

PROLOG による 初級日本語データベース

林 文賢

0 始めに

小論は中国人に対する日本語教育に使う例文を集めるために、PROLOGで組んだ日本語データベースである。筆者の計画では、現在市販されている初級、中級、上級などの日本語教科書を収集してきて、一つの教育用日本語データベースにしたかったのだが、色々制限があり、焦点を初級に絞って初級日本語データベースに切り替えたのである。

ところが、一口初級日本語と言っても、最近日本語教育が目覚ましい進展ぶりをみせ、初級教科書で好評を博しているものが幾つもあるから、一回きりでそれらをデータベースにするのも無理なので、蔡茂豊博士の日本語読本第一冊を選んでスタートをきったのである。

周知の通り、言語データの処理によく使うプログラミング言語として、COBOLとBASICが挙げられる。COBOLは大量処理に向いていて、大型コンピュータがその独占場である。国語国立研究所ではCOBOLを用いて言語データの整理、集計、分析しているようである。BASICは初心者向けでパソコンではよく使われている。草薙裕博士の「コンピュータ言語学」⁽¹⁾ではBASICを用いてデータ処理の技法を説明している。

こうした現状の中で、なぜ大量処理に向かないPROLOGで⁽²⁾プログラムを組む

のかということ、理由は三通りある。プログラムの作りやすさ、実行の速さと目的の在りかである。順を追ってわけを述べよう。

まず、プログラムの作りやすさ。データベースは資料を検索するために作ったものである。資料検索を実行するにはどうしてもパターン照合をやらなければならない。問題はこのパターン照合であるが、伝統的なプログラミング言語、例えば、COBOL、BASIC で書くといかに厄介なことであるかは、経験のある方に御分かりだろうと思う。

ところが、PROLOG でパターン照合をやれば、仕事が簡単に済む。それというのは PROLOG にパターン照合の機能が内蔵されているので、ほとんどのパターン照合は PROLOG が自動的にやるのである⁽³⁾。さらに、再帰性という強力なデータ処理能力が PROLOG では簡単に引き出せることもプログラムの作りやすさにあずかって力があるのである⁽⁴⁾。

次に、実行の速さについて。

コンピュータを動かすには、それに指示を与えなければならない。そのコンピュータへの一連の指示のことをプログラムと言う。コンピュータを動かすべく、プログラムを組んでコンピュータに入力するが、コンピュータが動き出す前に人間が与えた指示、つまり、プログラムの意味を理解しなければならない。この点について、「コンピュータ言語学入門」には詳しい説明がしてあるから、そのまま引用させて戴く。なお、引用文のなかの注釈は筆者がつけたものであることをあらかじめ断っておく。

コンピュータになにか処理をさせるには、その処理の手順を一つひとつプログラムにして入力するわけだが、そのプログラムは直接的にコンピュータが理解できるようにしなければならない。

コンピュータは二進法、或いはそれをまとめた十六進法しか理解しない。この二進法、或いは十六進法の数字の組み合わせが機械の操作に対応しているわけで、これを機械語と言う。機械語は（中略）人間にとって非常にあつかいに

くい。

そこで機械語の命令一つひとつに対応するもの数字ではなく記号であらわすようにした「言語」を使うことがある。これをアセンブラ言語という。

しかしアセンブラ言語で書かれたプログラムはコンピュータは直接理解できないので、これを自動的に機械語に翻訳するアセンブラというプログラムを用いる。

ただアセンブラ言語は機械語と対応しているので、コンピュータの機種によって命令や手続きが異なるし、コンピュータのハードウェア⁽⁶⁾の操作も知っていなければならないのでそんなに簡単なものとはいえない。

そこで、自然言語や数値計算の式に近い形の命令が使えるようにしたものを使うことになる。これを高級言語という。FORTRAN（フォートラン）とかCOBOL（コボル）、BASIC（ベーシック）、pascal（パスカル）などの名前を耳にした方も多いだろう。これらがこの高級言語なのである。

