連載講座

自然言語処理入門—III

文を解析してみよう†

岡田直之††　中村順一††

前回述べた言語理論に沿って，文を解析する手法を示そう．解析のステップとしては，大きく形態素，構文，意味および文脈の4つがあるが，今回は，構文，意味および文脈解析に取り組もう．

4. 言語解析(2)

紙面の都合で，形態素解析の部分を前回の **3.** で述べた．そのため多少前後するが，**4.1** で言語処理とプログラミング技法について，簡単に歴史を振り返っておこう．

次に，構文構造をとらえるために提案された句構造文法 (Phrase Structure Grammar) を用いた構文解析の手法を紹介する．PSG は，数理的性質がよく調べられており，解析に必要な時間およびメモリ量の観点から効率的な手法がいくつも提案されているが，**4.2** では，**トップダウン／ボトムアップ** (top-down/bottom-up) という最も基本的な考え方を説明しよう．また，この考え方を通じて，計算機内部でのプログラムの振舞いを少し詳しく述べよう．そして，具体的な解析手段として，**確定節文法** (Definite Clause Grammar) の概念を導入し，これに基づいて Prolog と呼ばれるプログラミング言語を用いて簡単な解析システムを実現してみよう．また，その他の解析手法についても二，三触れよう．

最後に，意味および文脈解析について述べる．**4.3** では，始めに意味解析の役割に触れ，DCG を用いた簡単な意味解析の例を示そう．次に，**2.2** で述べた意味論，特に意味素性や格フレームを実際に取り扱う手法として，**単一化文法** (Unification Grammar) を紹介する．最後に **4.4** で文脈解析について述べるが，文脈に関する理論・解析手法は今のところ十分成熟するに至っていない．そこで，文脈解析で扱う必要のある言語現象を中心に二，三の考え方を例示するにとどめる．

4.1 言語処理とプログラミング技法

自然言語処理 (Natural Language Processing) の歴史は，計算機の歴史とほぼ同じ 1940 年代の後半にスタートしたことを **1.2.1** で述べた．その当時は，使用可能なプログラミング言語は“アセンブラ”と呼ばれるハードウェアに密着した言語や，数値計算に便利な Fortran のみであった．そこで 1950 年代の NLP はこれらのプログラミング言語を用いて行われた．

当時はハードウェアの性能が現在と比較にならないくらい低かった．たとえば，比較的大きな辞書を記憶するのに主として磁気テープが用いられたので，1列に格納されたデータを効率よく検索する手法が大きな課題の一つであった．Fortran による機械翻訳システムの代表例としては SYSTRAN がある．

1960 年代に入ると，特に米国において，Lisp と呼ばれる記号処理用言語が人工知能の分野で盛んに用いられるようになった．Lisp は，M. I. T. の J. マッカーシーらが考案した言語で，記号の並べ換え，比較，検索などを簡単に処理することができる．**1.2.2** で触れた T. ウィノグラードの SHRDLU を始め，1970 から 1980 年代の NLP システムの多くで Lisp が用いられた．

1970 年代に入り，機械翻訳を研究していた A. コルメロアは，彼の考案したアルゴリズムを一般化すれば，記号論理における定理証明のアルゴリズムと同等になることに気が付いた．これをプログラミング言語として実現したのが Prolog である．Prolog は，**4.2** および **4.3** で示すように，PSG を基礎にした構文解析や記号論理を基礎とした意

† Outline of Natural Language: Natural Language Analysis by Naoyuki OKADA and Jun-ichi NAKAMURA (Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology).

†† 九州工業大学情報工学部知能情報工学教室

味処理との整合が非常によい．そのため急速に普及し，特に我が国では第5世代コンピュータ計画がその開発に力を注いだ．

今日，NLP システムを研究する上では，Prologの普及などにより，プログラミング上の障害はかなり克服されたといえよう．しかし，実用システムを構築する上では，やはり処理効率の観点などから，C言語など汎用プログラミング言語が多く用いられている．今後は，大規模な実用システムや新しいパラダイムの超並列・学習システムを目指すプログラミング言語や処理技法の開発が課題となろう．

4.2 構文解析

構文解析 (syntactic analysis) とは，与えられた文（単語列）に対して，与えられた文法規則を用いて文の構造を求める操作であり，その結果は**解析木**で表現される．

