LIFT (Language Interface Facility by Translator-generator) による簡単な言語インターフェースの構成

中川貴え（日立）　桃内佳雄、宮本衛市（北大工）

1. まえがき

LIFT (Language Interface Facility by Translator-generator) は言語インターフェースを作成するためのシステムである。言語インターフェース仕様定義をLIFTに入力することによって、LIFTはその仕様に基づいて言語の変換を行う言語インターフェースを構成する。LIFTは言語インターフェース仕様定義処理部、スキャナー、パーザー／トランスレーター、エリプシスから構成される。LIFTは G.G. Hendrix によって作成されたLIFER (Language Interface Facility with Ellipsis and Recursion) を参考にして作成された。意味文法を基礎とした仕様の形式的記述、ATNを基礎とした構文解析の機構をとりいれている。LIFTはデータベースへの検索、質問応答システムへの質問などにおける、簡単な自然言語のためのインターフェース、データの編集とファイルへの格納のためのインターフェースなどの構成のための実用的かつ基本的な道具であると考えられる。本報告はLIFTの構成、LIFTによる簡単な言語インターフェースの構成、データエディターの基本機能について述べる。LIFTは PL/I言語で書かれている。

2. LIFT

(1) 概要

図.1.がLIFTの概略構成図である。言語インターフェース仕様定義は、言語変換仕様定義、区切り子定義、フラグ定義とから構成される。それぞれ言語変換仕様定義処理部、区切り子定義処理部、フラグ定義処理部により処理される。言語変換仕様定義処理部は、言語変換仕様定義入力をトランジション・トリーに変換格納する。フラグ定義処理部は、フラグ定義入力をフラグに変換格納する。区切り子定義処理部は、区切り子定義入力を区切り子リストに変換格納する。言語インターフェース仕様定義をLIFTに入力することにより、その仕様による言語インターフェースが構成される。言語インターフェース仕様定義により定義される入力言語に属する入力文がその言語インターフェースにより処理される。入力文はスキャナー、パーザー／トランスレーター、エリプシスによる処理をへて出力文として出力される。スキャナーは区切り子リストを参照しつつ入力文の文字列に区切りを与える。パーザー／トランスレーターはトランジション・トリーを参照しつつトップダウン、左から右へ入力文の構文解析および出力文への変換処理を行う。エリプシスは入力文における省略の処理をスタックに格納されている前文に関する情報を参照しつつ行う。各処理におい

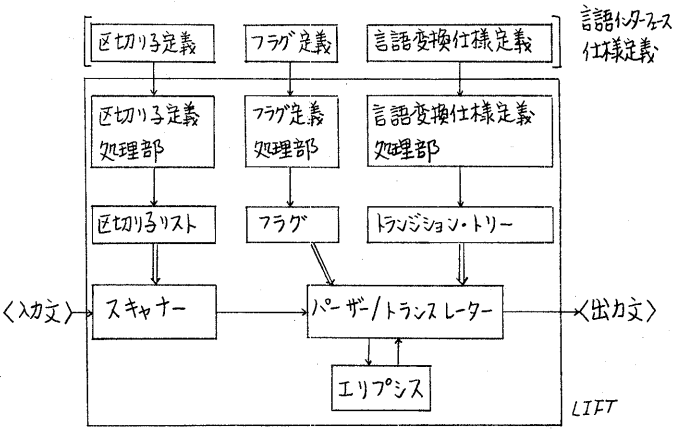


図.1.

て必要に応じてフラグが参照される。

（2）言語インターフェース仕様定義

(イ) 言語変換仕様定義：言語変換仕様を定義するために4コのコマンド、PD、MS、MP、JPが用意されている。

i) PDコマンド：　PD: *metasymbol* ⇒ *pattern* | *expression* ;

metasymbol ⇒ *pattern*を構文部、*expression*を意味部とよぶ。構文部で入力言語の構文規則および意味規則、意味部で出力言語への変換規則を表現する。構文部は意味文法における書き換え規則に相当する。*metasymbol*は非終端記号のアトムである。*pattern*はアトムの並びである。*expression*はアトム、リストあるいは関数式である。関数式において用いることのできる関数として CONS、LIST、FLAT などがあり、引数はアトムかリストである。FLATは、アトムである引数はそのアトムを、アトムでない引数はその CAR 部を引数とした LISTに等価である。*expression*の評価値が*metasymbol*の値となる。

ii) MSコマンド：　MS: *metasymbol* ⇒ *list-of-terminals* ;