高級言語は機械語やアセンブラ言語ほどコンピュータの機種に左右されない。

これらの高級言語はコンパイラあるいはインタプリタと呼ばれるプログラムで機械語に翻訳される。

コンパイラは高級言語で書かれたプログラムをひとまとめにして翻訳して機械語のプログラムにする。そして翻訳が終わってからプログラムの実行を行う。

これに対して、インタプリタは手続きを一つずつ翻訳して、その翻訳通りに実行する。プログラムの中の手続きを解釈しながら実行するので、プログラムにまちがいがあれば直ちに実行が止まってあやまりを表示するので、あたかもコンピュータと対話しながらプログラムを作っていくように感じられる。

長い引用で恐れ入るが、以上で分かるようにコンピュータとプログラム言語、例えば、COBOL、BASIC、PROLOGなどの間に通訳者がいなければ、「意思疎通」ができないのである。そして、通訳者には同時通訳者のインタプリタと逐次通訳

者のコンパイラーがあることもこれで分かる。

さて、PROLOG にもいろいろな方言があり、こちら台湾でよく使われているのは micro-PROLOG と TURBO PROLOG である。

micro-PROLOG はいわゆるインタプリターで、「対話型だから、初心者には扱いやすいが、速度が落ちる」⁽⁷⁾。それに対して、TURBO PROLOG はコンパイラーであり、インタプリターの micro-PROLOG より実行がはるかに速い。Prolog Programming という本によると十倍以上も速いという⁽⁸⁾。

しかし、実行が速いかわりに TURBO PROLOG の利用者も相当の犠牲を払っている。micro-PROLOG では書かないで済むことだが、TURBO PROLOG で組んだプログラムの始めに pascal のように資料タイプやら述語タイプやら面倒な宣言をしなければならないのである。pascal におなじみの方は何とも思わないが、pascal になじみの薄い方はこれに慣れるまでずいぶん時間が掛かる。

micro-PROLOG にせよ、TURBO PROLOG にせよ、両者とも欠点と利点をもっているが、実行の速さを考慮に入れ、筆者が TURBO PROLOG を採用することに踏み切ったのである。

ところで、どうしてこんなに速さを気にするのか。それは情報検索の際、何千もある資料の中で指定のデータを捜し出すには、プログラムの実行が速ければ速いほど好いと思うからである。

PROLOG がひとつの指定項目を見付け出すために、蓄積されたデータの最初から最後まで探し尽くさなければならない。データの蓄積がまだ少ない間は実行が遅くてもそんなに感じられないが、データが膨大に膨れ上がると、実行の遅さに我慢ができなくなるのである。

さて、最後に目的について述べるが、筆者が PROLOG に目を付けたのはその計量能力ではなくして、その蓄積と検出能力なのである。言い換えれば、某の単語が何回使われたかというようなことを一切考慮しない。それよりも、データの蓄積がたやすくできるかどうか、情報検索の際、指定の項目が簡単に引き出せるかどうか、

ということが考慮のポイントなのである。

実をいうと、人工知能言語である PROLOG は TURBO や BASIC などの伝統的なプログラミング言語とは比べものにならないほど計量に弱いものなのである。もっとも、計量のためではなく、文字処理のために作った、新しいプログラミング言語だからであるが、しかし、データの蓄積と検出の面から考えれば、PROLOG は COBOL や BASIC などよりも遥かに優れているのである。

周知のように、データベースに蓄積された日本語の文のながさが一定してない。この可変長データをあつかうには、COBOL や BASIC などの伝統的なプログラミング言語を使うと、面倒な手続きを踏まなければ済まないが、PROLOG で行くと、手を濡らさずに済む。PROLOG 自身、というよりも処理系自身が肩代わりしてそれを処理してくれるのである。

PROLOG にはこういう素晴らしい長所をもっている。これは筆者が伝統的なプログラミング言語を取らずに PROLOG を採用した所以である。小論は不成熟ながら、中国人に対する日本語教育のための日本語データベースに向かって筆者が歩み出した第一歩と言えよう。

1 データ入力

データベースを作る前にまずやらなければならないことはデータの構造を決めることである。以下、行文の便を図り、日本語読本第一冊第十八課を例にして論じていこう。

第十八課は 64 ページ、65 ページと、二ページあり、各ページの始めの文は次のようである。

机の上に何がありますか。(64 ペ)

銀行の左にあるのは煙草屋ですか、果物屋ですか。(65 ペ)

これを次のようなデータ構造にすることが出来る。

018 064 tukue no ue ni nani ga arimasu ka.

