

Title	自然言語による計算機との対話
Author(s)	上原, 邦昭
Citation	大阪大学大型計算機センターニュース. 1986, 60, p. 57-83
Version Type	VoR
URL	<a href="https://hdl.handle.net/11094/65678">https://hdl.handle.net/11094/65678</a>
rights	
Note	

*Osaka University Knowledge Archive : OUKA*

<https://ir.library.osaka-u.ac.jp/>

Osaka University

# 自然言語による計算機との対話

大阪大学産業科学研究所 上原邦昭

## 1. 幻 想

U : I have been very depressed lately.

(最近とても気分が滅入ってしまうのです)

S : You say you have been very depressed lately.

(あなたは最近とても気分が滅入ってしまうのですね)

U : Yes, my mother beats me and my father calls me names.

(はい、母は私をおちますし、父は悪口を言うのです)

S : You're being somewhat short with me.

(少し私に無愛想ですね)

U : What about my mother beating me ?

(母がおつことをどう思いますか)

S : Tell me more about your family.

(もっと家族のことを話して下さい)

U : My father calls me nasty names.

(父親は意地の悪いことを言うのです)

S : Who else in your family calls you nasty names ?

(他に御家族のどなたが意地の悪いことを言いますか)

U : I think the dog would if he could.

(もしできるのなら犬も悪口を言うと思います)

S : Do you think it's likely that he could ?

(犬が悪口を言えると思っているのですか)

U : No, do you.

(いいえ)

S : You don't seem very talkative today.

(今日はちっともしゃべりませんね)

U : I guess what is really bothering me is that I don't have any money.

(本当の悩みはお金を持っていないことだと思います)

S : Can you elaborate on that ?

(その点を詳しく話して下さい)

U : The problem is that I know psychiatrists are expensive and I may not be able to pay you.

(問題なのは精神科医はお金がかかりますでしょう、だから、あなたにお支払いできないかもしれない点なのです)

S : Please continue, this may be interesting.

(どうぞ続けて下さい、これは面白い話だと思います)

U : I thought you might think so.

(そうだろうと思っていました)

S : Perhaps in your fantasy we thought each other.

(たぶん私たちがお互いにそう思うと空想したのですね)

U : Right doc.

(はい、先生)

S : Earlier you said your mother beating you.

(前にお母さんがあなたをおつと言いましたね)

これは、1966年にWeizenbaumによって開発されたELIZA (Weizenbaum 1966) と呼ばれるプログラムの対話例である。ELIZAは、ユーザの発言を分析する精神科医のような振る舞いを示し、個人的な問題を文にしてタイプすると、相手の発言を忍耐づよく確認しながら返答する。このように、ELIZAはプログラムと思えないほどすばらしい知性を示し、多くの注目を集めた。ELIZAによって精神科医は将来失業するだろうと恐れられたほどである。しかしながら、本当の正体は、単純なメカニズムによって構成された小さなプログラムの集まりにすぎない。

自然言語処理の研究(いわゆる人工知能や知識工学を含む)では、論文中に示されるのは「うまくいった例」のみで、「失敗した例」は示されないことが多く、研究の本質を見失うと、計算機(人工知能)万能という幻想にとらわれるかもしれない。本小論では、まずELIZAの種明かしをし、次に現在どのようなことが可能で、どのようなことが不可能かを、実例を示しながら、そのメカニズムも含めて解説する。

## 2. ELIZAのメカニズム

ELIZAは、ユーザの発言を理解しているわけではなく、単に入力文からキーワードを見つけ、対応する応答をしているにすぎない。たとえば、入力文に”I am”という句を含めば、ELIZAの持つ規則(I AM\_)と条件が一致し、それに対応する出力文を生成するわけである。ユーザの入力文と照合させる規則の各単語には、次のように番号がつけられている。

( I AM \_ )

1 2 3

もし、この規則が入力文と一致すると、ELIZAは応答文として

(HOW LONG HAVE YOU BEEN 3?)

を発生し、応答文の3の部分に、入力文の3の部分置き換えて出力する。たとえば、ユーザが

" I am very unhappy these days "

と入力すると、ELIZAは

"HOW LONG HAVE YOU BEEN VERY UNHAPPY THESE DAYS?"

と応答するわけである。したがって、たとえば

" I am a boy "

と入力すれば

"HOW LONG HAVE YOU BEEN A BOY?"

というとんでもない応答が生成される可能性もある。

規則はキーワードによって分類して記憶されている。各キーワードに対応して、いくつかの規則が用意されており、各規則には複数の応答文が用意されている。複数の応答文は順に選ばれるようになっている。このため、同じ入力文でも異なる応答をすることが可能になっている。また、代名詞を正しく対応させるために、規則の照合を行う前に入力文の"you"を"I"に、"I"を"you"に置換している。キーワード"remember"に対応する規則と応答は、次のようなものである。

キーワード REMEMBER

規則 ( \_ YOU REMEMBER \_ )

応答 DO YOU OFTEN THINK OF 4 ?

DOES THINKING OF 4 BRING SOMETHING ELSE TO MIND ?

WHY DO YOU REMEMBER 4 JUST NOW ?

WHAT IN THE PRESENT SITUATION REMINDS YOU OF 4 ?

WHAT IS THE CONNECTION BETWEEN ME AND 4 ?

規則 ( \_ DO I REMEMBER \_ )

応答 DID YOU THINK I WOULD FORGET 5 ?

WHY DO YOU THINK I SHOULD RECALL 5 NOW ?

WHAT ABOUT 5 YOU MENTIONED 5 ?

規則 ( \_ )

応答 NEWKEY

最後の規則 (一) はどんな文とも一致することを意味している。上の二つの規則に一致しないときには、(二) と一致することになり、NEWKEY の応答がなされる。NEWKEY は入力文から新しいキーワードを探し、再処理することを示している。

このように ELIZA は入力文を理解しているとはいえないが、精神科医とその患者という特殊な状況設定では、驚くほどもっともらしい会話をすることができる。しかしながら、メカニズムとしてはパターンの照合のみで、プログラム自体は知的な動作を一切行っていない。次章では ELIZA と同様に特別な言語処理機能は持っていないが、もう少し高度なメカニズムで応答文を生成するエキスパートシステムの説明機能について述べる。

### 3. エキスパートシステムにおける対話例

本特集 1 の「エキスパートシステム構築のための言語と支援ツール」で、エキスパートシステムの説明機能が解説されている。この説明機能は、エキスパートシステムが行った推論の過程を、自然言語でユーザに提示するために不可欠なものである。本章では、MYC IN (Shortliffe 1976) を例にとって、説明文の生成過程を解説する。MYC IN のルール

RULE084

```
PREMISE: ($AND (GRID (VAL CNTXT PORTAL) PATH-FLORA)
            (SAME2 CNTXT GRIDVAL
              (QUOTE (GRAM MORPH AIR))))
```

```
ACTION: (CONCLIST CNTXT IDENT GRIDVAL .8)
```

は、以下の英語に翻訳される。

RULE084

```
IF: 1) THE LIST OF LIKELY PATHOGENS ASSOCIATED WITH THE
      PORTAL OF ENTRY OF THE ORGANISM IS KNOWN, (この菌の侵入
      門戸に関連があると考えられるもっともらしい病原体のリストがわかっており、)
      AND (かつ)
```

```
2) THIS CURRENT ORGANISM AND THE MEMBERS YOU ARE
      CONSIDERING AGREE WITH RESPECT TO THE FOLLOWING
      PROPERTIES: GRAM MORPH AIR (この菌といま考慮の対象となっている
      病原体とが共に次の特徴を持っている: グラム染色性、形態、好気性)
```

```
THEN: THERE IS STRONGLY SUGGESTIVE EVIDENCE (.8) THAT EACH
      OF THEM IS THE IDENTITY OF THIS CURRENT ORGANISM
      (この菌が考慮対象の病原体であることを示唆する強い根拠(.8)がある)
```