2.1.1 で示した PSG の具体例 EG 1 をもう1度復習しておこう．次の文

ES 1 : Taro drives a sports-car

に関する EG 1 の書き換え規則 ER 1 は，**図-1** に示すものであった．たとえば規則 (ⅰ) は，“文Sは，名詞句 NP と動詞句 VP からなる”といった，句の間の階層と順序の関係を示している．ER 1 に従うと ES 1 の解析木は，**図-2** になり，これは **2.** の**図-2** で示した導出木にほかならない．

4.2.1 基本的手法

では，具体的にどのようにすれば，解析木を求めることができるのであろうか．

ER 1 = {S→NP VP, (ⅰ)
NP→PrpN, (ⅱ)
VP→V NP, (ⅲ)
NP→Det N, (ⅳ)
PrpN→Taro, (ⅴ)
V→drives, (ⅵ)
Det→a, (ⅶ)
N→sports-car} (ⅷ)

図-1 句構造文法 ER 1 (再掲)

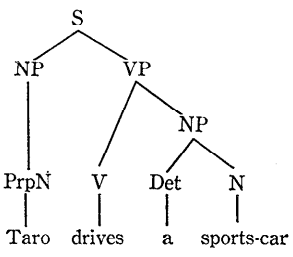


図-2 解析木

ボトムアップ解析　一つの方法は，初めて英語を学んだ人が英文を解釈するのと類似した方法である．上記 ES 1 の場合，まず，**図-3** の1のように辞書を引いてそれぞれの品詞を調べる．ER 1 の規則 (ⅴ)-(ⅷ) を右辺から左辺に逆にたどることに相当する．次に，図-3 の2のように，それぞれの品詞およびその並びから上位の構文カテゴリを見つけ，より大きな句を調べる．これは，規則 (ⅱ) と (ⅳ) を用いて主語と目的語にあたる名詞句 NP を求めることに相当する．その結果から，図-3 の3，続いて4のように，学校文法でいう“S+V+O”のような，文の骨組みとなる構造を考える．ER 1 でいえば，まず (ⅲ) を用いて動詞句 VP を求め，次いで (ⅰ) を適用して S を求めることに相当する．

このように，規則を右辺から左辺に逆に適用して，“下（ボトム）”から“上に向けて（アップ）”木を組み立てていき，首尾よくSにまとまったら解析成功という方法を**ボトムアップ解析** (bottom-up analysis) と呼ぶ．

トップダウン解析　ボトムアップ解析とはまった

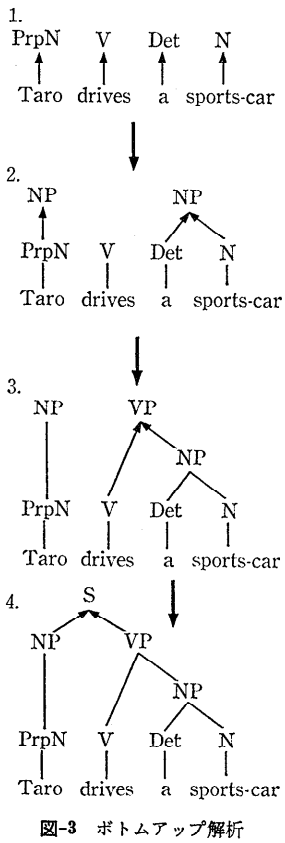


図-3 ボトムアップ解析

く逆向きの方法がある．これを理解するため，**図-4** を考えてみよう．

まず，初期記号Sを左辺にもつ規則(ⅰ)を適用する．これにより同図1の木が得られる．次に，一方の枝の末端記号 NP に注目し，NP を左辺にもつ規則(ⅱ)を適用して，同図2の木に成長させる．さらに，(ⅴ)を用いて同図3を得る．ここで木の末端の語が，入力単語列の先頭の語 Taro と首尾よく一致するので，ひとまずこの部分の展開を終了する．引き続き，まだ作り終えていない VP の展開に移る．規則(ⅲ)を用いれば，4の木が作れる．以下同様の操作を繰り返し，最終的に図-4 の8の解析木が得られる．