018 065 ginkoo no hidari ni aru nowa tabakoya desu ka, kudamonoya desu ka.

つまり、課、ページ、文の順序である。このデータ構造を PROLOG で表せば、次のようになる。

```
sentence ( " 018 " , " 064 " , [ " tukue " , " no " , " ue " , " ni " , " nani " , " ga " ,  
      " arimasu " , " ka " ] )
```

```
sentence ( " 018 " , " 065 " , [ " ginkoo " , " no " , " hidari " , " ni " , " aru " , " now  
a " , " tabakoya " , " desu " " ka " , " , " , " kudamonoya " , " desu " , " ka " , "  
      . " ] ) (10)
```

sentence は、PROLOG の述語であり、引数を三項もっている。第一項は課を意味している「" 018 "」で、第二項はページを表している「" 064 "」（二番目の sentence について言えば、「" 065 "」であるが）である。そして、第三項は [] で閉じているが、これはリストという意味で、そのリストの要素は " " の中に入っているローマ字である。

データ構造を決めてから、さて、今度はデータの入力である。データベースのデータは別のところで PE II などのエディタで作り、ファイルとして PROLOG に入力してもいいのだが、上の sentence とい述語の構造を見ても分かるように、TURBO PROLOG はほかの PROLOG よりも「" "」を使用している。同じ構造を micro-PROLOG で書けば次のようになる。

```
(sentence 018 064 tukue no ue ni nani ga arimasu ka.)
```

```
(sentence (018 065 ginkoo no hidari ni aru nowa tabakoya desu ka , ku  
      da-monoya desu ka . )
```

この構造は TURBO PROLOG のそれよりすっきりしているが、前にも述べたように micro-PROLOG の速度は我慢できないほど遅い。TURBO PROLOG の速度の良さを発揮させるために、その構造の弱みをなんとかしなければならない。そこで

考え出したのは次の方法である。

方法 画面の指示に従い、端末から " " という記号なしの文を入力し、PRO LOG が自動的に所定の構造に変えてくれるプログラムを組む。

次の enter はこの方法の産物である。

enter:—

```
write ( " \n Which lesson? : " ) ,  
readln (Lesson) ,  
write ( " \n Which Page ? : " ) ,  
readln (Page) ,  
sentences (Lesson, Page, List) .
```

sentences (Lesson, Page, List) :—

```
write ( " \n Please type sentence : \n ) ,  
readln (Sentence1) ,  
concat (Sentence1, " / " ,Sentence) ,  
parse (Sentence,List) ,  
reverse (List,Tsil) ,  
assert (sentence (Lesson,Page,Tsil) ) ,  
write ( " \n more ? (y/n) : " ) ,  
readln (A) ,A=y,enter.
```

reverse (L1,L3) :—

```
reverse2 (L1,[ ],L3) .
```

reverse2 ([],L,L) .

reverse2 ([X|L],L2,L3) :—

```
reverse2 (L,[X|L2],L3) .
```

parse (S1,L1) :—

```
parse1 (S1,[ ],L1) .
```

```

parse1 (S1,L1,L2) .
    next-word (S1,S2,W) ,
    parse1 (S2,[W|L1],L2).parse1 (S,L,L) :-
    forntchar (S,Ch,-) ,
    Ch='/' .

```

```

next-word (S,S2,W) :-
    find-delim (S,Count,0) ,
    Count>0,
    frontstr (Count,S,W,S3) ,
    stripe-space (S3,S2) .

```

```

find-delim (S,Count,C) :-
    forntchar (S,Ch,S2) ,
    Ch <> ' ' ,Ch <> ' / ' ,
    C2=C+1,
    find-delim (S2,Count,C2) .