終端記号を生成する規則である。*list-of-terminals* はリストであり、その要素はアトムまたはアトムのドット対またはアトムとリストのドット対である。*metasymbol*の値は要素がアトムの場合はアトムそのもの、ドット対の場合は CDR 部のアトムかリストである。*metasymbol* はPDコマンドで定義されている規則の中の*metasymbol*と重複してはならない。

iii) MPコマンド：　MP: *metasymbol* ⇒ *predicate-function* ;

*predicate-function*は述語である。*metasymbol*の値は処理中のアトムに関してその述語が成り立つ場合はそのアトム、成り立たない場合は *nil* である。述語として次のようなものが用意されている。ATOM、NOATOM、ALPHA、NUMBERP、EOR。ALPHAはアトムがアルファベットのみからなるとき真、EORは処理すべき入力が入力バッファにあるとき真となる述語である。

iv) JPコマンド：　JP: *metasymbol* ⇒ *predicate-function* ;

MPコマンドと同様の働きをする。異なる点は、述語が成り立つ場合でも入力文の処理位置が進まないことである。

言語変換仕様定義における注意：PDコマンドにより定義される仕様の構文部における左回帰性を禁止している。トランジション・トリーの構成はPDコマンドによる入力順に行われる。

(ロ) 区切り子定義：区切り子定義のために2コのコマンド、SCGEN、SCINTが用意されている。

i) SCGENコマンド：　SCGEN: ;

PD、MSコマンドによる言語変換仕様定義中の終端記号を区切り子とする。終端記号と非終端記号の区別は非終端記号を〈〉でくくることにより行われる。

ii) SCINTコマンド：　SCINT: *delimitters* ;

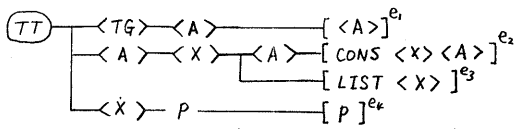
*delimitters*はアトムの並びである。それらのアトムが区切り子となる。

(ハ) フラグ定義：フラグ定義のために10コのコマンド、MSFLG、NOMSFLG、EFLG、NOEFLG、MGFLG、NOMGFLG、STFLG、NOSTFLG、BTFLG、NOBTFLG が用意されている。それぞれ、MSフラグ、Eフラグ、MGフラグ、STフラグ、BTフラグを、1、0にする。1の時の各フラグの意味は次のようである。

MS：区切り子としての定義にMSコマンドに現われる終端記号を加える。

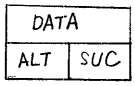
E：省略処理を行う。
MG：MSコマンドで同じmetasymbolによる定義が行われた時は併合する。
ST：構文解析時の処理情報を出力する。
BT：バックトラック処理を行う。
(ニ) その他のコマンド
i) BYE コマンド：　BYE：　;
LIFTの処理終了のコマンド。終了時には、言語インターフェース仕様および記憶領域の使用状況などに関する情報を出力する。
(3) 言語インターフェース仕様定義処理部について
区切り子定義はSCGEN、SCINTコマンドで行われるが、区切り子定義処理部は、SCINTコマンドに対しては定義された順番に、SCGENコマンドに対しては文字列としての長さの順に区切り子リストに区切り子を格納する。
言語変換仕様定義処理部が作成するトランジション・トリーの例を次に示す。

(例)
```
PD : <TG> ⇒ <A> | <A> ;
PD : <A>  ⇒ <X> <A> | CONS <X> <A> ;
PD : <A>  ⇒ <X> | LIST <X> ;
PD : <X>  ⇒ P | P ;
```
言語変換仕様定義

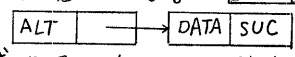


トランジション・トリー

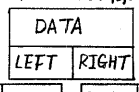
トランジション・トリーは次のようなノードによって表現される。



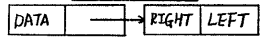
実際には、トランジション・トリーはリストによって実現される。したがって、ノードは次のように実現される。