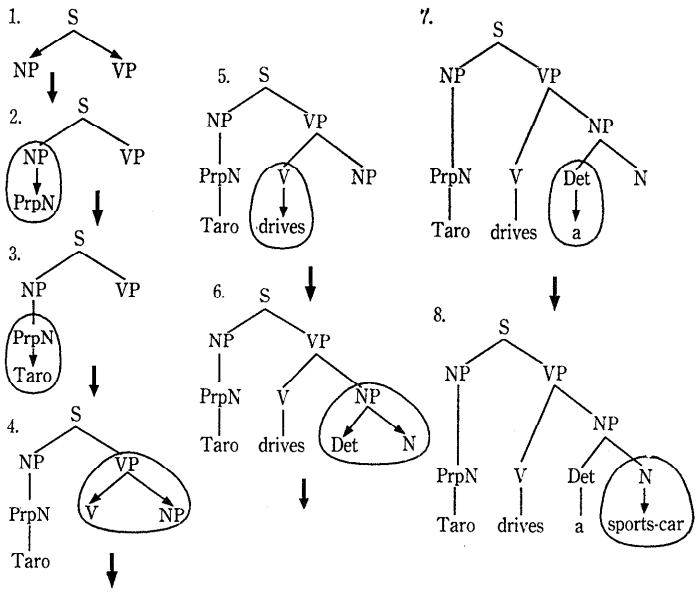


図-4 トップダウン解析

このように，書き換え規則をSから順方向に適用して“上（トップ）”から“下に向けて（ダウン）”木を組み立てていき，最終的に“葉”の並びが入力文と一致したなら解析成功という方法を，**トップダウン解析** (top-down analysis) と呼ぶ．

[問 1] ボトムアップにしろトップダウンにしろ木を組み立てる過程でいろいろな“曖昧性”に出会う．その一つに，複数の箇所に規則が適用できるとき，いずれの部分から処理すればよいか明確でない，ということがある．たとえば，図-3 の1では4カ所で規則の適用が可能であるが，どのような順序で2が得られたか，同図からは不明である．統一的な処理順序として，どのようなことが考えられるであろうか．

解答例：大きく2通りが考えられる．複数の適用箇所に対して，一つは同時並列的に処理する方法と，もう一つはたとえば左端から逐次直列的に処理するものである．

処理順序に関して，同時並列的処理をパラレル法，逐次直列的処理をシリアル法と呼ぶ．なおシリアル法では，図-4 のように1の局面で左端を処理すると2が得られ，その局面で左端を処理すると3が得られ，というように次々に得られる新しい局面で左端から処理する，という考え方が一般的である．

プログラムの振舞い　では次に，計算機内部でのプログラムの振舞いを解説しよう．初めに，“スタック”と呼ばれる，データを記憶するための道具立てを説明する．スタックでは，**図-5** に示すように，新しいデータがくるたびに棚の上にそれを“積み”上げていく．データが必要になれば，一番上から順に“降ろす”．先に入れたデータほど，後に出てくることになる．

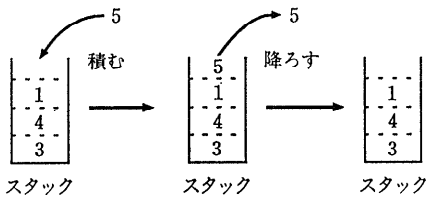


図-5 スタックの操作

それでは，**図-6** を用いて，トップダウン・シリアル法のプログラムの振舞いを述べよう．図-6の番号は，図-4 の番号と対応しているので注意してほしい．

0. まず，何も積んでいないスタックと入力文(単語列）から始める．

1. 最終的には単語列を初期記号 S にまとめたいので，Sを左辺とする規則を探して，スタックに積む．このとき，処理の進みぐあいを示す記号“＊”を，規則の右辺の左端に挿入しておく．

2. スタックの一番上の規則の，“＊”の右側の記号が展開すべき構文カテゴリである．この場合は NP であるので，NP を左辺にもつ規則をスタ

ックに積む．このときNPの左の“＊”をNPの右に進めるとともに，新しく積んだ規則の右辺の左端に“＊”を挿入する．

3. 同様に“＊”の処理を行いながら，PrpNを左辺にもつ規則をスタックに積む．

ここで，スタックの一番上の規則の右辺の記号が入力の単語列の先頭の記号，すなわちTaroと一致することが分かったので，図-4の4のように，VPの部分へ処理を進める必要がある．このため，図-6の3.1-3.2のように規則をスタックから降ろしていく．そうすると，まだ処理が終わっていない規則 S->NP＊VP が出てくる．右辺は，NPの処理は終わったがVPが未処理であることを示している．そこで，図-6の4のように，VPを左辺にもつ規則を再びスタックに積み，同様の処理を繰り返す．