この過程を RULE084 の PREMISE 条件 (GRID (VAL CNTXT PORTAL) PATH-FLORA) を用いて説明する。MYCIN のルールで使われるすべての記号は TRANS と呼ばれる属性を持っている。この属性は、ルールに現れる各記号がどのような英語に翻訳されるかを示したもので、翻訳辞書に相当する。PREMISE 条件に含まれる各記号の TRANS 属性を以下に示す。

GRID:            (THE (2) ASSOCIATED WITH (1) IS KNOWN)  
VAL:             (((2 1)))  
PORTAL:         (THE PORTAL OF ENTRY OF \*)  
PATH-FLORA:    (LIST OF LIKELY PATHOGENS)

TRANS 属性に含まれる数は、対応する引数の翻訳が代入されることを示している。したがって、GRID の第 2 引数 (PATH-FLORA) の翻訳 (LIST OF LIKELY PATHOGENS) が、GRID の TRANS 属性の (2) に代入される。VAL の TRANS 属性に付いている余分な括弧は、VAL の第 1 引数 (CNTXT) の翻訳 (詳しくは述べないが、THE ORGANISM と訳される) が VAL の第 2 引数 (PORTAL) の TRANS 属性中の星印部分に代入されるべきことを示している。したがって、(VAL CNTXT PORTAL) は

THE PORTAL OF ENTRY OF THE ORGANISM

となる。この翻訳を GRID の TRANS 属性の (1) に代入し、PATH-FLORA の翻訳をその (2) に代入すれば、PREMISE 条件は最終的に

THE LIST OF LIKELY PATHOGENS ASSOCIATED WITH THE  
PORTAL OF ENTRY OF THE ORGANISM IS KNOWN

となる。

ELIZA は入力文のパターンと出力文が対になっていたが、MYCIN ではルールの各記号と英語が一对一に定められているため、それらを組み合わせて、少々複雑なルールも完全な英語に翻訳することができる。しかしながら、自然言語には曖昧性とか多義性といった困難な問題が横たわっている。このような問題に対処するためには、言語理論で得られた諸成果を積極的にシステムのメカニズムとして取り込んでいかなければならない。以降の章では、言語理論の成果を応用したシステムについて述べる。

#### 4. 意味の導入

1970 年代前半より、従来のシステムで欠落していた「意味」を取り扱おうとする気運が生まれしてきた。単純なマッチングや統語論のみではどうしても取り扱えない事例が発見されてきたため

ある。また対象分野を特定の領域に限定すると、「意味」を導入することによって比較的早く、簡単にシステムを構成することができる。さらに、自然言語インターフェイスは素人も使用するために、頑丈(robustness)でなければならないという要請がある。ここでシステムの頑丈さとは、文法的でなかったり、誤りがあったり、省略を含むような入力にも十分対応できる能力を持っていることである。このような問題も、「意味」を用いれば柔軟に対処できるという利点がある。

LIFER(Language Interface Facility with Ellipsis and Recursion)(Hendrix 1977)は、SRIのGary Hendrixが任意の分野における応用のための「自然言語フロントエンド」を実現するためのユーティリティとして設計したものである。LIFERは、いくつかの自然言語インターフェイスを作成するために実際に利用されている。たとえば、医療情報データベース、タスク・スケジューリングと資源割り当てシステムなどがある。LIFERインターフェイスを用いて作成された最も複雑なシステムは、自然言語フロント・エンドの開始に数人月を要したLADDER(Language Access to Distributed Data with Error Recovery)システム(Hendrix 1978)で、海軍の指揮と統率のデータベースに関する質問に答えるように設計されている。

LIFERシステムは、次の二つのコンポーネントからなる。

- 1) 言語の仕様を指定するための対話機能
- 2) 入力された文を解析するためのパーザ

インターフェイスの作成者は言語仕様機能を用いて、テンプレートの形で入力される質問文の形式を記述する。テンプレートの一例を示す。

WHAT IS THE <ATTRIB> OF THE <SHIP> ?

パーザは左から右に入力単語系列(ユーザの質問文)とテンプレートとのマッチングを取る。上のテンプレートがユーザの質問文と一致するためには”What is the …”で始まるものでなければならない。<ATTRIB(属性)>、<SHIP(船)>は非終端記号と呼ばれるものである。たとえば<SHIP>には船の名(潜水艦ノーチラス、航空母艦ジョン・F・ケネディ、コンステレーションなど)、<ATTRIB>には船の持つ属性(全長、排水量、等級など)がある。上のテンプレートは以下のような文とマッチングする。

- i) What is the length of the Constellation ?
- ii) What is the displacement of the Nautilus ?
- iii) What is the class of the John F. Kennedy ?

上のテンプレートと質問文がマッチングすると、分析木が作られ、さらに次のようなデータベース問い合わせ言語が生成される。

((NAME EQ <SHIP>)(? <ATTRIB>))

これは名前(NAME)が<SHIP>と等しい(EQ)船のデータベースから<ATTRIB>に関する情

報を探すコマンドである。たとえば、1) の質問の分析木は

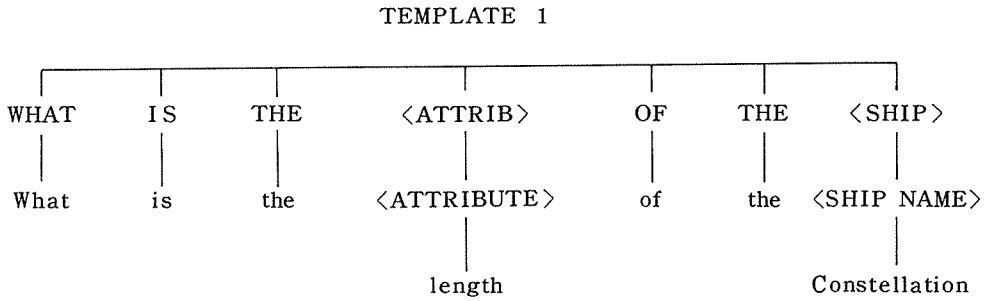


図 4.1 文 1) の分析木

問い合わせ言語は

((NAME EQ CONSTELLATION)(? LENGTH))

となる。

LIFERには、ユーザが対話を簡単に行うために、綴り訂正や省略の補足を行うための二つの付加的な機能が組込まれている。綴り訂正は、LIFERの記述言語である Interlisp (Lispの方言の一種)の綴り訂正機能によって実現され、LIFERが入力を解析することに失敗した場合に試みられる。この機能は、疑わしい単語の綴りに最もよくマッチングが行える単語の候補を見つけ出すものである。綴り訂正を行った応答例を以下に示す。

\*\* How far is the Kitty Hwk from Gibraltar ?

(Kitty Hwk はジブラルタルからどれだけ離れているか?)

Spelling → Hawk (LIFERがユーザの綴り誤りを指摘している)

Parsed ! (構文解析が成功したことを示すメッセージ)

(GCDIST 1087)

(距離は 1,087 海里である)

省略の補足には先行する文の分析木が用いられる。たとえば、以下の文

1) What is the length of the Constellation ?

2) The home port.