リストを構成するセルは2成分からなり、セル領域はアレイとしてとられている。セル領域のアレイは、0以下のインデクスをもつ。
言語変換仕様定義処理部は、PDコマンドに対応してpdリストを構成する。これはトランジション・トリーである。MS、MP、JPコマンドのそれぞれに対応してms, mp、jpリストを構成し、これらのリストは入力文の処理においてパーザー/トランスレーターにより用いられる。
(4) スキャナーについて
スキャナーは文字列としての入力文に区切り子リストにもとづいて区切りを与え、その区切られた入力文を入力文リストに格納する。␣、;、(は無条件に区切り子とされ、;はコマンド単位の区切りとして用いられる。
区切り子リスト、入力文リストなどのリストを構成するセルは2成分からなり、セル領域はアレイとしてとられている。文字列そのものは文字列テーブルに格納される。アトムの領域もアレイとしてとられている。アトム領域のアレイは0以下のインデクスをとつ。アトムは、探索の速度をあげるために印字名の辞書引順に二分木に格納される。この二分木はトランジション・トリーのノードと同様に、次のようなノードによって表現される。



実際には、トランジション・トリーと同様に二分木はリストによって実現される。したがって、ノードは次のように実現される。



(5) パーザー/トランスレーターについて
パーザー/トランスレーターは、言語インターフェース仕様定義入力から得ら

れたトランジション・トリーおよびms、mp、ipなどのリストを用いて、入力文の構文解析と出力文への変換処理を行う。トランジション・トリーと入力文リストとをつきあわせて、トップダウンに構文解析を進める。入力言語に対する構文部の仕様が非決定性を含む場合にはバックトラック処理を行うこともできる。パーザー/トランスレーターはトップダウン、左から右へ処理を進めるもので、W. A. Woodsによって提案されたATNを基盤とするものである。

metasymbol ⇒ pattern | expression という言語変換仕様において、patternの各構成要素の構文解析および評価が終了した時点で、それらの評価の結果を用いて、expressionの評価が行われその結果がmetasymbolの値となる。与えられた入力文に対してその構成要素について次々と構文解析と評価が進められ、最終的に開始記号であるmetasymbolに対するexpressionの評価の値が出力文となる。

（6）エリプシスについて

自然言語が入力言語であるような言語インターフェースにおいては、前文と重複する部分を省略することはよく行われる現象である。LIFTにおいては、Eフラグが1の時、パーザー/トランスレーターが入力文の処理に失敗すると、前文の処理後格納してあった、前文の構文のトップレベルでの構成成分に関する情報を用いて入力文の省略処理を行う。省略部分を復元し、再びパーザー/トランスレーターに処理をまかせる。復元できない場合はその入力文の処理は失敗である。

前文の図.2.のような情報がスタックDHSに格納されている。

処理中の省略文を $m_1, m_2, \cdots, m_\ell$ により構文解析する。今、m_{i-1}まで処理が終り、m_iの処理に入るとする。処理を終った入力列をI_{i-1}、未処理の入力列をI_iとする。m_iで構文解析する。失敗した時は、S_iが省略されているとみなし、I_{i-1}にS_iを付加してI_iをI_{i+1}としI_{i-1}にS_iを付加したものをI_iとしてm_{i+1}の処理に移る。成功した時は、I_iからS_iを削除したものをI_{i+1}としI_iにS_iを付加したものをI_iとしてm_{i+1}の処理へ移る。ただし$I_0 = \varepsilon$。$I_{\ell+1}$がεでないときは省略処理は失敗である。

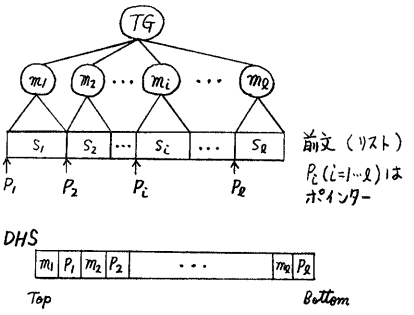


図.2.

3. データエディター

LIFTを基盤としたデータエディターの構成について検討する。データエディタを一つの言語インターフェースとしてとらえる。データエディタの機能として次のようなものが考えられる。

① データのエラーの検出および修正
② データモデルの基本的単位の形成
③ 自動インデキシング、インバーテッドファイルなどの作成

①はデータエディタの基本的機能である。これは言語変換仕様に適合しない入力文に対してその適合しない個所を指示し修正を加えて再び処理を進めるというパーザーの機能としてとらえられる。

②、③の機能は言語変換仕様の意味部において、一部記述されるものであろう。データの内容やデータを格納するファイルの構成にどこまで介入するかは、言語変換仕様の与え方に依存する。また、記述能力にも依存する。