以上の振舞いを整理すると，以下のようになる．

1. まず，左辺がSである“適切な”規則をスタックに積む．このとき，右辺の左端に処理の進みぐあいを示す＊を入れておく．

2. スタックが空にならないかぎり，以下の処理を繰り返す．

（a） スタックの1番上の規則を降ろす．

（b） その規則の＊の右に記号があれば，それ(“C”とする）を取り出し，以下のことを実行する．何もなければ何もしない．

● Cが単語である場合は，単語列の先頭と一致するかどうか調べる．一致すれば，その単語を入力から取りさる．一致しなければ，解析失敗で終了．

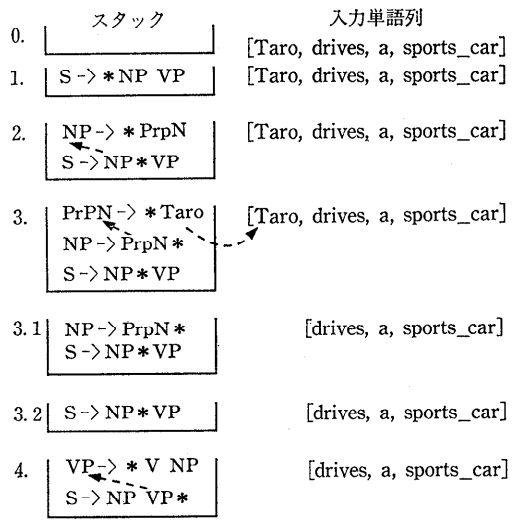


図-6　スタックを用いたトップダウン解析

● Cが構文カテゴリである場合は，

i．降ろした規則で，＊を一つ右へ移動させ，再びスタックに積む．

ii．左辺がCである“適切な”規則をスタックに積む．このとき，右辺の左端に＊を入れておく．

3. スタックが空で，かつ，単語列も空になっていれば，解析成功であるが，そうでなければ，失敗である．終了．

[問 2] 図-6の続きを図-4に対応して完成させよ．

解答例：図-7 のようになる．

さて，問1で処理順序の曖昧性について考えた．実は，上記アルゴリズムの中に規則の選択に関する，もう一つ別の曖昧性が潜んでいる．1と2(b)iiの部分で，“適切な規則を選択する”という部分である．図-6の2では，NP->PrpNを，また6では NP -> Det N を選択しているが，実際の計算機ではこのように都合よくは判断できない．

このように複数の規則の適用可能性をうまく扱う方法としては，(1)並行してすべての可能性を調べる方法と（2）とりあえず一つを選択し，他の可能性は後で扱う方法とがある．(1)はパラレル法に，(2)はシリアル法に適している．ここで

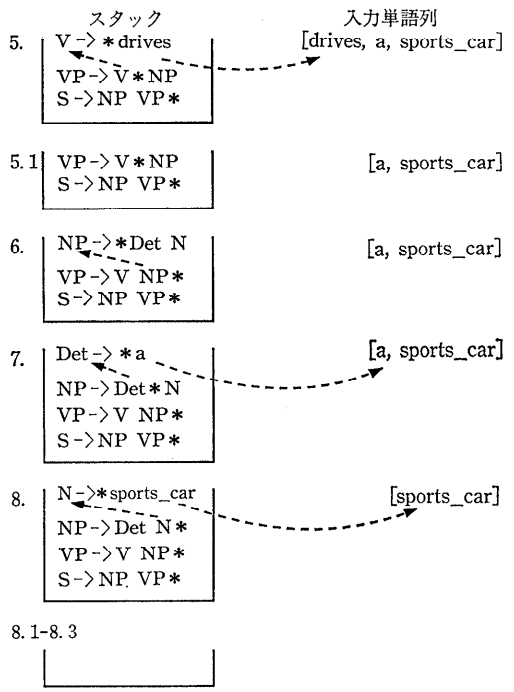


図-7　トップダウン解析の続き

は，より簡単なシリアル法の場合のヒントだけを示そう．

シリアル法では，**図-8** に示すような“バックトラック”という方法を用いる．曖昧性がある場合，とりあえずある一つ（通常は最初）の規則を選択して処理を進めていくが，他の可能性があることを“メモ”しておく．解析が途中で失敗したら，その“メモ”の場所まで後戻り（バックトラック）し，処理の状態を復元して，2番目の規則を選択して処理を続ける．つまり，先のアルゴリズムを，1で“メモ”を行い，2bで終了ではなく後戻りをするように変更すればよい．

