自然言語インターフェイス

人間が他者とコミュニケーションを行うのに最も効率の良い手段は、日本語や英語などの自然言語である。しかし、計算機とのやりとりには、依然使いづらい人工言語を使わなければならない。本実験では、自然言語の機械的な処理の一例として、簡単なデータベースシステムへの自然言語での問い合わせを可能にするためのフロントエンドを作成することにより、自然言語処理の基本的な技術を習得する。

第1日目	自然言語処理概要
第2日目	構文解析
第3日目	意味解析
第4日目	対話処理

目 次

作成者：佐川 雄二 (1995/2/10)　　改定： 河口 信夫 (1996/8/9)

自然言語インターフェイス

1　目的

簡単なデータベースシステムとの自然言語インターフェイスの設計を通して、自然言語処理の基本的な問題とその解決手法を習得する。

2　解説

2.1　自然言語処理とは

人間は他者とコミュニケーションするのに多くの手段を用いるが、その中でもっとも良く使われているのが、日本語や英語などの自然言語である。他方、計算機とのやりとりには、人間が設計した人工言語 (プログラミング言語) を依然用いなければならない。人工言語は曖昧さがないなど、機械的に処理するには都合のいい言語であるが、計算機を使うためにいちいちそれを覚えなければならないのは、ユーザにとって苦痛である。使いなれた自然言語で計算機と対話できれば、ユーザの負担は激減する。

自然言語処理とは、自然言語の機械的な処理を行なう技術である。自然言語処理の応用としては機械翻訳と自然言語インターフェイスが代表的である。

自然言語処理の問題には、大きく分けると自然言語理解および自然言語生成がある。そのなかでも自然言語理解は、自然言語を計算機に理解させるためにはどうすれば良いかの問題を扱う。言語を理解する能力は、人間の知的活動の中でも大きな位置を占めるため、自然言語理解は人工知能の中でも重要な問題となっている。

2.2　自然言語インターフェイス

自然言語インターフェイスは、エキスパートシステムやデータベースシステムに対して自然言語で問い合わせができるようにするためのツールである。したがってその仕事は大ざっぱに言えば、入力された自然言語の文字列をエキスパート／データベースシステムの問い合わせ言語に変換することである。

自然言語インターフェイスの処理は、一般的に、形態素解析、構文解析、意味解析、文脈解析の順で行なわれる。形態素解析は文字列を単語へと分解し、その単語を辞書を用いて特定する処理である。構文解析（統語解析）は文法を用いて係り受けなどの文の構造を特定する処理である。意味解析は知識を用いて文の意味を特定し、問い合わせ言語として出力する。文脈解析はその文

単独では決定できない、前後の文脈を参照する必要のある処理（たとえば代名詞が何を指すか）を行い、文の最終的な意味を決定する。

これらの処理は必ずしも明確に分けることはできない。少なくとも人間の場合は、各処理が相互にインタラクトしながら文の理解を行っていると思われる。例えば一部の単語を間違っていても、意味を理解できることがある。これは必ずしも形態素／構文解析できなくても意味解析のできる例であり、意味解析主導での処理が人間には可能であることを示している。しかし多くの場合、上のような流れで文の解析をモデル化できるため、現状ではこのような処理を行っているシステムが大半である。

2.3 形態素解析

形態素解析 (morphological analysis) では、入力された文字列を単語に分割し、それぞれの単語の品詞や意味を辞書を用いて調べる。英語のように単語が空白で分割 (分かち書き) されている言語では単語への分割は容易であるが、日本語のように分かち書きの習慣のない言語では、単語への分割自体も難しい問題である。例えば例 2.1 を考えると、