```

```

strip-space (S,S2) : -
    frontchar (S,Ch,S2) ,
    Ch=

```

```

strip-space (S,S) .

```

この述語を実行すると、画面に

Which lesson? :

というメッセージが現れる。そして、端末から

018

と打ち込めば、

Which page? :

と、メッセージがでてくる。それに

と答えると、PROLOG から

Please type a sentence

と、またメッセージ。今度は

tukue no ue ni nani ga arimasu ka.

のような文を入力すれば TURBO PROLOG は自動的に所定のデータ構造に変えてくれるのである。

2 データ検出

PROLOG ではデータ検出がリストの操作にかかわっている。データベースから指定項目に合う文を見付け出すことはあるリストに指定項目があるかどうかをチェックすることに等しい。指定項目が含まれているリストを捜し出し、それを普通の文に変えるのは、PROLOG でのデータ検出のキーポイントである。

あるリストに指定項目があるかどうかを調べるのに三つの方法が考えられる。

- 1 指定項目を要素にし、この要素がリストに入っているかどうか。
- 2 指定項目をリストにし、それがあるリストの部分リスト (sublist) であるかどうか。
- 3 指定項目をリストにし、このリストを集合と見立て、それがあるリスト (これも一つの集合と見なすが) の部分集合であるかどうか。

PROLOG で方法 1 表せば次のようになる。

```
member (A,[A|_]) : - !.
```

```
member (A,[_|B]) : -
```

```
member (A,B)
```

この member という述語は、A が単語、しかもただ一つの単語であれば成功するが、一つ以上の単語なら失敗する⁽¹¹⁾。一つ以上の単語でも失敗させないようにするに

は、方法 2 を使わなければならない。次の sublist は方法 2 で組んだ述語である。

sublist (A,B) : -

append (C,D,B) ,

append (A,C,D) ,

append ([],L,L) .

append ([H1|L1],L2,[H1|H3]) :-

append (L1,L2,L3) .

しかし、sublit は A が順序あるリストであれば成功するが、順序あるリストでなければ失敗する。これをうまく解決するには A を集合と見なければならない。

subset という述語は方法 3 を遺憾なく表している。

subset ([],[]) .

subset ([First|Sub],[First|Rest]) : - ! ,

subset (Sub,Rest) .

subset (Sub,[_|Rest]) : -

subset (Sub,Rest) .

さて、目的が違えば方法も違うから、この三つの方法の中でどれが一番良いかは一概にはいえないが、かかり結びを考慮に入れ、筆者が方法 3 に軍配を上げたのである。以下の serch は方法 3 で書いた述語である。

serch:-

write (" Which word/string/sentence ? : ") ,

readln (Word) ,

str-to-list (Word,Subset) ,

search1 (Subset) .

search1 (Subset) : -

sentence (L,P,Set) ,

subset (Subset,Set) ,

```

write (T, " ", L), nl,
write ( " " ), writelist (Set) ,
nl, fail.

writelist ([ ]) : - write ( " \n " )

writelist ([H|L]) : -
    write (H, " " ), writelist (L) .

str-to-list (S,[H|T]) : -
    fronttoken (S,H,S1) , str-to-list (S1,T)

str-to-list (-,[ ]) .

```

この述語を実行すると、画面に

Which word/string/sentence ? :

というメッセージが現れ、そのあとに探したい項目を入力し、述語 search が自動的にその項目をリストに転換してくれる。そして、転換されたリストを search1 に送り出す。

search1 は search から送られてきたリスを引数としてもらって、パターン照合の機能を発揮し、sentence というデータ構造に合った文を引き出す。そして、その文を subset に送り出し、テストする。もし、subset が成功すれば、指定項目に合った文を画面に打ち出す。それから fail という組み込み述語で強引に失敗させ、バックトラッキングを引き起こす。もし、subset が失敗すれば、自動的にバックトラッキングが実行される。バックトラッキングが起こると、sentence という述語のお陰で新しいデーターを引き出し、sebsset に送り、またテストさせる。このように generate and test を繰り返し、データベースの最後のデータまで検索を行うのである。