が入力された場合、LIFERは文 1) を参考にして、文 2) の分析木を推論し、"What is the home port of the Constellation?" の略であろうと決定することができる。

省略の補足は次のようにして行われる。まずパーザは、テンプレートとのマッチングを試みる。2) はどのテンプレートともマッチングしないので、次に綴り訂正を試みる。2) の各単語はすべ



て正しい綴りなので、これも失敗する。そこで前の文1)の分析木を調べる。図4.1参照のこと。もし分析木の部分に一致した省略がなされているなら、その省略された部分を分析木に付け加え、完全な分析木を再構成する。図4.2は、再構成された省略文の分析木である。

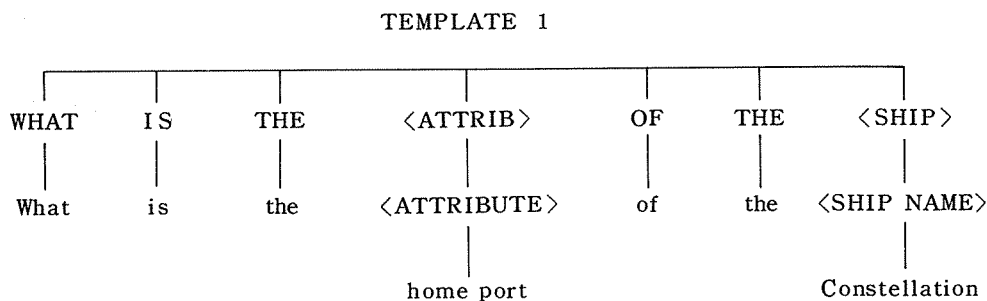


図 4.2 文 2) の分析木

この分析木から省略文2)に対応する新しい質問が作られ、問い合わせ言語が生成される。

LIFERで用いられた文法(テンプレートの集合)は、非終端記号として、対象としている分野における意味的な情報を反映した意味カテゴリを用いている。たとえば、先の例では<SHIP(船)>、<ATTRIB(属性)>などである。このような文法は意味文法と呼ばれ、知識情報処理システムの開発によく用いられるものである。意味文法は、言語理論における困難な問題を扱うことはできないけれども、特定の目的に適合したパーザを比較的早く、簡単に構成することができるため、簡単な応用のための自然言語フロント・エンドを作成するのに適している。たとえばLADDERの場合、ユーザの質問は構文的には限られているのだから、実際の操作では、ユーザはLADDERの統語的能力(文法を理解する能力)の弱さに気がつかないだろうといわれている。しかしながら、意味文法には、以下のような欠点がある。

- 1) 対象領域を絞らない限り、意味カテゴリ数が際限なく増加する可能性がある。
- 2) 文法規則が対象分野に強く依存しているため(LADDERは海軍情報データベースを調べるために設計されている)一般性がない。したがって、対象分野が異なる場合、仮に使われている語彙や概念が大変似かよっていても、すべての文法規則を書き直さなければならないということになる。

## 5. 文法を逸脱した文の処理

前章では頑丈さを達成するために、意味を導入することについて述べた。本章では、システム側が自分の持つ文法の制約条件を、入力文に応じて緩和し、文法を逸脱した文を柔軟に解析するシステムについて述べる。

人間が会話をする際、話し手が常に文法規則に従った言語表現のみを用いているとは限らない。

それにもかかわらず、その文法的逸脱の度合いが少ない場合には、聞き手が柔軟に解釈・理解して、会話がスムーズに進行することが多い。こうした人間の行為を考えると、人間と計算機が自然言語でやりとりを行なおうとする場面に於いても、人間は、文法を逸脱した文を無意識的に入力することがあると考えられる。

文法的逸脱は人間 (native speaker) が同意する逸脱か否かによって、絶対的逸脱と相対的逸脱に分類される。前者はシステムも人間も共に誤りと認める言語表現である。後者は人間は誤りと思わないが、システムの用意している文法規則の限界を越えている言語表現である。それぞれについて以下に例を示す。

(1) 絶対的逸脱

- 文法的制約条件の誤り

He want to buy these book. (数の一致)

He shall lives in Tokyo. (動詞の活用)

I saw he at the station. (格)

Give \_\_\_\_ job description for Mike Smith. (冠詞の省略)

Jack is is here. (不要な語句の挿入)

(2) 相対的逸脱

- 擬人法や比喻による意味的制約条件の誤り

The car drinks gasoline.

- 挿入語句

Show me please the way to the station.

- 接続詞句

John loves Mary and (John) hates Sue. (省略)

接続詞句は文法的に合法的なものであるが、接続詞句を受理する文法規則を用意している自然言語インターフェイスが少ないという認識の下に、ここでは相対的逸脱と分類している。

一方、従来の自然言語インターフェイスは、各システムが個々に持つ文法規則に合致する入力文のみを処理対象とし、合致しない文が入力された場合は単に拒絶して再入力を要求するのみであった。従って、実用性という面から見れば、不十分であったと考えられる。

Eastman と McLean の実験 (Eastman 1981, 1984) によれば、これらの文法的逸脱の起こる度合いとして、表 5.1 のような統計が得られている。実験は、231名の被験者 (フロリダ大学の学生) に対し、人事情報 (社員名、住所、性別、婚姻の有無、学歴、専門分野、所属等) を管理するデータベースに対する簡単な質問文を 1人あたり 3文ずつ作成させ、集められた 693文中に含まれる文

法的逸脱を分析したものである。

表 5.1 文法的逸脱の発生する割合

文法的逸脱の種類	文数（693文中）	割合
文法的制約条件	85文	12.3%
代名詞の格の誤り	4	0.6
主語と動詞の数の一致	16	2.3
動詞の活用の誤り	2	0.3
アポストロフィーの 用いかたの誤り	46	7.6
所有格変形の誤り	12	1.7
冠詞、数詞と名詞の数の一致	5	0.7
その他	9	1.3
語の省略		
冠詞の省略	23	3.3
前置詞の省略	4	0.6
その他の語の省略	8	1.2
余分な語句の挿入	11	1.6
不完全文・断片文	64	9.2

上の例からもわかるように、情報システムへの自然言語アクセスの場合、かなりの入力文が文法的に逸脱している可能性がある。この状況は、辞書や文法がどんなに大きくなっても、自然言語という性格上つねにつきまとう問題である。

以上のような認識に立ち、我々は文法的逸脱を含む入力文に対しても柔軟に構文解析を行なうシステムFPS (Flexible Parsing System) (井上 1985) の開発を進めている。FPSは、Pereiraらが提案したDCG (Definite Clause Grammar) モデル (Pereira 1980) を拡張し、1) 接続詞句の解析処理、2) 制約条件を逸脱した表現の柔軟な解析処理、3) 文法規則に適合しない挿入語句の読み飛ばし処理の三つの機能を付加したものである。

文法的不適格文の一部は、入力文が文法規則の設けた制約条件に違反したために生じたものであるから、それらを単に文法規則から取り除けばよいとする考え方がある。しかしながら、文法規則中に置かれる制約条件は、単に文法的整合性を調べているだけではなく、1) 曖昧性を除去したり、2) 誤った文法規則の適用を早期に排除することによって探索空間を制限し、解析効率の改善に役立っている場合が多い。また、3) 構文解析時にのみ得られる意味情報を抽出する場合もある。FPSでは、文法規則に制約条件を残し、入力文に制約条件を逸脱した箇所がある場合に限って、そ

の逸脱部分をチェックしている制約条件のみを局所的に緩和して解析を進めるという手法で対処している。制約条件を緩和する際は、

- (1) 文法記述者によって緩和可能であると指定されている制約条件のみを緩和の対象とする。
- (2) 制約条件が失敗した場合でも、すぐに緩和するわけではなく、失敗した制約条件をいったんマーク付けしたうえで再度解析を試み、他の解釈による解析木が生成できないかどうか調べる。それも失敗し、先にマーク付けした制約条件を緩和しなければ、全く解析木が生成できないことを確認した後に緩和する。