【例 2.1 】

今日は出てない

「今、日は出てない」なのか「今日は、出てない」なのかあいまいである。

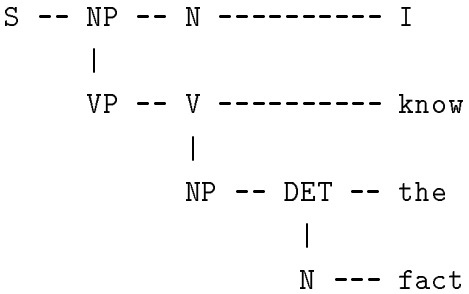
また、動詞のように活用のある単語の場合、そのすべてを辞書に登録しておくことは無駄であるので、活用が規則的な場合は語幹と語尾を別々に登録しておいて、「考え・られ・な・い」のように分割する。このような単語よりも小さい構成単位を形態素 (morpheme) と呼ぶ。したがって形態素解析とは正確にいえば、文を形態素に分析する処理である。

2.4 構文解析

構文解析 (syntactic analysis または parsing) では文法を用いて文の構造を解析する。例えば、例 2.2 の文は次のような構造をしている。

【例 2.2 】

I know the fact.



すなわち、文 (sentence, S) は名詞句 (noun phrase, NP) と動詞句 (verb phrase, VP) からなり、動詞句は動詞 (verb, V) と名詞句からなり、名詞句は名詞 (noun, N) 単独からなる場合と冠詞 (determiner, DET) と名詞からなる場合がある、という知識を用いれば、あとは各単語の品詞がわかればよいわけである。そうした知識を文法とよび、例えば次のように記述する。また S や NP などを文法カテゴリとよぶ。

```
S -> NP VP
VP -> V NP
NP -> DET N
NP -> N
```

2.4.1 文脈自由型句構造文法

先の文法のように各文法規則の左辺が 1 個のカテゴリだけからなる文法を、文脈自由型句構造文法 (context-free phrase structure grammar) と呼び、その文法で記述できる言語を文脈自由言語 (context-free language) と呼ぶ。言語にはこの他にも、有限状態言語 (finite-state language)、文脈依存言語 (context-sensitive language)、再帰可算集合 (Recursively enumerable set) があり、後のものほど、記述できる言語の範囲が広くなっている。文脈自由言語は有限状態言語と文脈依存言語の間に位置する。

例えば例 2.3 の文は

【例 2.3 】

That that that he is young is false is false is false.

文脈自由文法では解析可能であるが、有限状態文法では解析できない。

自然言語の文の大部分は文脈自由言語の範囲に入ると考えられている[1]

2.4.2 下位範疇化と素性

2.4 の文法では、例えば、次のような文を正しく解析することはできるが、

【例 2.4 】

I have a pen.
I have an apple.

次のような文を文法的に正しくないと判定することはできない。

【例 2.5 】

I have an pen.
I have a apple.

[1] チューリッヒ地方で話されているドイツ語の方言の中に文脈自由言語の範囲を越えるものが発見されている。

これを修正する一つの方法は、DET および N をそれぞれ 2 つのグループに分け、それぞれについての規則を設けることである。すなわち a を DET1、an を DET2 とし、pen を N1、apple を N2 としたうえで、文法を次のようにすればよい。

```
S -> NP VP
VP -> V NP
NP-> DET1 N1
NP-> DET2 N2
NP -> N1
NP -> N2
```

このように文法カテゴリを細分化して文法を詳細にしていくことを下位範疇化 (subcategorization) と呼ぶ。下位範疇化は有効な方法であるが、あまりに細分化するとどのカテゴリがどういう目的でわけられたのかがわからなくなってくる。その原因は下位範疇化した場合、表に残るのがカテゴリ名だけである点にある。

下位範疇化と同様な効果を、そのような欠点をなくして実現するのが、素性 (feature)[2] の導入である。素性とは各カテゴリの属性であり、カテゴリは素性の集合として定義されることとなる。

例えば先の例については、initial という素性を導入し、a, pen にはその値として vowel、an, apple には consonant を与えることを辞書に記述しておく。そのうえで規則 NP-¿DET N にこの規則の適用条件として (DET initial)[3] =(N initial) を与えておけば、例 2.5 をはじくことができる。