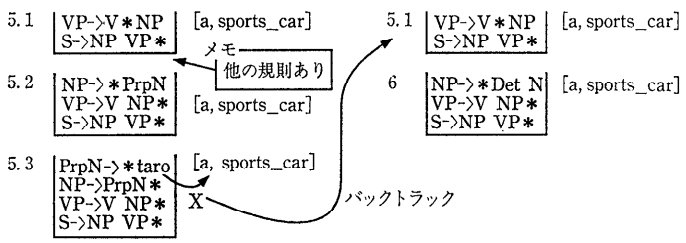


図-8 バックトラックの実行

図-8であれば，5.1から次に進む際に，とりあえずNP->PrpNを使って5.2に進んでおく．しかし，これは5.3で失敗するので，メモを頼りに5.1の状態まで“バックトラック”する．そして別の規則 NP -> Det N を使って，6の状態を得る．この選択が結果的に“適切な”選択となる．

[問 3] 図-1の(i)と(ii)の規則の間に，新たに NP->NP who VP という規則を追加すると，上記のアルゴリズムがどのように動作するか，考察せよ．なお，この規則は“Taro who drives a sports-car”のような関係代名詞句を扱うためのものである．

解答例：図-6の2の状態において，NP->*PrpNの代りに NP->*NP who VP がスタックに積まれる．そうすると*の右はNPなので，再びこの規則がスタックに積まれる．これを繰り返し，いつまでたってもアルゴリズムは終了しない．**2.1.1** で，このような規則を“左再帰”の規則と呼んだ．トップダウン・シリアル法では，新たな工夫をしないと左再帰の規則がうまく扱えない．

なお，複雑なPSGの場合は，一つの文に対して解析木が複数求まることがある．読者も英文を読んでいるとき，ある句がどの句を修飾しているかはっきりしなかった，という経験をおもちだろう．たとえば，

Taro drived a sports-car in Jiro's garage.

という文は，“次郎のガレージの中にあるスポーツカーを運転した”とも“次郎のガレージの中でスポーツカーを運転した”ともとれる．このような状況を“構文的曖昧性”と呼ぶ．可能な解析木をすべて求めること自体は，バックトラックを用いる場合，一つの解が得られても，強制的に失敗させれば求めることができる．どの解析木が“適切”かを判断するには **4.3**, **4.4** で説明する意味解析や文脈解析などが必要である．

4.2.2 構文解析システム

プログラムの振舞いをある程度理解した後は，実際に解析システムを動作させてみよう．

システム もし，身近なところで Prolog と呼ばれるプログラミング言語が利用可能であれば，比較的簡単に PSG の解析が実験できる．多くの Prolog 処理系には始めから確定節文法 (DCG) と呼ばれる形式で書かれた文法を用いて解析ができるようになっている．DCG は，PSG を基礎とした文法記述を Prolog でうまく解析できるようにしたものであり，F. ペライラらによって提案された[1]．DCG，特に Prolog の詳細については本講座の範囲を越えてしまうので，たとえば参考文献2) を参照願いたい．

DCG を用いて図-1に示した PSG の規則 ER 1 を記述すると，**図-9** のようになる．二つの図の対応関係は一目瞭然であろう．このように DCG による記述は，PSG をほぼそのまま書き下したものである．ただし，

1. すべての構文カテゴリと単語は小文字で始めること，
2. 書き換え規則の最後にピリオド ‘.’ をつけ

```
s --> np, vp.
np --> prpn.
vp --> v, np.
np --> det, n.
prpn --> [taro].
v --> [drives].
det --> [a].
n --> [sports_car].
```