3 プログラムとその説明

以下、データ入力とデータ検出の述語を一本のプログラムにまとめ、JPNDBASE

。PROと命名し、これについての説明はプログラムの後ろに付する。なお、参考としてJPANDBASE、PROとの対話の断片と、第十八課全課のデータベースも合わせ載せる。(付録を参照)

nowarings文、domains文、database文、predicates文などがいわゆるTURBO PROLOGの宣言である。goal文は内的目標を設定するものである。この場合の内的目標はstart文である。TURBO PROLOGのには内的目標と外的目標を設定することが出来る。内的目標が設定されるとコンパイラしてから、内的目標があるためプログラム自身が自動的に実行に走る。内的目標がない場合、外的目標の設定と見なされ、コンパイラが終われば、画面にgoalというプロンプト(prompt)が現れ、端末から目標の入力を待つようになっている。

clauses文はいわゆるプログラムの本文である。

startの定義は三つからなっている。初めのstartは画面を白紙に戻してから、existfileという組み込み述語で18.datというファイルが存在するかどうかを調べる。もし存在すれば、それを作業場所に卸したあと、failで強引に失敗させ、二番目のstartに走らせる。もし存在しなければ、existfileが失敗し、二番目のstartに走る。

二番目のstartは、データを端末から入力するかどうかを利用者に聞く述語である。入力するならば、"y"と入力し、enterが触発される。enterで入力を終え、18.datというファイルに保存してから、choiceに行く。もし、入力しない場合、"y"以外の文字を入力し、二番目のstart全体が失敗して、三番目のstartに飛んで行く。

三番目のstartは二つの述語からなっている。ひとつはchoiceで、今一つはpurgeである。これはちょうど二番目のstartの後ろの二つ述語と同じである。choiceは検索するか終了するかを利用者に選択させる述語である。purgeはいままで作業場所に卸した、あるいは端末から入力したデータを皆無にするものである。つまり、作業場所を白紙に戻すのである。purgeがない場合、あとでもう一度プログラムを実行すれ

ば、データが重なる可能性があるから、purgeで片付けるわけである。choiceが作動すると画面に、

search or quit ? (s/q) :

といメッセージが現れる。端末から利用者の入力を待つのである。そして、利用者の入力を readln で読み込み、check でチェックするわけである。

check は三つからなっている。端末からの入力がある場合、はじめの check が触発され、search に突っ走る。端末から "q" が入力された場合、check 全体が成功し、purge に差し掛かる。purge は前述した通り作業をきれいに掃除するものである。

もし、端末からの入力が "s" でもなければ、"q" でもない場合、三番目の check が触発される。まず鈴を鳴らし、利用者の注意を喚起する。そして、画面に

Please make choice again.

とメッセージを送り、もう一度 check に戻る。

さて、先程の enter は、前節のデータ入力のところでおおづかの流れについてのべたが、ここでは深入りして説明する。

前節で述べたように、enter は端末から入力出来るようになっている述語である。まず、端末から何課、何ページの情報を読み込み、sentences に渡す。

sentences は端末から入力した文の後ろに contact で行末記号である " / " を付けて parse に送り出す。仮に

tukue no ue ni nani ga arimasu ka .

という文が入力されたとする。contact が掛かり、文の最後に " / " が付けられ、

tukue no ue ni nani ga arimasu ka ./

という文になる。この文はまた parse に扱われ、順序の逆転した

[".","ka","arimasu","ga","nani","ni","ue","no","tukue"]

のようなリストになるから、reverse で元の順序に転換する。仮に文を入力する前に、端末からすでに 018 と 064 が入力してあるとする。018 は第十八課を表し、064

は64 ページを意味している。そうすると、assert で

```
sentence (018", "064", ["tukue", "no", "ue", "ni", "nani", "ga", "arimasu", "ka",  
    "."])
```

のようなデータ構造の文が TURBO PROLOG の database に加入される。入力した文が database に入ってから、画面に

```
more ? (y/n) :
```