制約条件を緩和して解析が成功した場合、システムの解釈が正しいかどうか実際に文を入力した人の同意を求めるため、エラーメッセージを出力する。エラーメッセージは、誤りの種類と入力文中の逸脱箇所の2点を指摘する。図 5.1 に例を示す。

```

I gave he three pencil.

制約条件を緩和しました !!
代名詞の格が誤っています。
逸脱箇所は
  he
  の内部もしくは近辺です。
名詞句内で左修飾部と名詞の数が一致していません。
逸脱箇所は
  pencil
  の内部もしくは近辺です。

cputime = 12.033325 sec.
sentence = assertion = sa = []
  subject = lnr = ln = []
    nvar = pro = * i *
    rn = []
  tense = []
  sa = []
  verb = |v = []
    |v = * gave *
    rv = []
  object = nn = lnr = ln = []
    |nvar = pro = * he *
    |rn = []
    |lnr = ln = tpos = []
      |qpos = |qr = |q = []
        |avar = q = * three *
        |apos = []
        |nvar = n = * pencil *
        |rn = []
    rv = []
  sa = []

```

図 5.1 制約条件の緩和

また、挿入語句に対しては、それらを受理する文法規則を追加すればよいという考え方がある。しかしながら、絶対的逸脱となる挿入語句を前もって予期するのは実際問題として不可能であり、仮に可能だとしても解析すること自体無意味であると考えられる。一方、相対的逸脱となる語句の挿入はある程度予測可能であるが、すべてを文法規則で用意しようとすると、無制限に文法規模が増大して解析効率が著しく低下してしまう。FPS では、文法の規模は適当なサイズにとどめ、そ

の許容範囲を越える挿入語句は単に読み飛ばすこととしている。挿入語句を読み飛ばした場合にも、制約条件を緩和した場合と同様に、システムの解釈が正しいかどうか実際に文を入力した人の同意を求めるために、メッセージと読み飛ばした語句を明示する。図 5.2 に例を示す。

```

Show me please the way to the station.

挿入語句
  please
を読み飛ばしました。

cputime = 3.849975 sec.
sentence = imperative =
  do = []
  vverb = |v = []
           |tv = * show *
           |rv = []
  object = nn = |nr = |ln = |in = []
                 |nvar = pro = * me *
                 |rn = []
                 |nr = |ln = |tpos = t = * the *
                       |qpos = []
                       |apos = []
                       |nvar = n = * way *
                       |rn = pn = p = * to *
                             |nr = |ln = |tpos = t = * the *
                               |qpos = []
                               |apos = []
                               |nvar = n = * station *
                               |rn = []

```

図 5.2 挿入語句の読み飛ばし

接続詞句の形態をとるものは、名詞句、前置詞句、形容詞、あるいは文など、ほとんどすべての文法カテゴリに渡っている。このため、すべての接続詞句のパターンを文法規則として用意することは不可能である。FPS では、入力文を左から右に解析してゆき、接続詞句が出現した場合にはその解析を一時中断して、接続詞句解析用の特別なルーチン（デーモンと呼ばれている）を起動する。接続詞句の解析が終了すると、先に中断していたメインの解析を再開する。このように、標準的な文法と接続詞句処理用の特別なルーチンを分離して、文法サイズの肥大化を防いでいる。接続詞句内部の解析においては、中に含まれる省略語句の取り扱いが問題となる。接続詞句内部で省略された部分が発見されると、その文法カテゴリ以前に現れる同一の文法カテゴリとの間で語句が共有されているとみなし、以前の解析結果をそのまま省略部分の解析結果として補足する。接続詞句内部の省略を補足しながら解析を進めることにより、最終的に省略のない完全な解析結果を得ることができる。図 5.3 に例を示す。

He chose a rose and she too.

cputime = 5.599975 sec.

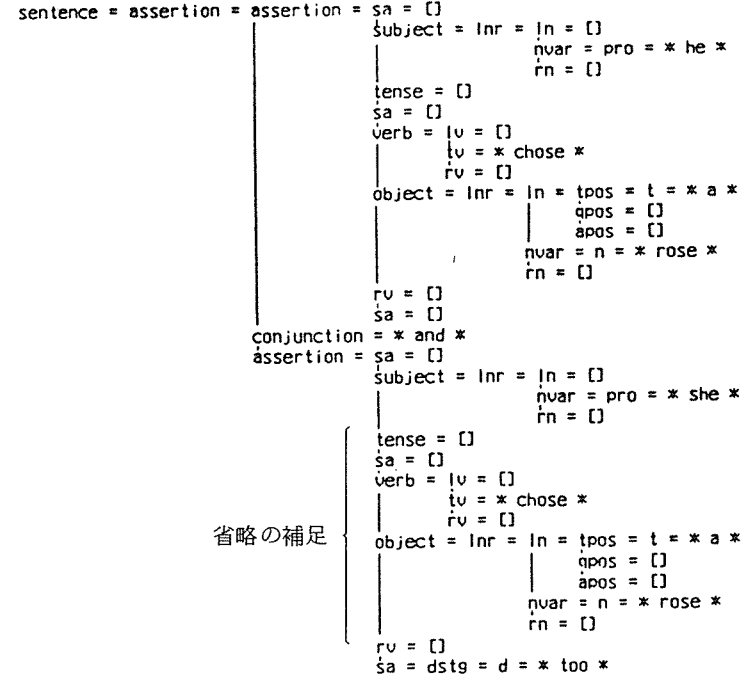


図 5.3 接続詞句の処理

## 6. 自然言語インターフェイスにおける協調的な対話

LIFERは、「もし会話が小さな対象分野についての質問に限定されているならば、かなり正確に質問の意味を認定できるであろう」という仮定に基づいて設定されている。したがって、ユーザが対象分野における語彙や概念、システムで使用可能な文体のパターンを熟知しているならば、すばらしい応答を発揮する。しかしながら、全くの素人のユーザでは使いものにならないということもありうる。また、仮にLIFERで使用可能な語彙や文法を知っていても、対象分野のデータベースに蓄積されたデータについて知らなければ、大変使いづらいものとなる。本章では、ユーザが的是ずれの質問を行ったとき、あるいは実行不可能な問い合わせをしたときに、推論機能を用いて協調的な（システム側からユーザに歩み寄る）応答を行うことができるシステムについて説明する。

たとえば Kaplan が示した仮想的な自然言語によるデータベースへの問い合わせ例について考える (Kaplan 1983)。

User: Which students got a grade of F in CS105 in Spring 1980 ?

System: Nil. (空集合であることを示す)

User: Did anyone fail CS105 in Spring 1980 ?

System: No.

User: How many people passed CS105 in Spring 1980 ?

System: Zero.

User: Was CS105 given in Spring 1980 ?

System: No.

なぜこのようなつまらない対話が生じたのであろうか。これは、ユーザが「CS105 という科目が1980年の春に開講されている」と頭から信じて疑わなかったためである。確かに上例での”No”あるいは”Zero”などは正しい答えであるが、ユーザの無用な誤解を招き、無駄な質問を長々と繰り返す原因となっている。このような場合、協調的なシステムでは、質問の前提となっている誤りを発見し、ユーザの誤解を解くために、以下のような応答を返さなければならない。

User: Which students got a grade of F in Spring 1980 ?

System: CS105 was not given in Spring 1980.