このようにして素性の導入により主語の数と動詞の数の一致や自動詞/他動詞と目的語の有無の整合性などをチェックすることができる。このような文のある部分とある部分がある種の関係を持っていることを共起関係という。

2.4.3 あいまいさ

自然言語処理を難しくしているものの多くは自然言語が持つ「あいまいさ」に起因している。たとえば単語レベルでは同音意義語というあいまいさがある。また構文のレベルでは次の例のような係り受けのあいまいさがある。

【例 2.6 】

I saw a man with a telescope.

この例では、”with a telescope”が”man”に係るのか、”saw”に係るのかがあいまいである。

このようなあいまいさは、次に述べるような意味解析、文脈解析の過程でどちらかにしぼれるものも多い。しかし常識に基づく深い推論や聞き手の好みまで考えなければ解決できないものも多い。

[2] 「そせい」と読む

[3] DET の initial という素性の値を表わす

2.5 意味解析

意味解析 (semantic analysis) で問題になるのは最終的な出力をどのような形式にするかである。つまりどのような知識表現法 (knowledge representation) を用いれば人間の持っている知識、行う推論 (reasoning/inference) をすべてカバーできるかは未だ最終的な答えは出ていない。これは「理解とは何か」という非常に難しい問題と密接に関連する大問題である。

しかしエキスパートシステムもしくはデータベースシステムとのインターフェイスという意味で捕えるならば、自然言語インターフェイスが出力するのはそれらの問い合わせに用いられる表現でよい訳である。

本実験で意味表現として用いるのはエキスパートシステムでも広く用いられている一階述語論理式 (first order predicate logic equation) である。それは式としては述語名と引数からなる。例えば「太郎が花子に花をあげる」という文の意味は (give taro hanako flower) と表現される。ここで give が述語名であり、taro, hanako, flower が引数である。述語名を何にするか、引数の並びをどうするかは、特に決まりはない。システム全体で整合性がとれていればよい問題である。

では、構文解析結果からどのようにして意味表現が得られるのであろうか。ここで重要となるのが Frege の構成性原理 (principle of compositionality) である。それは「全体の意味はその部分部分の意味から構成される」というものである。つまり A という句が B という句と C という句からなる場合、A の意味は B の意味と C の意味の組み合わせで決まるということである。

例えば

【例 2.7 】

Mary uses Tom's pencil.

ではまず"Tom's pencil"の意味として「Tom が鉛筆を所有していること」(own Tom penci l) 、つづいて"uses Tom's pencil"の意味として「その (Tom が所有している) 鉛筆を使うこと」(use ?someone pencil)[4] が構成され、最終的に文全体の意味として (use Mary pencil), (own Tom pencil) が構成される。このように１つの文の意味は複数の述語を含んでいることも多い。

いずれにしても重要なのは、意味を構成する手続きが文法規則と対応しているので、文法規則に付随してその手続きを記述することができるという点である。

2.5.1 意味的整合性による構文的あいまいさの解消

次の例

【例 2.8 】

Get a screwdriver with your left hand.

では、"with your left hand"が"screwdriver"に係るのか、"get"に係るのかあいまいである（少なくとも構文的には)。しかし意味的には１つの解しか存在しない。これは、人工物についているものは人工物でなければならないという条件を用いれば前者の解釈を無効とすることで実現できる。

[4] まだこの時点では、誰が鉛筆を使うかはわかっていない。

2.6 文脈解析

自然言語処理の難しさの一つに、文の意味がその文単独で決まらない場合が多い、という点がある。すなわち文の意味は前後の文脈 (context) とあわせてはじめて決定する。その最もわかりやすい例が照応 (anaphora) 現象である。

