がもともと属していた属性は、「性別」である。これは、レベル１の制約である。

Ⓖ「男」に関して得られるレベル０の制約は［名前，性別，住所］であるが、これは二つ以上の要素を含むので、レベル１の制約［性別］との積集合をとって、［性別］を得る。これは、ただ一つの要素しか含まないので、「男」の属する属性として「性別」が推定される。

６．まとめ

ここで紹介したシステムは、ＳＡＸ－Ｃトランスレータの利用によって、少ない作業領域で高速に構文・意味解析を行なうことができる。このため、パーソナル・コンピュータ上であっても、構文規則数を実用上充分なものにすることができる。また、自然言語の問い合わせ文では、属性名が省略されることが多いが、これに対しては、属性名リストと制約階層を使った属性名推定法を開発した。これは非常に柔軟性に富んだ方式なので、あらかじめ動詞類をシステム辞書に登録しておいて、ユーザに使わせながら格助詞と属性の関係をシステムが学習していくこともできる。これによって、ユーザが使用する単語をシステムに教えてやる手間がほとんどいらなくなった。

なお、今後の課題としては、次のようなものが挙げられる。

①より高速に、しかも、より少ない作業領域で動作する構文・意味解析システムを実現するために、ＤＣＧ規則を直接アセンブラ・プログラムに変換してしまうトランスレータを開発したい。

②現在も、文脈に依存した処理を一部受けつけるようにしているが、さらに処理の範囲を広げて、追加質問のようなものも受けつけるようにしたい。

③現在対象としているのは、関係表が一つだけの関係データベースである。したがって、将来的には関係表が複数個ある一般の関係データベースを扱えるようにしたい。その際、ＳＲＴをＳＱＬに変換するトランスレータも作成したい。

④ＳＡＸ－Ｃトランスレータは、ＤＣＧをＣ言語に変換しているため、Ｐｒｏｌｏｇに変換した場合に比べ、デバッグがやりにくい。そこで、ＳＡＸ－Ｃ専用のデバッガを開発したいと考えている。

⑤文脈自由文法で構文規則を記述していくと、構文規則同士が複雑に絡み合い、中には矛盾するものも出てくる。したがって、文脈自由文法で１０００以上も規則が増えたときは、メンテナンスがかなりやりにくくなる。したがって今後は、依存文法処理系を開発し、この上にシステムを移植したいと考えている。

《参考文献》

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

[２] 藤崎他：データベース照会システム「ヤチマタ」と名詞句データ模型，情報処理学会論文誌，Ｖｏｌ．２０，ｎｏ．１，１９７９

[３] 服部他：データベースの日本語インターフェースにおける対話処理について，電子通信学会技術研究報告，Ｖｏｌ．８６，ｎｏ．２１６，１９８６

[４] 伊藤，高橋：データベース用自然言語インタフェース，情報処理学会第３８回全国大会講演論文集，ｎｏ．２，１９８９

[５] Pereira,F.C.N. and Warren,D.H.D.: Definite Clause Grammars for Language Analysis: A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence, Vol.13, 1980

[６] Matumoto,Y., et al.: BUP: A Bottom-Up Parser Embedded in Prolog, New Generation Computing, Vol.1, No.2, 1983

[７] 松本，杉村：論理型言語に基づく構文解析システムＳＡＸ，コンピュータソフトウェア，Ｖｏｌ．３，ｎｏ．４，１９８６

[８] Aho,A.V., Hopcroft,J.E. and Ullman,J.D.: Data Structures and Algorithms, Addison-Wesley, 1983

[９] 横田：省略を含む自然言語談話の理解処理について，第３回情報処理学会九州支部研究会報告，１９８９

[１０] Borning,A., et al.: Constraint Hierarchy, Proc. of OOPSLA, 1987