図-9 DCG による規則 ER 1 の記述

```
1 | ?- s([taro, drives, a, sports_car], []).
2    1  1  Call: s([taro, drives, a, sports_car], []) ?
3    2  2  Call: np([taro, drives, a, sports_car], _280) ?
4    3  3  Call: prpn([taro, drives, a, sports_car], _280) ?
5    4  4  Call: prolog: 'C'([taro, drives, a, sports_car], taro, _280) ?
6    4  4  Exit: prolog: 'C'([taro, drives, a, sports_car], taro, [drives, a, sports_car]) ?
7    3  3  Exit: prpn([taro, drives, a, sports_car], [drives, a, sports_car]) ?
8    2  2  Exit: np([taro, drives, a, sports_car], [drives, a, sports_car]) ?
9    5  2  Call: vp([drives, a, sports_car], []) ?
...
14   8  3  Call: np([a, sports_car], []) ?
15   9  4  Call: prpn([a, sports_car], []) ?
16  10  5  Call: prolog: 'C'([a, sports_car], taro, []) ?
17  10  5  Fail: prolog: 'C'([a, sports_car], taro, []) ?
18   9  4  Fail: prpn([a, sports_car], []) ?
19   9  4  Call: det([a, sports_car], _1538) ?
...
27   8  3  Exit: np([a, sports_car], []) ?
28   5  2  Exit: vp([drives, a, sports_car], []) ?
29   1  1  Exit: s([taro, drives, a, sports_car], []) ?
30 yes
```

図-10 Prolog による解析の状態

ること，

3. 単語は，'[' と ']' で囲うこと，

という約束がある．

Prolog に図-9 が与えられているとき，たとえば文 ES 1 を解析するには，?- という入力要求に対して次の命令を与えればよい．

s([taro, drives, a, sports_car], []).

ここで，'[' と ']' でかこわれ，"," で区切られた単語の列をリストと呼ぶ．また，1 により Taro は taro と変えている．なお，第二引数の [] は，Prolog の処理の都合で必要なものであるが，詳細は参考文献 2) を参照願いたい．

Prolog は，入力文を前節で示したトップダウン・シリアル法とほぼ同じ方法で解析していく．この過程の主要部分を**図-10** に示す。Call/Exit は，それぞれスタックへの積み／降ろしに相当し，Prolog : C は単語の一致をみる部分に当たる．2-4 行で s，np，prpn と順にスタックに積み（左辺のみが表示されている．内部ではどの規則で右辺のどこまで処理が進んでいるかも記憶している），5 行目で taro が見つかると，6-8 行目でスタックから降ろす．9 行目から vp の処理を進めるが，そのときには単語のリストから taro が取り除かれている．

14 行目からは np の処理を行う．15-18 行でとりあえず prpn として処理を進めるが，失敗に終わる（18 行目の Fail）．そこでバックトラックが行われ，改めて np を det，n として 19 行目からの処理が行われる．このようにして，最終的に解析が終了し，成功したことを示す yes が出力される（30 行目）．もし失敗すれば no が出力される．

```
s(s(NP, VP)) --> np(NP), vp(VP).
np(np(PRPN)) --> prpn(PRPN).
vp(vp(V, NP)) --> v(V), np(NP).
np(np(DET, N)) --> det(DET), n(N).
prpn(prpn(taro)) --> [taro].
v(v(drives)) --> [drives].
det(det(a)) --> [a].
n(n(sports_car)) --> [sports_car].
```

図-11 解析木を求めるための DCG 規則の例

解析木の作成 これまでに示した解析方法では，解析に成功したかどうかは判定できる．しかし，解析結果として必要な解析木を得ることはできないので，このままでは役に立たない．解析木を作成する方法にはいろいろあるが，ここでは，DCG での方法を示そう．

DCG では，各構文カテゴリに "引数" と呼ばれる補助的な情報をもたせることができる．引数は，構文カテゴリの後に "(" と ")" をつけて記述する．この引数を用いれば簡単に解析木を作成することができる．たとえば，ER 1 に解析木を作成するための引数を付けると**図-11** のようになる．ここで，最初の規則 s(s(NP, VP)) --> np(NP), vp(VP). は，

np および vp で作られた部分的な解析木

NP および VP から解析木 s(NP, VP) を作れ

ということを示している．

この場合，NP や VP などのように大文字で始まる記号は変数と呼び，計算の中間結果を蓄える箱のようなものである．以降，大文字（変数）と小文字（定数）の区別に十分注意してほしい．また，左辺の s(NP, VP) は，**図-12** の左下の部分に示すように，解析木を Prolog の内部で表現する方法である．最初の s が解析木の上位の構文カテゴリを表し，その下に NP と VP の二つの部分的な解析木を入れる箱があることを表現している．図-12 に示すように，この箱には，右辺の np と vp の解析結果が変数 NP と VP により入れられ，最終的な解析木が作られる．