照応現象とは、いわゆる代名詞化（＋代動詞化）および省略 (ellipsis、ゼロ照応）のことである。単純な場合は最も最近登場した名詞で数や性の一致するもの（they が Mary を指すことはない）で置き換えればよいが、例外も多く、完全な解決は実現していない。

また、人間は必ずしも言いたいことを直接言葉にするわけではない。「この部屋、暑いね」という発話は、部屋が暑いという事実を伝えたいというよりはむしろ窓を開けるなり、エアコンをかけるなりして涼しくしてほしい、という依頼として解釈した方が自然である。こうした発話を行うことを間接発話行為 (illocuitionary act) とよび、これを正しく認識するには相手がその発話の背後に持っているゴールとそれを実現するためのプランに関する推論が必要になる。

3 実験

3.1 実験に用いるツール

今回の実験には、参考文献 [1] のツールを今回の実験用に若干変更したプログラムを用いる。

基本的には、文脈自由文法に基づいた構文解析、素性を用いた構文的・意味的共起関係による曖昧さの解消、意味表現としての述語論理式の抽出とそれを用いたデータベースへの事実の追加と問い合わせを行なうことができる。それぞれ具体的な例をあげて説明する。

3.2 文法規則および辞書の記述

文法規則 (素性を用いた簡単な文脈自由文法) は以下のように記述する。

```
(setq rules
   '((Rule (S   -> NP VP)
           (S cat) = S
           (NP cat) = NP
           (VP cat) = VP)

     (Rule (VP  -> V NP)
           (VP cat) = VP
           (V cat) = V
           (NP cat) = NP)

     (Rule (NP -> DET N)
           (NP cat) = NP
           (DET cat) = DET
           (N cat) = N
```

```
            (DET initial) = (N initial))

      (Rule (NP -> N)
            (NP cat) = NP
            (N cat) = N)))

(setq lexical_rules
   '((Word (this)
            (cat) = N)
      (Word (is)
            (cat) = V)
      (Word (a)
            (cat) = DET
            (initial) = con)
      (Word (an)
            (cat) = DET
            (initial) = vow)
      (Word (pen)
            (cat) = N
            (initial) = con)
      (Word (eraser)
            (cat) = N
            (initial) = vow)))
```

例からわかるように、X というカテゴリの f という素性は (X f) と記述する。各規則には、素性の値のチェックを与えることができる。規則は、与えられた式すべてが成立する時にのみ、適用可能である。この例では、initial という素性をもちいて前章で述べたような共起関係のチェックを行なっている。また、後述するようにこれらの式は、チェックの他に上位のカテゴリに情報を伝達する目的でも使われる。

3.3 意味解析

構文解析の後、意味解析を行なって文の意味を表す述語を抽出するには、次の例のようにやはり素性を用いる。

```
(setq rules
     '((Rule (S   -> N VP)
             (S cat) = S
             (N cat) = N
             (VP cat) = VP
             (S sem) = (VP sem)
             (S sem arg0) = (N sem))
```

```
        (Rule (VP  -> V NP)
              (VP cat) = VP
              (V cat) = V
              (NP cat) = NP
              (VP sem predicate) = (V sem)
              (VP sem arg1) = (NP sem))

        (Rule (NP -> DET N)
              (NP cat) = NP
              (DET cat) = DET
              (N cat) = N
              (NP sem) = (N sem))))

(setq lexical_rules
   '((Word (i)
           (cat) = N
           (sem) = i)
     (Word (love)
           (cat) = V
           (sem) = love)
     (Word (the)
           (cat) = DET)
     (Word (man)
           (cat) = N
           (sem) = man)))
```

この例のように、最終的にカテゴリ S の sem という素性に意味の情報が集められるようになっている。この素性はさらにサブ素性を持っており[5]、predicate が述語名に対応する。その他の arg0、arg1 は引数に対応し、arg3 までの 4 引数が使用できる。これにより (predicate arg0 arg1 arg2 arg3) という述語が意味として抽出される。なお、arg3 までのすべての引数を使う必要はない。