データエディターの言語変換仕様の構文部の規則は、LIFTのパーザーはトップダウン、左から右への処理方式であるために、エラーの検出、修正を対話的に行うためには、決定的でなければならない。
データエディタのエラー修正コマンドとして次の3コのコマンドが用意されている。 REPLACE atom , DELETE , INSERT atom 。すなわち、データのエラー発生の場所で、誤っているとみなされるアトムの置換、削除を行ったり、新しいアトムを挿入したりすることができる。コマンドの処理が終ると、再びパーザーはエラーが発生した場所へ戻って処理を開始する。
データエディターにおける出力文の出力はファイルへの出力に相当するが、どの時点で出力するかが問題になる。すべてのデータの入力処理が終了してからファイルへ出力するのが簡単であるが、一般に一回に入力するデータの量は非常に多く、それらを一時格納しておくファイル出力用の多量の記憶領域が必要となる。これは現実的ではない。構文解析の中間段階で出力を行うことを考えるべきである。また、言語変換仕様の意味部で SETR という操作を用いることによりグローバルなレジスターに中間段階での出力情報を格納することを考える。意味部の評価は、構文部の左辺に現われる metasymbol の値を計算するものであった。構文単位の処理と意味部の評価の機構を拡張する。構文部に // が現われたらそこで意味部の評価を行い、あわせて、意味部の頭に // が現われたら意味部の評価は行わない。 PD: metasymbol ⇒ pattern1 // pattern2 | // expression ;

↑ パーザがここにくると意味部の評価を行う。

↑ 後に、ここにくると意味部の評価は行わない。

metasymbolの値は True となる。

このような機構によって、構文解析の中間段階での出力情報の収集が可能となる。
データエディターの言語変換仕様定義の簡単な例を次に示す。

(例) 入力データは日付とファイル名にレコードの並びが続き、レコードは色、数量、場所という三つの項目をもつものとする。

入力データ: 0322 STOCK RED 30 PARIS YELLOW 26 LONDON;

出力ファイル: ファイル名 'STOCK'

<色>	<数>	<場所>	<日付>
R	30	PRS	0322
Y	26	LND	0322

仕様定義:

```
PD: <TG> => <DATE> // <FILEOP> ! // SETR DATE <DATE> ;
PD: <FILEOP> => <FILENAME> // <REC* ! // SETR FILE <FILENAME> ;
PD: <REC* => <RECORD> <REC* ! T ;
PD: <REC* => <RECORD> ! T ;
PD: <RECORD> => <COLOR> <NUMBER> <PLACE> ! OUT FILE <COLOR> <NUMBER>
 <PLACE> DATE ;
MS: <COLOR> => ((RED.R)(BLUE.B)(YELLOW.Y)) ;
MP: <NUMBER> => NUMBERP ;
MS: <PLACE> => ((TOKYO.T)(LONDON.L)(PARIS.P)) ;
MP: <DATE> => NUMBERP ;
MP: <FILENAME> => ATOM ;
```