図-11 の規則と入力文を与えると Prolog は解析木を出力してくれる．**図-13** にその様子を示す．1行目の X は解析結果を求めるための変数であり，その値は2行目で表示されている．かっこが何重にも受け込まれているが，図-2 と同じ解析木になっていることを確認してほしい．

4.2.3 その他の手法

前項では，PSG に基づく構文解析法の基本的な考え方について解説した．しかし，そこで示したアルゴリズムには，図-3 で考えたように，単純には扱えない規則がある．また，バックトラックが最も最近の選択から順にやりなおすため，最初のほうの選択に間違いがあると，同じような計算を何度も繰り返すという処理効率上の致命的欠点があり，実用的ではない．

一般に，シリアル法は，メモリの使用量は少ないが処理時間が長くかかる．パラレル法は，メモリを多く使用するが，処理時間は短くてすむ．このことを踏まえて，優れたアルゴリズムが種々提案されている．特にボトムアップ法の CKY 法やトップダウン法のアーリー法，両者を融合できるチャート法，プログラミング言語の解析手法を拡張した一般化 LR 法などが知られている．

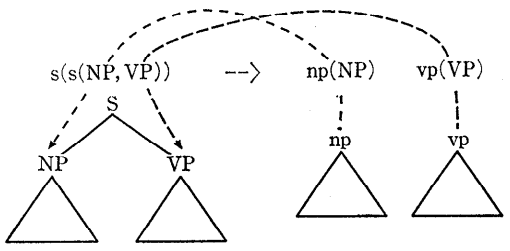


図-12 DCG 規則と解析木の図式表現

```
1 | ?- s(X, [taro, drives, a, sports_car], []).
2 X = s(np(prpn(taro)), vp(v(drives), np(det(a), n(sports_car)))) ?
3 yes
```

図-13 引数を用いた場合の解析結果

また，構文解析を行う手法には，PSG の書き換え規則を使わないものもある．その一つとして，W. ウッズが開発した ATN (Augmented Transition Network) と呼ばれるものがある．これは **3.** で述べた有限オートマトンを拡張したものであり，句の構造を記述すると同時に，解析手順も記述する．その他，木構造そのものを取り扱う手法もあり[2]，**2.1.2** で述べた変形文法の考え方をそのまま記述することができる．

以上，いくつかの手法を紹介したが，これらに興味のある読者や，本格的に構文解析を勉強したい読者は，たとえば参考文献 1)〜3) を参照してほしい．

4.3 意味解析

1.2 で述べたように NLP の歴史には大きく二つの流れ，すなわち MT（機械翻訳）志向と AI（人工知能）志向がある．**意味解析** (semantic analysis) の役割りも，この観点から以下の二つに分けることができる．

1. 構文的曖昧性がある場合に，正しいものを選択するため意味を利用する立場．**4.2.1** の最後で触れたが，文には，構文的に可能な解釈が複数ある場合がある．MT 志向の研究ではこの問題をどのようにして解決するかに悩まされているが，意味を利用すれば，一つに限定される場合がある．

2. 文の“意味”を解釈して，その結果を利用する立場．AI 志向の研究では，言語が表現する“意味”を捉えようとする．自然言語インタフェースであれば，利用者の要求を捉えて適切な応答をしなければならない．翻訳の場合であっても，原文の“意味内容”を捉えることにより翻訳を行う必要がある．意味解析は，入力言語の世界と別の世界（自然言語インタフェースではデータベースなどの操作の対象，機械翻訳では相手側の言語）とを橋渡しするものである．

むろん，両者を完全に分離することはできないが，以下では，この二つの観点をふまえて意味解析について説明しよう．

4.3.1 確定節文法と意味解析

まず，文の意味を解釈して，その結果を利用する例として，DCG を用いて簡単な自然言語インタフェースを作成してみよう．このシステムでは，**2.2.2** で説明した格文法の考え方を用いる．格文法は，文の意味を動詞を中心とし，動詞と主語や目的語の名詞の間の“動作主 (Agent)”や“対象 (Object)”などの意味的関係を捉えるものであった．このインタフェースは，“Taro drives a sports-car.”と教えてやると，その格構造を記憶し，“Who drives a sports-car ?”という質問に答えることができる，というものである．