また、引数が述語に対してどういう関係であるかはユーザが決めることであり、その管理もユーザに任されている。さらに本ツールでは、先の例のように 1 つの文から複数の述語を得ることはできない。

3.4 意味を用いた構文的曖昧さの解消

意味解析の途中で、構文的には正しいけれども意味的におかしな文をはじくには、共起関係と同様に素性の値のチェック式を用いる。

```
(setq rules
```

[5] C 言語風に書くなら、(VP sem arg1) は VP.sem.arg1 のようなものだと思えばよい。

```
   '((Rule (S -> N VP)
           (S cat) = S
           (N cat) = N
           (VP cat) = VP
           (S sem) = (VP sem)
           (S type) = (VP type)
           (S sem arg0 ) = (N sem)
           (S type arg0) = (N type)    )

     (Rule (VP -> V NP)
           (VP cat) = VP
           (V cat) = V
           (NP cat) = NP
           (VP sem predicate) = (V sem)
           (VP type) = (V type)
           (VP sem arg1) = (NP sem)
           (VP type arg1) = (NP type)  )

     (Rule (NP -> DET N)
           (NP cat) = NP
           (DET cat) = DET
           (N cat) = N
           (NP sem) = (N sem)
           (NP type) = (N type)     )))

(setq lexical_rules
    '((Word (i)
            (cat) = N
            (sem) = i
            (type) = animal     )
      (Word (eat)
            (cat) = V
            (sem) = eat
            (type arg0) = animal
            (type arg1) = food  )
      (Word (repair)
            (cat) = V
            (sem) = repair
            (type arg0) = animal
            (type arg1) = machine  )
      (Word (the)
```

```
        (cat) = DET )
    (Word (apple)
        (cat) = N
        (sem) = apple
        (type) = food     )
    (Word (computer)
        (cat) = N
        (sem) = computer
        (type) = machine  )))
```

この例では動詞"eat" の動作主は動物 (animal)、動作対象は食べ物 (food) でなければならないという条件を素性 type を使ってチェックしている。この規則により、"I eat the apple." は適切に意味が抽出されるが、"I eat the computer." はこのチェックではねられ、構文解析はできるが、意味解析には失敗する。

3.5 システムとの対話

システムと自然言語で対話するには、文の意味だけでなく、その文がどういう目的で用いられるかを明らかにしなければならない。例えば、平叙文は文の内容を聞き手に情報として伝達するために用いられる。疑問文は聞き手にその質問に対する答を返答するよう要求するために用いられる。

本ツールでは、平叙文、yes-no 疑問文、wh 疑問文の 3 種類の文を扱うことができる。それぞれ (S sem mode) に declarative, yes-no-q, wh-q という値をセットすることによって指定できる。これらは各文形に対応した文法規則の中で行なう。また、wh 疑問文の際はさらに何について尋ねているかを明らかにしなければならない。これは対応する引数に?をセットすれば良い。つまり"What does Mayumi eat?" の場合は、(eat mayumi ?) を意味解析の結果得られるように規則を書く必要がある。

```
(setq rules
   '((Rule (S -> N VP)
           (S cat) = S
           (N cat) = N
           (VP cat) = VP
           (S sem) = (VP sem)
           (S sem arg0) = (N sem)
           (S sem mode) = declarative )

     (Rule (S -> WP N V)
           (S cat) = S
           (N cat) = N
           (WP cat) = WP
           (V cat) = V
```

```
            (S sem) = (VP sem)
            (S sem arg0) = (N sem)
            (S sem arg1) = (WP sem)
            (S sem mode) = wh-q
            (S sem predicate) = (V sem) )

      (Rule (VP -> V NP)
            (VP cat) = VP
            (V cat) = V
            (NP cat) = NP
            (VP sem predicate) = (V sem)
            (VP sem arg1) = (NP sem) )

      (Rule (NP -> DET N)
            (NP cat) = NP
            (DET cat) = DET
            (N cat) = N
            (NP sem) = (N sem)   )))
(setq lexical_rules
    '((Word (Mayumi)
            (cat) = N
            (sem) = mayumi )
      (Word (eats)
            (cat) = V
            (sem) = eat    )
      (Word (eat)
            (cat) = V
            (sem) = eat    )
      (Word (the)
            (cat) = DET )
      (Word (apple)
            (cat) = N
            (sem) = apple  )
      (Word (what does)
            (cat) = WP
            (sem) = ?      ) )))
```