と、またデータを入力するかどうかを尋ねるメッセージが出現する。"y" 答えたら、enter が繰り返される。"n" と入力したら、sentence が失敗し、そのため、トップレベルの enter も巻き添えを食い、失敗してしまう。しかし、二番目の start は enter の前に not が来ているから、失敗せずに後ろの述語が触発される。それは not という組み込み述語が、成功した述語を失敗させ、失敗した述語を成功させる性格をもっているので、この notのおかげで一度失敗した enter が成功に復活したのである。そして、enter で入力したすべての文が save で 18.dat というファイルに保存されてから、choice が触発される。

さて、なぜ入力した文の後ろに行末記号 " / " を付けたのかと言うと、parse のための工夫である。

parse は入力した文をトークン (token、単語か記号に当たる) に分解する述語である。分解している際、文が終わりに来たかどうか、parse に知らせるために " / " を付けたのである。この、入力したをトークンに分解するやり方は、人工知能でコンピュータが自然言語を理解するのによく使う手法である。

parse を成功させるには、parse1 という次目標を達成すれば良い。parse1 は parse より余計に補助引数 [] をもっている。これは文をトークンに分解し、そのトークンを集めるためのリストである。

一番目の parse1 は入力した文を next-word で一単語一単語切り出し、リストである第二引数に入れる。もし、文の行末記号 " / " を切り出した場合、一番目の parse1 が失敗し、二番目の parse1 が触発される。

二番目の parse 1 は " / " を切り出したとき、いままでリストである第二引数に入れたトークンを一挙に第三引数に引き渡す。前にも述べたように、引き渡されたトークンの順序は顛倒になっているので、reverse で正常の順序に変えるのである。

ここで、先程「一番目の parse 1 は入力した文を next-word で一単語一単語切り出」すと言ったが、next-word を見てみると、実際は一単語一単語を切り出すのではなくして、一文字一文字切り出すのである。

next-word は、find-delim で入力した文から文字を順に一つずつ読み込む。単語の境界である空白 " " を読み込んだら、それを無視していままで読み込んだ文字を一トークンと見なす。この過程は行末記号 " / " が来るまで繰り返されるのである。

さて、前述したように、入力されたデータは

```
sentence ( " 018 " , " 064 " , " [ " tukue " , " no " , " ue " , " ni " , " nani " , " ga " ,  
          " arimasu " , " ka " , " . " ] )
```

のような形になっている。このようなデータ構造の文を検索するのに search を触発しなければならない。

search が作動すると画面に

```
Which word/string/sentence ? :
```

というメッセージを打ち出し、端末からの入力を待つのである。しかし、検索したいものが単語にせよ、文字列にせよ、文にせよ、端末から入力するとリストの形になっていないから、このまま検索に掛かると失敗するばかりである。これを失敗させないようにするには、入力したものをリストにすればよい。str-to-list はこの役割を担うのである。

端末から入力した文を str-to-list でリストにし、そのリストを search 1 の引数として送り出した後のプロセスは、前節に詳しい説明があるから、ここでは省略する。

以上でプログラムの説明を終わりにする。

4 終わりに

始めに述べたように、小論はデータ入力とデータ検出から構成されている。データ入力のトップレベルの述語は `enter` であり、データ検出のトップレベルの述語は `search` である。`enter` は確かに端末から入力できる利点をもっているが、TURBO PROLOG のデータ構造の制限でせいかく `enter` で構築したファイルは伝統的なプログラミング言語では使いにくい。この点、解決方法として共用ファイルの技法が考えられる。即ち、PE II などのエディタで PROLOG にも COBOL にも BASIC にも使えるようなファイルを作る。それを工夫して PROLOG のデータベースにする。下の `start` という述語はこの働きをする。

```
start : -
```

```
    write ( " Which file to write to ? : " ) ,  
    readln (Filename) ,  
    openwrite (input,Filename) ,  
    write ( " Which file to write from ? : " ) ,  
    readln (File) ,  
    openread (input,File) ,  
    clearwindow,process.
```

```
process : -
```

```
    readdevice (input) ,  
    not (eof (input) ,  
    readln (Sentence) ,  
    parse (Sentence,List) ,  
    reverse (List,Tsil) ,
```



```

do (Tsil,A,B,List2) ,
writedevise (input) ,
write ( " sentence ( " , " \34 " ,A, " \34 " , " , " , " \34 " ,B, " \34 " , "
, " ,List2, " ) " ) ,nl,readdevice (input) ,process.

process : -
writedevise (screen) ,
write ("The data has been sended ! " ) .

do ([A,B|List],A,B,list)

```

この述語を実行すると PE II で作った

018 064 tukue no ue ni nani ga arimasu ka.