CO-OP (Cooperative Query System) (Kaplan 1983) はこのような応答を与えるために設計されたシステムである。CO-OPでは、ユーザの問い合わせから推論し、もしデータベースの状態がユーザの見込みと食い違うとき (yes/no 疑問文に no と答えるようなとき)、本来の正しい応答よりもさらに有益な、矯正的な間接応答 (corrective indirect response) を以下のようにして生成する。

1) 与えられた質問をパースし、MQL (Meta Query Language) という問い合わせ言語で計算機内部に表現する。MQL はグラフで表わすことができ、各ノードはユーザによって提示された概念に属する値の集合、各リンクは集合間に定義された関係を表すものである。質問

”Which students got Fs in Linguistics courses?” (言語学の成績がFの者は誰か?) は、図 6.1 のように表現される。

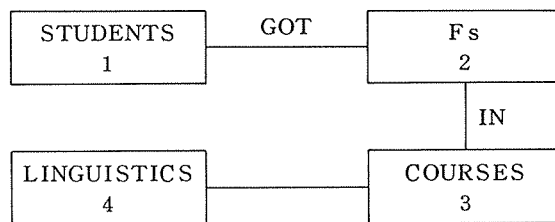


図 6.1 MQL 表現

概念 "Linguistics", "Fs" は可算集合とならないように感じる読者もいようが、ここでは、それぞれ「言語学の授業を受けた者全体」、「成績がFの者全体」の集合を表すものとする。

2) MQL を用いてデータベースの検索を実行する。ユーザの前提に誤りがある場合は、このグラフを参照しながら、どのノードに対応する集合が空集合であるかによって、以下のような協調的な応答を示すことができる。

- 1) " I don't know of any students."
- 2) " I don't know of any Fs."
- 3) " I don't know of any courses."
- 4) " I don't know of any Linguistics."
- 1,2) " I don't know of any students that got Fs."
- 2,3) " I don't know of any Fs in courses."
- 3,4) " I don't know of any Linguistics courses."
- 1,2,3) " I don't know of any students that got Fs in courses."
- 2,3,4) " I don't know of any Fs in linguistics courses."

上の応答で、文の左に示された数字はどのノードが空集合であることを示したものである。たとえば、4番目の応答では "Linguistics" が空集合なので、授業でFをとった学生はいるが、言語学自体が開講されなかったことを示している。

ユーザの質問の前提には誤りがないが、質問の制約が厳し過ぎる場合、それを緩めることによって親切な応答を返すことが可能になることがある。これを示唆的な間接応答 (suggestive indirect response) と呼ぶ。すなわち、ユーザへの正確な返事にはなっていないが、付加的な情報を加えることによって、ユーザが不要な質問をさらに繰り返さなくても済むというものである。以下の例も (Kaplan 1983) より採られたものである。

User: Are there any seats available in the orchestra for tonight's Rolling Stones' concert ?

System: No.

User: Are there any in the balcony ?

System: Yes.

上の会話では席に関する制限を除去し、すなわち質問の焦点となっている制約条件を緩め、再度データベースを探索し、そこで得られた答えを基にして、以下のような応答をすれば、より自然な会話が可能になる。



User: Are there any seats available in the orchestra for tonight's Rolling Stones' concert ?

System: No, but there are some in the balcony.

上の質問文を MQL で表わせば、図 6.2 のようになる。

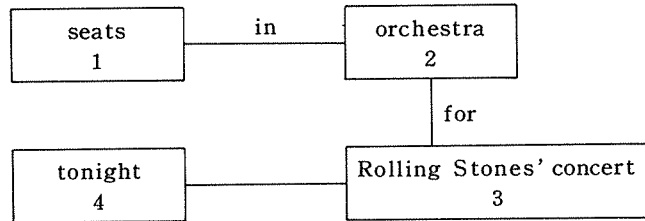


図 6.2 MQL 表現

このうち、焦点となっている部分はオーケストラボックスなので、その制約（ノード 2）を緩め、図 6.3 を新たに生成する。

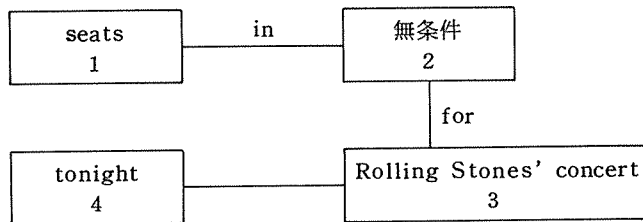


図 6.3 制約を緩められた MQL 表現

ここで、ノード 2 にバルコニー席をあてはめれば、MQL のすべてのノードが空でなくなると仮定する。バルコニー席に関する情報は、直接ユーザの望んでいるものではないが、少なくともユーザにより多くの情報を与えるという点で協調的である。上の応答例では、システムはバルコニー席にまだ空席があることをユーザに知らせている。

## 7. 意図の推定に基づく対話

前章では、推論によって協調的な応答をするシステムについて述べた。本章では、さらにユーザの間接的な発言から暗に含まれている意図を抽出し、その意図に沿って応答するシステムについて述べる。

計算機を十分に使いこなしている専門家でも、コマンドやパラメータの順序を間違えて、思わぬ

失敗をすることがある。ましてや素人のユーザはなおさらである。計算機のOSの一つにUNIXがある。UNIXは、ミニコン等で広く知られたOSであるが、専門家の使用を仮定して設計されているために、コマンド体系は決してきれいなものではなく、ユーザに余りメッセージを返さないようになっている。したがって、初心者は頻繁にシステム管理者に質問したり、自分でマニュアルを調べる必要がある。このような問題を解決するシステムとして、UC (Unix Consultant) (Wilensky 1984)がある。UCはUNIXについてユーザが説明を求めると、自然言語で解説してくれるコンサルテーションシステムである。このUCは、カルフォルニア大学バークレー校で開発されたもので、VAX11/780上で稼動している。

まずUCの対話例を示す。

User: How can I delete a file ?

UC: Typing 'rm filename' will remove the file with name filename from your current directory.

User: Move a file ?

UC: Typing 'mv filename newfilename' will move the file with name filename to a file with name newfilename.

User: How might a file be created in UNIX ?

UC: The best way to create a new file is to use the editor, i. e. -'ex filename'.

UCが従来のシステムと異なる点は、ユーザの間接的な発話から隠されている意図を抽出し、それに基づいて意図を満たすような応答をしている点である。たとえば、ユーザが

I need more disk space.

という文を入力すれば、UCはまずこの文から「より多くのディスク空間を獲得したい」というユーザの隠された意図を抽出する。この意図に従って応答を生成するわけであるが、問題となるのは、単純に

Type 'rm \*' (すべてのファイルを消去せよ)

という応答を返す可能性があることである。このようなことをすれば、当然システムが破壊されてしまう。そこでUCは、「ユーザは通常必要なファイルを保持しておきたいものだ」という常識的な知識を用いて、

Delete all the files you don't need.

あるいは、

Ask the systems manager for some more storage.