4. 言語インターフェースの構成例

①数式→ポーランド記法

```
*IN
#       (NOSTFLG:)
*IN
#       (BTFLG:)
*IN
#       (NOEFLG:)
*IN
#       (PD: <TG> => <ASS> ! <ASS>)
*IN
#       (PD: <ASS> => <VAR> := <EXP> ; ! FLAT := <VAR> <EXP>)
*IN
#       (PD: <EXP> => <TERM> + <EXP> ! FLAT + <TERM> <EXP>)
*IN
#       (PD: <TERM> => <FCT> * <TERM> ! FLAT * <FCT> <TERM>)
*IN
#       (PD: <FCT> => %( <EXP> %) ! <EXP>)
*IN
#       (PD: <FCT> => <VAR> ! <VAR>)
*IN
#       (MP: <VAR> => ATOM)
*IN
#       (PD: <EXP> => <TERM> - <TERM2> + <EXP> ! FLAT + - <TERM> <TERM2> <EXP>)
*IN
#       (PD: <EXP> => <TERM> - <TERM2> - <EXP> ! FLAT - - <TERM> <TERM2> <EXP>)
*IN
#       (PD: <EXP> => <TERM> - <TERM2> ! FLAT - <TERM> <TERM2>)
*IN
#       (PD: <TERM> => <FCT> / <FCT2> * <TERM> ! FLAT * / <FCT> <FCT2> <TERM>)
*IN
#       (PD: <TERM> => <FCT> / <FCT2> / <TERM> ! FLAT / / <FCT> <FCT2> <TERM>)
*IN
#       (PD: <TERM> => <FCT> / <FCT2> ! FLAT / <FCT> <FCT2>)
*IN
#       (PD: <FCT2> => <FCT> ! <FCT>)
*IN
#       (PD: <TERM2> => <TERM> ! <TERM>)
*IN
#       (PD: <EXP> => <TERM> ! <TERM>)
*IN
#       (PD: <TERM> => <FCT> ! <FCT>)
*IN
#       (SCGEN:)
   8DELIMS DEFINED ###
*IN
#       (YZ := AB + CD - PQ ;)
*OUT
#       (:= YZ + AB - CD PQ)
*IN
#       (YZ := %( AB + CD %) * PQ - DE / FG ;)
*OUT
#       (:= YZ - * + AB CD PQ / DE FG)
*IN
#       (A := B / C + D * E ;)
*OUT
#       (:= A + / B C * D E)
*IN
#       (A := %( B / C + D %) / E ;)
*OUT
#       (:= A / + / B C D E)
*IN
#       (A := B - %( C - D / %( E * F + G * H %) %) / I ;)
*OUT
#       (:= A - B / - C / D + * E F * G H I)
*IN
#       (BYE)
 * MAX OF USED CEL =         875
#PATTERN DEFINITION SETS :
#       ((<TERM2> <TERM> ! <TERM>) (<FCT2> <FCT> ! <FCT>) (<FCT> (<VAR> ! <VAR>
) %( <EXP> %) ! <EXP>) (<TERM> <FCT> ((!
 <FCT>) / <FCT2> ((! FLAT / <FCT> <FCT2>) / <TERM> ! FLAT / / <FCT> <FCT2> <TERM
>) * <TERM> ! FLAT * / <FCT> <FCT2> <TER
M>) * <TERM> ! FLAT * <FCT> <TERM>) (<EXP> <TERM> ((! <TERM>) - <TERM2> ((! FLAT
 - <TERM> <TERM2>) - <EXP> ! FLAT - - <T
ERM> <TERM2> <EXP>) + <EXP> ! FLAT + - <TERM> <TERM2> <EXP>) + <EXP> ! FLAT + <T
ERM> <EXP>) (<ASS> <VAR> := <EXP> ; ! FL
AT := <VAR> <EXP>) (<TG> <ASS> ! <ASS>))
#   SET DEFINITIONS OF MS
#       NIL
 # PREDICATE DEFINITIONS OF MP:
#       ((<VAR> ATOM))
  # LIFT HALT ##
```

フラグ定義

言語変換仕様定義

区切り子定義

入力出力

入力出力

入力出力

入力出力

入力出力

BYE コマンド

終了時情報出力

各定義、入力文は端末から直接ではなく、データセットから LIFT に入力している。本文中の記述と書式が若干異なるところがある。

② 英語によるデータベースの検索
英語 → データベース検索・中間言語

言語インターフェース仕様定義

```
PD: <TG> => GET <TARGET> FOR <COND> %. ! FLAT SELECT <TARGET> <COND> ;
PD: <TG> => GET <TARGET> OF <COND> %. ! FLAT SELECT <TARGET> <COND> ;
PD: <TG> => GET <TARGET> FOR <COND> , <OCOND> %. !
    FLAT SELECT <TARGET> <COND> <OCOND> ;
PD: <TARGET> => <FIELDNAM> AND <TARGET> ! FLAT <FIELDNAM> ,
    <TARGET> ;
PD: <TARGET> => <FIELDNAM> ! LIST <FIELDNAM> ;
MS: <FIELDNAM> =>((FULL-DETAILS.*) (PART-NUMBERS.P#)) ;
MS: <FIELDNAM> =>((SUPPLIER-NUMBERS.S#) STATUS) ;
PD: <COND> => ALL PARTS SUPPLIED ! LIST FROM SP ;
PD: <COND> => ALL SUPPLIERS ! LIST FROM S ;
PD: <COND> => SUPPLIERS <RESTR> ! FLAT FROM S WHERE <RESTR> ;
PD: <OCOND> => IN <ORDER> ORDER OF <FIELDNAM> !
    FLAT ORDER BY <FIELDNAM> <ORDER> ;
PD: <RESTR> => <RESTRELM> <RESTR> ! FLAT <RESTRELM>
    AND <RESTR> ;
PD: <RESTR> => <RESTRELM> ! <RESTRELM> ;
PD: <RESTRELM> => IN <LOC> ! LIST CITY = <LOC> ;
PD: <RESTRELM> => WITH <FIELDNAM> <OP> <VALUE> !
    LIST <FIELDNAM> <OP> <VALUE> ;
MS: <LOC> => (PARIS) ;
MS: <OP> => ( < > <= >= <> = ) ;
MS: <ORDER> => ((DESCENDING.DESC)(ASCENDING.ASC)) ;
MP: <VALUE> => ATOM ;
SCGEN: ;
EFLG: ; NOSTFLG: ;
```