018 065 ginkoo no hidari ni aru nowa tabakoya desu ka, kudamonoya desu
ka.

のような文を自動的に

```

sentence ( " 018 " , " 064 " , [ " tukue " , " no " , " ue " , " ni " , " nani " , " ga " ,
" arimasu " , " ka " ] )
sentence ( " 018 " , " 065 " , [ " ginkoo " , " no " , " hidari " , " ni " , " aru " , " now
a " , " tabakoya " , " desu " " ka " , " , " , " kudamonoya " , " desu "
, " ka " , " . " ] )

```

と TRBO PROLOG のデータ構造に合った文を変える。

ところで、enter だけ欠点をもっている訳ではなく、search という述語も完璧とは言えない。単語、文字列、文など検索が出来るが、全ページ、あるいは全課の文を一括して検索出来ない。また、検索する際、指定項目がない場合、search が全然反応を示さない。これらはミスと言わざるを得ない。記して後日の修正を俟つ。

(1) 草薙裕 (1983) を参照。

(2) 萩野綱男・古田啓 (1983) 113 ペ。

- (3) 『TURBO PROLOG USER'S GUIDE』 6 ペ。
- (4) 注 3 同。
- (5) コンピュータの機械装置のことを総合してハードウェアという。『コンピュータ言語学入門』 11 ペ を参照。
- (6) この点から言うと、PROLOG も高級言語の一種である。
- (7) 草薙裕、前掲書、13 ペ。
- (8) Claudia, M. (1986) 52 ペ。
- (9) Michael, A. Donald, N. and Andre', V. (1987) 458 ペ。
- (10) 筆者が使っているパソコンは宏碁の MPF-PC 55 XT である。
日本語の文字が表せないので、ローマ字で表すしかない。
- (11) 成功と失敗は PROLOG の術語である。成功は述語がうまく行けることを意味し、失敗はその反対である。

参考文献

草薙裕 (1983) 『コンピュータ言語学入門』、大修館。

萩野綱男・古田啓 (1983) 「人文系研究のための言語データ処理入門」『朝倉日本語新講座 5 運用 I』

Claudia, M. (1986) 『Prolog programming』.

Michael, A. Donald, N. and Andre', V. (1987) 『Prolog Programming in depth』.

『TURBO PROLOG USER'S GUIDE』.

付 録

/*JPNDBASE.PRO*/

nowarnings

domains

list = symbol*

database

sentence(symbol,symbol,list)

predicates

start

enter

sentences(symbol,symbol,list)

writelst(list)

choice

check(symbol)

search

str_to_list(string,list)

search1(list)

reverse(list,list)

reverse2(list,list,list)

purge

subset(list,list)

next_word(string,string,symbol)

find_delim(string,integer,integer)

parse(string,list)

strip_space(string,string)

parsel(string,list,list)

goal

start.

clauses

start:-

clearwindow,existfile("018.dat"),
write("** Loading database **"),nl,
write("** Please waiting **"),nl,
consult("018.dat"),fail.

start:-

write("\n enter data ? (y/n) : "),
readln(A),A=y,not(enter),
save("018.dat"),!,choice,purge.

start:-

choice,purge.

choice:-

nl,write("\n search or quit ?(s/q) : "),readln(Q),check(Q).