という応答を返すことができるようになっている。

## 8. 自然言語インターフェイスの移植性

前章までは自然言語インターフェイスへの要請のうち頑丈さについて述べた。もう一つの自然言語インターフェイスへの要請として、移植性という点が挙げられる。LIFERは予め与えられたデータベースに対してのみうまく働くような、対象分野固有の知識を利用した意味文法を用いて設計されていたために、他の分野のデータベースに移植しようとするとはじめから作り直さなければならなかった。移植性を高めるためには、分野固有の知識と分野独立な知識を分離しなければならない。このような方針の基に開発されたシステムにXCALIBUR (Carbonell 1983)がある。

XCALIBURは、XSEL (McDermott 1982)と呼ぶエキスパートシステムに自然言語によってアクセス(入力文の理解と出力文の生成)できるようにするための自然言語フロント・エンドである。XSELはDEC社のセールス・エンジニア用に開発されたエキスパートシステムで、ユーザとの対話によってVAXコンピュータの周辺機器を自動的に選定するものである。VAXファミリーは、約8,500のコンポーネントから通常10~20の構成要素を選択して、システムが構成される。XSELが行う仕事の内容は、エキスパートシステムとしてあまり高度なものとは言えないが、セールス・エンジニアのミスを防ぎ、機器マニュアルを調べる労力を省くことを目的としているために、自然言語フロント・エンドは不可欠である。図8.1にXCALIBURの概念構成図を示す。

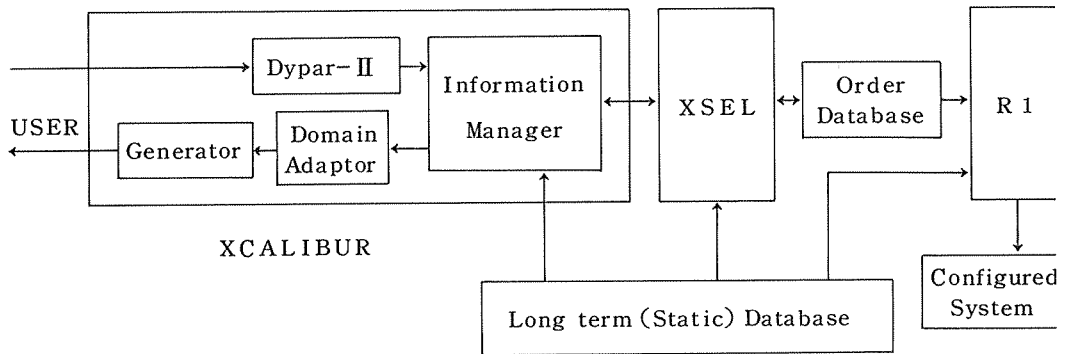


図 8.1 XCALIBUR の概念図

以下にXCALIBURの応答例を示す。

>What is the largest 11780 fixed disk under \$40,000 ?

The rp07-aa is a 516 MB fixed pack disk that costs \$38,000.

>Add two rp07-aa disks to my order.

Line item 1 added : (2 rp07-aa)

>Add a pointer with graphics capability.

fixed or changeable font ?

>fixed font.

lines per minute ?

>make it at least 200, upper/lowercase.

Ok, Line item 2 added:(1 lxy11-sy)

>Tell me about the lxy11.

The lxy11 is a 240 LPM line printer with plotting capability.

XCALIBURは、LIFERのようにデータベースに依存した自然言語インターフェイスとしてではなく、汎用性のあるものとするために、DYPAR-II (DYnamic PARsing-II)(Hayes 1981) というパーザを用いている。DYPAR-IIでは入力文を解析すると直接問い合わせ言語に変換することなく、いったん格フレーム表現(case frame representation)と呼ばれる中間形式に変換する。たとえば、例文

>What is the price of the largest dual port fixed media disk ?

は、以下のように変換される。

```
[QUERY (OBJECT (SELECT (disk ..... データベースで
                        (ports (VALUE (2))) デュアルポート
                        (disk-pack-type (VALUE (fixed))) の固定ディスク
                        について調べよ
                        (OPERATION (SORT ..... そのうちでディスクの
                        (TYPE (*descending)) サイズが最大のものを
                        (ATTR (size)) 選べ
                        (NUMBER (1)))
                        (PROJECT (price)) ..... ディスクの価格を求めよ
                        (INFO-SOURCE (*default)))))] ..... データベース名を示している
```

さらにこの格フレーム表現は、Information Managerによってデータベース問い合わせ言語に変換される。データベース検索後、答えとして

```
(render-result
  (class (disk))
  (actor-list (rp07-ba))
  (projection-attr (price))
  (focus (ports disk-pack-type
```

number-of-megabytes)))

という結果が返される。この検索結果は、domain adaptor によって、以下のような概念依存表現 (conceptual dependency representation)

(cd-be ..... 文型は be 動詞文  
(actor (rp07-ba (ref (def)))) ..... 主語 (主格) は rp07-ba  
(object ..... 目的語 (目的格) は  
(disk (focus (ports media number-of-megabytes price))  
(rel (cd-cost (actor (disk)) ..... 関係詞節で、ディスクの価格  
(object (dollar (count (43140)) ..... は 43,140ドルであることを示  
(focus (count)))))) ..... している  
(tense (pres))) ..... 時制は現在形

へと変換される。この概念依存表現は generator に渡され、最終的に出力文

The rp07-ba is a dual port fixed pack 516 MB disk that costs \$43,140 dollars.  
が生成される。

XCALIBURでは、たびたび表現形式の変換を行なう複雑な言語処理方式を採用しているが、information Manager と domain adaptor の持つ内容のみを変更することによって、各種のデータベースと接続することができ、LIFER等に比べ移植性の高い方式となっている。LIFERの場合は、対象とするデータベースごとにすべてを書き直さなければならない。XCALIBURのように、対象分野独立の知識 (言語や常識に関する知識) と分野依存の知識 (XCALIBURの場合はVAXファミリーに関する知識) を分離すれば、移植性のある自然言語インターフェイスを設計することができる。しかしながら、両者は互いに関連している部分も多いために、知識の分離は自然言語処理研究のうちの重要な研究目標の一つとなっている。

## 9. 自然言語による仕様からのプログラム合成

本章では、解説の終わりとして、自然言語処理の新たな応用分野について述べる。これは、前章までの計算機のインターフェイスとして自然言語を利用するというものではなく、より積極的な自然言語へのアプローチである。

ソフトウェア危機が叫ばれる現在、その解決法として、自動プログラム合成の研究が盛んに行われている。自動プログラム合成研究の目標の一つに、自然言語からプログラムを直接生成することがあるが、まだ満足のゆく成果は上げられていない。これは、自然言語そのものに内在する問題の他に、仕様の記述形式、すなわち仕様を手続き的あるいは宣言的のいずれによって記述すべきかについて、統一した見解がないということにもよる。手続き的に記述された仕様は、制御構造に関する

る言及が含まれるために、大変読みづらいものになる。また、仕様を記述する際には、ある種のマシンを想定しなければならず、通常のプログラミングと同様の問題が生じる。一方、宣言的な仕様は制御構造を考慮する必要がなく、容易に記述することができる。しかし、常識的に判る対象や動作などは、読み手の常識を仮定して通常記述されないことが多く、仕様に曖昧な部分が含まれる可能性がある。したがって、プログラムを合成する場合は、仕様からユーザの考えている意図を抽出し、様々な知識をうまく適用して、ユーザの考えている処理の道筋を推論し補足してやる必要がある。

我々は後者の立場をとり、知識工学の研究で提案されたプランとゴールの概念(Schank 1977)を導入した自動プログラム合成システムを開発している(藤井 1985,1986)。ゴールは行為の目的を示すもの、プランはゴールを達成するための計画を表すものである。たとえば、「おなかがあつた」という発話に対して、ゴールは「空腹を満たす」ことであり、そのゴールを達成するプランとして「レストランに行く」とか「弁当を食べる」がある。UCにおける意図の抽出もこれと同様の考え方に基づくものである。この考え方をプログラミングに当てはめてみると、ゴールは仕様が内包するユーザの意図、プランはその意図を達成するための手続き(プログラム)に対応している。したがって、仕様からゴールを抽出し、ゴールを達成するためのプランを推論することができれば、最終的に自然言語からプログラムが合成されることになる。