3.6 ツールの使用法

本ツールは、Common Lisp で記述されている。Lisp を立ちあげてシステムのファイルを読み込んだ後、talk 関数を使ってシステムに文を入力する。システムファイルは dialog.lisp, lisppart.lisp, dagunify.lisp, subst.lisp からなる。また、lisp-init.lisp をホームディレクトリに置いておくと、Lisp は起動時にこれを読み込む。

```
% lisp                          # Lisp の起動
<起動 メッセージ>
> (load "dialog.lisp")          # システムの読み込み
> (load <辞書ファイル名>)       # 辞書の読み込み
> (talk '(i have a pen))        # 対話文の入力
```

システムは構文解析に失敗すれば I cannot parse your utterance、成功したが解が複数ある場合には There are xx ambiguities、解が一意に決定し場合は There is no ambiguity と表示する。

さらに意味解析に成功すると、述語が表示される。その上で mode の値が正しければ、前節で述べたような答を返す。すなわち、平叙文の場合にはその述語をシステムの知識に加え、yes-no 疑問文の場合は同じ述語がシステムの知識に存在すれば yes、そうでなければ no を返す。wh 疑問文の場合にはマッチする述語があれば対応する答を返す。

システムは立ち上がった状態では、知識は空の状態である。いくつかの平叙文で知識を与えた後、質問をしてみると良い。

また、文法規則がどのように解析されているのかを調べるには、デバッグコマンド(debugon)を実行すると、以後、解析経過が表示される。表示を止めるコマンドは(debugoff)である。実行を高速に行うためには(compile 'talk)によりコンパイルを行うと良い。

システムを終了するには(quit)を入力する。

4 実験項目

1. 2.4 の文法規則および辞書を入力し、文を与え構文解析せよ。
2. 文法を文脈自由文法の枠内で（素性を導入しないで)、より広い英文が構文解析できるよう拡張せよ。どの程度拡張するかは各自の判断に任せる。
3. 素性を用いた規則適用制限機能を使って一文内のさまざまな共起関係を扱えるよう文法および辞書を拡張せよ。
4. 意味を用いた構文的曖昧さの解消を行なう文法を作れ。
5. 構文解析後、意味記述が得られるように文法および辞書を拡張せよ。文は、平叙文、yes/no 疑問文、wh 疑問文が扱えるようにし、それぞれ解説で述べたような意味記述が得られるように記述せよ。能動態と受動態のように構文が違っていても、同じ意味のものをきちんと扱えるようにせよ。
6. 作成した自然言語インターフェイスを用いてシステムとの対話を行なえ。インターフェイスの題材は各自工夫すること。

5 考察

1. 今回作成したインターフェイスで扱えない文の種類をできるだけ多く挙げよ。
2. 今回の実験は英文を対象に行なったが、対象を日本語文とした場合、どのような点が異なるか。英語より簡単な点、英語より難しい点について考察せよ。
3. 今回は文を理解する問題のみを扱った。自然言語処理のもう一つの問題として、どのように文を生成する (組み立てる) かの問題がある。生成の問題が理解の問題と異なる点、類似する点を挙げよ。
4. その他、自然言語の機械的処理を難しくしている問題を挙げ、考察せよ。
5. 感想

参考文献

[1] Gerald Gazdar and Chris Mellish: *Natural Language Processing in Lisp*, Addison Wesley (1989).