check(Q):-

Q=s,search,!,

check(Q):-

Q=q,!,

```

check(Q):-
    beep,
    nl,write("Please make choice again."),choice.
enter:-
    write("\n Which lesson ? : "),nl,
    readln(Lesson),
    write("\n Which page ? : "),nl,
    readln(Page),
    sentences(Lesson,Page,List).
sentences(T,L,List):-
    write("\n please type sentence : \n"),
    readln(Sentence1),concat(Sentence1,"/",Sentence),
    parse(Sentence,List),
    reverse(List,Ts1),
    assert(sentence(T,L,Ts1)).
    write("\n more ? (y/n) : "),
    readln(A),A=y,enter.
writelst([]):-write("\n").
writelst([H|L]):-
    write(H," "),writelst(L).
search:-
    write("\n which word/string/sentence ?:\n "),
    nl,readln(Word),nl,
    str_to_list(Word,List),
    search1(List).
search1(Subset):-
    sentence(T,L,Set),
    subset(Subset,Set),
    write(" ".T," ".L),nl,write(" "),
    writelst(Set),nl,fail.
search1(X):-
    nl,write("\n continue ? (y/n) : "),
    readln(Q),Q=y,search.
search1(X):-
    write("\n Press the space bar \n"),
    readchar(_).
str_to_list(S,[H|T]):-
    fronttoken(S,H,S1),str_to_list(S1,T).
str_to_list(_,[]).
reverse(L1,L3):-
    reverse2(L1,[],L3).
reverse2([],L,L).
reverse2([X|L],L2,L3):-
    reverse2(L,[X|L2],L3).
purge:-
    write("\n ** Purgeing ! \n"),
    write("\n ** Please waiting !"),
    nl,retract(sentence(_,_,_)),fail.
purge:-
    write("\n The data in workplace\n").

```

```

        write("\n has been cleaned !! \n").
subset([], []).
subset([First|Sub], [First|Rest]) :- !,
    subset(Sub, Rest).
subset(Sub, [_|Rest]) :-
    subset(Sub, Rest).
parse(S1, L1) :-
    parse1(S1, [], L1).
parse1(S1, L1, L2) :-
    next_word(S1, S2, W),
    parse1(S2, [W|L1], L2).
parse1(S, L, L) :-
    frontchar(S, Ch, _), Ch = '/'.
next_word(S, S2, W) :-
    find_delim(S, Count, 0), !,
    Count > 0,
    frontstr(Count, S, W, S3),
    strip_space(S3, S2).
find_delim(S, Count, C) :-
    frontchar(S, Ch, S2),
    Ch <> ' ', Ch <> '/',
    C2 = C + 1,
    find_delim(S2, Count, C2).
find_delim(_, Count, Count).
strip_space(S, S2) :-
    frontchar(S, Ch, S2), Ch = ' '.
strip_space(S, S).
/* 18.dat */

sentence("018", "064", ["tukue", "no", "ue", "ni", "nani", "ga", "arimasu", "ka", "."])
sentence("018", "065", ["ginkoo", "no", "hidari", "ni", "aru", "nowa", "tabakoya", "desu",
"ka", ".", "kudamonoya", "desu", "ka", "."])
/* PROLOG との対話 */

enter data ? (y/n) : y
Which lesson ? :
018
Which page ? :
064
Please type sentence :
tukue no ue ni nani ga arimasu ka .
more ? (y/n) : y
Which lesson ? :
018
Which page ? :
065
please type sentence :
ginkoo no hidari ni aru nowa tabakoya desu ka , kudamonoya desu ka .
more (y/n) ? : n

```

search or quit ?(s/q) : s
Which word /string/sentence ?:

no

018 064

tukue no ue ni nani ga arimasu ka .

018 065

ginkoo no hidari ni aru nowa tabakoya desu ka , kudamonoya desu ka .

continue ? (y/n) : y
which word /string/sentence ?:

ga

018 064

tukue no ue ni nani ga arimasu ka .

continue ? (y/n) : y
which word /string/sentence ?:

desu ka

018 065

ginkoo no hidari ni aru nowa tabakoya desu ka , kudamonoya desu ka .

continue ? (y/n) : y
which word /string/sentence ?:

ni

018 064

tukue no ue ni nani ga arimasu ka .

018 065

ginkoo no hidari ni aru nowa tabakoya desu ka , kudamonoya desu ka .

continue ? (y/n) : n
Press the space bar