以下、ゴール・プランに基づくプログラム合成の処理手順を概説する。自然言語で記述された仕様は、まず文法、辞書などの「自然言語に関する知識」を用いて、構文・意味解析を行い、文の意味表現を抽出する。ここでは、我々が開発したIP(Integrated Parser)(Uehara 1985,1986)と呼ばれるパーザが用いられている。IPは、いままで述べてきたような意味文法ではなく、統語、意味、文脈処理のすべてを融合したパーザである。次に、得られた意味表現からユーザの意図を推論する。この推論には、「意味表現とゴールの対からなる知識」が用いられる。次に、ゴールを達成するためのプランを推論し、さらに得られたプランからよりプリミティブなプランを推論するというように、段階的にプランを精密化し、実行可能なプラン列に変換していく。このプランニング過程では、ゴールを達成するために必要なプラン列が記述された「プランニングのための知識」が適用される。また、各プランとその前後のプランでデータがうまく受け渡しされるかどうかを確認するために、プランの実行に必要な「前提条件」と、プランの実行後に得られると予想される「処理結果に関する知識」が用いられる。予想される処理結果が続くプランの前提条件を満たさない場合、その前提条件を達成するために必要なプラン列が新たに生成され、プランの直前に付加される。また、仕様中に対象の省略が含まれる場合、このプランニング過程で随時常識や文脈情報を参照しながら補足される。最終的には、最もプリミティブで省略を含まないプラン列が得られる。最後に、個々のプリミティブなプランをそれぞれ対応する目標言語の命令系列に置き換えて、実行可能なプ

プログラムが生成される。ここでは、命令やフォーマットなどの「目標言語に関する知識」とプログラムの「入力形式、出力形式を規定した知識」が用いられる。以上のように、様々な知識を適用し、ユーザの意図を考慮したゴールとプランに基づく柔軟な推論を行うことで、ユーザの要求を満たすプログラムを生成することができる。

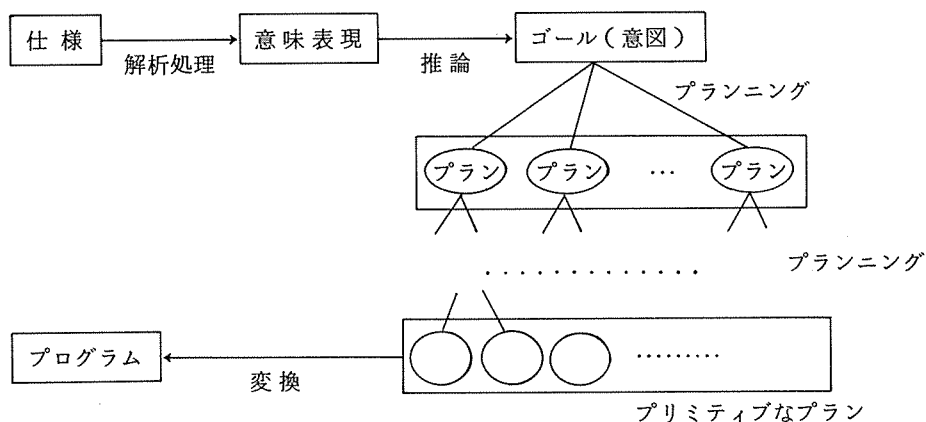


図 9.1 ゴール・プランに基づくプログラム合成の概念図

上記の手法を用いたプログラム合成システムは、自然言語による仕様を中間表現としてプロダクション・ルールの形式に変換する。これは、1) 宣言的な仕様には制御構造が記述されないので、制御は言語側で自動的に行う言語が望ましいこと、2) モジュールリティが良く、修正が容易なこと、3) if-then のルール形式は自然言語文と相性が良いことなどの理由による。さらに、合成されたプロダクション・ルールは、最終的な目標言語の Prolog プログラムに書き換えられる。プロダクション・ルールを中間言語、Prolog を目標言語とすることで、合成するプログラム中に制御構造を埋め込む必要がなくなっている。

また、本システムで用いられる知識は、それぞれ独立した知識源 (Knowledge Source) として捉えられている。各知識源は、個々の知識の集合 (ルール) と、それらの知識をどのように選択するかという選択手続き (メタ・ルール) からなる。個々の知識は、プロダクション・システムの枠組みの中で実行されるが、各知識源の持つ選択手続きで適用される知識が選択され、選択された知識は書き換え規則とみなされて実行される。また、現在の知識源に適用可能な知識が存在しない場合、制御知識源の知識 (メタ・ルール) が適用され、知識源の切り換えが行われる。このように、知識源の適用順序がメタ・ルールの形式で明確に規定されているので、知識源を容易に追加、修正することができる。

本システムの応用対象として三省堂出版の「新コンサイス英和事典」の磁気テープ (長尾 1982)

を原データとした構造化処理を取り上げている。図 9.2 に原データと対応する辞書イメージを示す。

ⓧⓧ\*ac·ceptⓧ[əksépt, -æk-]vt  
 ・1. ⓧを受ける, 受け取ⓧる。ⓧ2. ⓧ引き受ける(地位などを), 受諾する(提案などⓧを), 応じる(招待など○に○)。ⓧ3. ⓧを承認する。ⓧ4. ⓧ(人ⓧを)受け入れる, 仲間○に○入れる。ⓧ5.  
 ・ⓧ【商】引き受けⓧる(手形を)。□ⓧ名acceptance, -  
 acceptation. ⓧⓧAnt. -ⓧreject. -q-She-~ed-  
 his-hand-in ⓧmarriage. 彼○と  
 ○○の○結婚を承諾し○た○。~a-new  
 classmate 転▽入生を仲間○に○入れる。  
 ⓧ~the-person-  
 (face)-of ⓧをひいき○する。ⓧ

磁気テープ上の原データ

\*ac·cept [əksépt, -æk-] vt. -1. を受ける, 受け取る。2. 引き受ける(地位などを), 受諾する(提案などを), 応じる(招待など○に○)。3. を承認する。4. (人)を)受け入れる, 仲間○に○入れる。5. 【商】引き受ける(手形を)。acceptance, -acceptation.  
 Ant. -reject. -q-She-~ed-his-hand-in  
 marriage. 彼○と○○の○結婚を承諾し○た○。~a-new  
 classmate 転▽入生を仲間○に○入れる。  
 ~-the-person-(face)-of をひいき○する。

辞書イメージ

図 9.2 原データと辞書イメージ

構造化処理とは、辞書のイメージがそのまま入力されたデータから括弧類や制御用コードを手掛かりにして、見出し語・発音などの各データ項目を切り離し、見出し語を頂点とする木構造を再構成する処理のことである。このようなプログラムは、手続き型言語で記述すれば、複雑な構造を持ち、サイズのにも2~3,000行になる。さらに、開発には試行錯誤を伴うので、自然言語による仕様、および宣言型言語の利用が有効な事例であると考えられる。現在、日本語で記述された約100文からなる仕様を与えて、構造化処理を行うプログラムの合成を行い、本システムの評価、改良を進めている。図 9.3 に仕様から Prolog プログラムへの変換例を示す。

また、本システムは C-Prolog にウィンドウ表示機能を導入した Window-Prolog 上で実現している。この Window-Prolog は、エジンバラ大学で開発された C-Prolog を、我々が日本 DG 社のスーパーミニコン MV/8000-II に移植し、さらに機能の拡張を施したものである。図 9.4 に辞書構造化処理の実行例を示す。



日本語文：

辞書項目中のボールド体は見出し語を示している。



ゴール：

辞書項目からボールド体を見出し語として抽出する



プラン表現：

辞書項目にボールド体が存在する

--> 辞書項目からボールド体を取り出す  
ボールド体を見出し語として登録する  
辞書項目を更新する



プロダクションルール：

```
get_wm(辞書項目(X)), include(X,[#,Y,Ⓜ])
--> pick_up(X,[#,Y,Ⓜ],Z), delete_wm(辞書項目(X)), add_wm(辞書項目(Z)),
make_pointer(P), add_wm(top=見出し語(Y,P)).
```