入力－出力

```
*IN
#       (GET SUPPLIER-NUMBERS AND STATUS FOR SUPPLIERS IN PARIS , IN DESCENDING
 ORDER OF STATUS %.)
*OUT
#       (SELECT S# , STATUS FROM S WHERE CITY = PARIS ORDER BY STATUS DESC)
*IN
#       (GET PART-NUMBERS FOR ALL PARTS SUPPLIED %.)
*OUT
#       (SELECT P# FROM SP)
*IN
#       (GET FULL-DETAILS OF ALL SUPPLIERS %.)
*OUT
#       (SELECT * FROM S)
*IN
#       (GET SUPPLIER-NUMBERS FOR SUPPLIERS IN PARIS WITH STATUS > 20 %.)
*OUT
#       (SELECT S# FROM S WHERE CITY = PARIS AND STATUS > 20)
*IN
#       (BYE)
```

③ 日本語によるデータベースの検索
日本語 → データベース検索中間言語

言語インターフェース仕様定義

```
EFLG: ; NOMSFLG: ; NOSTFLG: ;
PD: <TG> => <SHIP> NO <ATTR> WA ? ! LIST <SHIP> <ATTR> ;
PD: <TG> => <SHIP> NO KAZU WA ? ! LIST <SHIP> (?(* NUM)) ;
PD: <SHIP> => <SHIPNAM> ! LIST NAM EQ <SHIPNAM> ;
PD: <SHIP> => <ATTRNAM> 'GA ' <VALUE> NO FUNE ! LIST <ATTRNAM> EQ
  <VALUE> ;
PD: <ATTR> => <ATTRNAM> <ATTR> ! CONS <ATTRNAM> <ATTR> ;
PD: <ATTR> => <ATTRNAM> ! LIST <ATTRNAM> ;
PD: <VALUE> => <NUMBER> <UNIT> ! LIST <NUMBER> <UNIT> ;
PD: <VALUE> => <NAME> ! <NAME> ;
MS: <UNIT> => ( T FEET M KG) ;
MS: <ATTRNAM> => ((NAMAE.NAM) (NAGASA.LEN) (OMOSA.WT)) ;
MS: <SHIPNAM> => ((PINCHI-GO.PIM) (POPPOKO-GO.POM) (PANKU-GO.PAN));
MP: <NAME> => ATOM ;
MP: <NUMBER> => NUMBERP ;
SCGEN: ;
```

入力－出力

```
*IN
#       (NAMAE GA  SYSTEM NO FUNE NO NAGASA WA ?)
*OUT
#       ((NAM EQ SYSTEM) (LEN))
*IN
#       (OMOSA GA  30 T NO FUNE NO KAZU WA ?)
*OUT
#       ((WT EQ (30 T)) (? (* NUM)))
*IN
#       (PINCHI-GO WA ?)
--&ELPS-START OF ELLIPSIS--
*EP
#       (PINCHI-GO NO KAZU WA ?)
*OUT
#       ((NAM EQ PIM) (? (* NUM)))
*IN
#       (BYE)
```

省略処理

用いられている英語、日本語ともに非常に断片的なものである。

5. 参考文献

1) G.G. Hendrix: The LIFER manual: A Guide to Building Practical Natural Language Interfaces, SRI Int. TN138 (1977)
2) G.G. Hendrix, E.D. Sacerdotti, D. Sagalowicz, J. Slocum: Developing a Natural Language Interface to Complex Data, ACM TODS, 3, 2 (1978)
3) R.R. Burton: Semantic Grammer: An Engineering Technique for Constructing Natural Language Understanding System, BBN Rep. 3453 (1976)