最適化されたプロダクションルール：

```
get_wm(辞書項目(X)), pick_up(X,[_,#,Y,Ⓜ|Z])
--> delete_wm(辞書項目(X)), add_wm(辞書項目(Z)),
make_pointer(P), add_wm(top=見出し語(Y,P)).
```



Prolog プログラム：

```
rule(辞書項目(X),WM,Out):-
pick_up(X,[_,#,Y,Ⓜ|Z]),
delete_wm(辞書項目(X)),
add_wm(辞書項目(Z)),
make_pointer(P),
add_wm(top=見出し語(Y,P)).
```

図 9.3 仕様から Prolog プログラムへの変換例

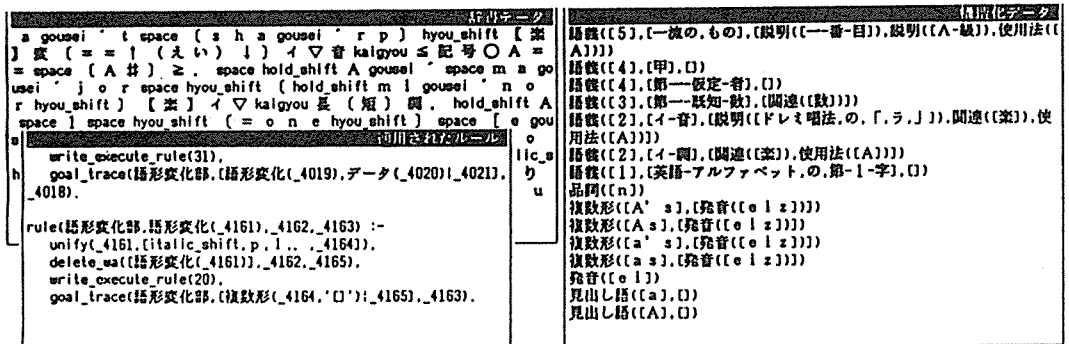


図 9.4 辞書構造化処理の実行例

## 10. 結 論

いわゆる自然言語処理システムは、自然言語処理にまつわる問題を一般的に研究するために開発されたシステムと、実用的な応用を念頭に置いて開発されたシステムの二種類に分類できる。もちろん、後者のシステムも前者の研究によって得られた知見に基づいて開発されたものであるが、言語理論における瑣末な問題についての取り扱いが省略されたものとなっている。ここに実用化を目指すものと、研究のみで終わってしまうものの大きな違いがある。前者についての解説では、現在、自然言語処理の研究において、何が問題で、何に興味を持って研究が進められているかがわからないと思ひ、あえて本小論では後者の立場から解説した。

いずれにせよ、ユーザと計算機の対話において、最も簡単に馴染みやすいものが自然言語である以上、今後もますますこの種の研究が盛んに行われると思われる。

## 謝 辞

本小論の構成は、計算機に全くの素人な妻との会話が参考になった。また技術的なことに関しては中村 孝技官のサーベイ、本学修士学生 垣内 隆志君、堀 雅洋君、一本木 真史君との議論が参考になった。記して感謝する。

## 参考文献

- [Barr 1981] Barr, A. and Feigenbaum, E. A. (eds.): The Handbook of Artificial Intelligence, Vol. 1 & 2, William Kaufmann.
- [Carbonell 1983] Carbonell, J. G., Boggs, W. M., Mauldin, M. L., and Anick, P. G.: The XCALIBUR Project: A Natural Language Interface to Expert Systems, Proc. IJCAI-83, pp. 653-656.
- [Eastman 1981] Eastman, C. M. and McLean, D. S.: On the Need for Parsing Ill-formed Input, Am. J. Comput. Linguist. Vol. 7, No. 4, pp. 257.
- [Eastman 1984] Eastman, C. M. and McLean, D. S.: A Query Corpus Containing Ill-formed Input, Tech. Rpt. 84-CSE-9, Southern Methodist University, Dallas, Texas.
- [藤井 1985] 藤井, 上原, 豊田: 知識を用いた仕様記述からのプラン抽出とその自然言語プログラミングへの応用, 知識情報処理シンポジウム, pp. 57-66.
- [藤井 1986] 藤井, 上原, 豊田: 知識工学的手法を応用した自然言語からの自動プログラム合成システム, プロトタイプと要求定義シンポジウム(発表予定).
- [Hayes 1981] Hayes, P. J. and Carbonell, J. G.: Multi-Strategy Construction-Specific Parsing for Flexible Data Base Query and Update, Proc. IJCAI-81, pp. 432-439.

- [Hendrix 1977] Hendrix, G.G.: Human Engineering for Applied Natural Language Processing, Proc. IJCAI-77, pp.183-191.
- [Hendrix 1978] Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., and Slocum, J.: Developing a Natural Language Interface to Complex Data, ACM Trans. on Database Systems, Vol. 3, No. 2, pp.105-147.
- [井上 1985] 井上, 上原, 豊田: 文法的逸脱を考慮した柔軟な構文解析, 情報処理学会自然言語処理研究会資料 47, pp.9-16.
- [Kaplan 1983] Kaplan, S.J.: Cooperative Responses from a Portable Natural Language Database Query System, in Brady, M. and Berwick, R.C. (eds.) Computational Models of Discourse, The MIT Press, pp.167-208.
- [長尾 1982] 長尾 真: 言語辞書活用のための計算機プログラムシステムの開発と言語辞書の解析, 昭和 55, 56 年度科学研究費補助金試験研究 (1) 研究成果報告書.
- [McDermott 1982] McDermott, J.: XSEL: A Computer Salesperson's Assistant, in Hayes, J., Michie, D., and Pao, Y.H. (eds.) Machine Intelligence, Vol.10, pp.325-337, Ellis Horwood Ltd.
- [Pereira 1980] Pereira, F.C.N. and Warren, D.H.D.: Definite Clause Grammar for Language Analysis-A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artif. Intell. Vol.13, pp.231-278.
- [Schank 1977] Schank, R.C. and Abelson, R.: Scripts, Plans, Goals and Understanding, Lawrence Erlbaum.
- [Shortliffe 1976] Shortliffe, E.H.: Computer-based Medical Consultations: MYCIN, American Elsevier.
- [田中 1981] 田中, 佐藤, 松本, 内田, 横山, 板橋, 三国, 池田, 大谷木, 寺津, 山梨: 自然言語処理技術と言語理論, 電子技術総合研究所調査報告, 電子技術総合研究所.
- [Tennant 1981] Tennant, H.: Natural Language Processing, Petrocelli Books.
- [Uehara 1985] Uehara, K., Ochitani, R., Mikami, O., and Toyoda, J.: An Integrated Parser for Text Understanding: Viewing Parsing as Passing Messages among Actors, in Dahl, V. and Saint-Dizier, P. (eds.) Natural Language Understanding and Logic Programming, pp.79-95.
- [Uehara 1986] Uehara, K., Kakiuchi, T., Mikami, O., and Toyoda, J.: Extended Prolog and Its Application to an Integrated Parser for Text Understanding, Proc. Logic Programming Conference '85, Lecture Notes in Computer Science, Springer-

Verlag (to be published).

[Weizenbaum 1966] Weizenbaum, J.: ELIZA - A Computer Program for the Study of Natural Language Communication Between Man and Machines, Commun. ACM, Vol. 9, No. 1, pp. 36-45.

[Wilensky 1984] Wilensky, R., Arens, Y., and Chin, D.: Talking to UNIX in English: An Overview of UC, Commun. ACM, Vol. 27, No. 6, pp. 574-593.