システム　**図-14** にその文法を示す．書き換え規則は，図-11 に示したものとほぼ同じであるが，引数では，解析木のかわりに格構造を作るようになっている．たとえば，6-7 行目では，動詞 v の格フレーム [Predicate, [Agent, Object]] に目的語の np の意味を対象格として代入することを意図している．12 行目で，動詞 'drives' の格フレームは Agent と Object を格として支配すると記述した．一方，9-10 行と 14-15 行の規則は，名詞に対する意味を規定している．たとえば，9行目は，名詞 'taro' の意味は，簡単化のためその指示対象とみなして r_taro と記述した．これにより，“taro drives a sports-car” に対して，

[drive, [r_taro, r_sports_car]]

という形式の格構造を作れるようになっている．

図-11 と本質的に異なる点は，4行目にある．DCG では，'{' と '}' で囲われた部分は，構文構造を表現する書き換え規則の一部ではなく，単なる Prolog のプログラムの呼出しとして扱われる．これを“補強項”と呼ぶ．この部分が，PSG だけでは記述できないことを実現するための補強機能である．

補強項の使い方はいろいろあるが，この例では，意味解析結果の格構造を dbq というプログラムに渡して利用するために用いている．dbq は，17-18 行にある．これは，Prolog でデータベース (db) を扱うプログラムである．17 行目で，与えられた格構造がデータベースに登録されていればそのまま値として返し，そうでなければ，18行目で，その格構造をデータベースに追加登録 (assert) し，その時点でのデータベースの登録内容を表示 (listing) することを示している．なお，1行目の記述は，db をこのように動的に扱うとの宣言である．

実行例　この自然言語インタフェースの実行例を**図-15** に示す．1行目で“Taro drives a sports-car”を入力すると，2行目でデータベースにその格構造が登録されたことが表示され，3行目で“I see”と答えている．さらに5行目で“Jiro drives a recreational-vehicle”を入力すると，6-7 行目で，[drive, [r_jiro, r_recreational_vehicle]] がデータベースに追加登録されたことが分かる．

次に，10 行目で 'Who drives a sports-car ?' という質問をしている．この場合は，データベースを検索した結果，11 行目のように，'r_taro' であることを記述した格構造が答 (Ans) として示される．これは，図-14 の 17 行での検索の結果が表示されたものである．

意味解析とその利用　意味解析の例として簡単なシステムを説明した．これには意味解析で必要な以下の要素が含まれている．

1. 各単語とその意味の関係：図-14 の 9-15 行

```
1  :- dynamic db/1.
2
3  s(Ans) --> np(Agent), vp([Predicate, [Agent, Object]]),
4             {dbq([Predicate, [Agent, Object]], Ans)}.
5  np(Sem) --> prpn(Sem).
6  vp([Predicate, [Agent, Object]]) -->
7             v([Predicate, [Agent, Object]]), np(Object).
8  np(Sem) --> det(_), n(Sem).
9  prpn(r_taro) --> [taro].
10 prpn(r_jiro) --> [jiro].
11 prpn(R_who) --> [who].
12 v([drive, [Agent, Object]]) --> [drives].
13 det(_) --> [a].
14 n(r_sports_car) --> [sports_car].
15 n(r_recreational_vehicle) --> [recreational_vehicle].
16
17 dbq(Frame, Frame) :- db(Frame), !.
18 dbq(Frame, i_see) :- assert(db(Frame)), listing(db).
```

図-14　DCG による簡単な自然言語インタフェースの例

```
1  | ?- s(Ans, [taro, drives, a, sports_car], []).
2  db([drive, [r_taro, r_sports_car]]).
3  Ans = i_see ?
4  yes
5  | ?- s(Ans, [jiro, drives, a, recrcational_vehicle], []).
6  db([drive, [r_taro, r_sports_car]]).
7  db([drive, [r_jiro, r_recreational_vehicle]]).
8  Ans = i_see ?
9  yes
10 | ?- s(Ans, [who, drives, a, sports_car], []).
11 Ans = [drive, [r_taro, r_sports_car]] ?
12 yes
```

図-15　意味処理の実行例

で，名詞には指示対象を，動詞には格フレームを対応させている．

2. 構文構造と意味の関係：たとえば3行目で，構文的な主語と動作主が対応することが表現されている．

3. 意味とその使用方法の関係：4行目の補強項で，データベースの追加登録・検索を行うプログラムを呼び出している．

本格的な意味解析を実現するためにはこれらの要素のそれぞれの機能を拡張・強化する必要がある．1と2については，次項でより高度な記述形式の例を示そう．

3については，解析結果を目的にあわせて扱うプログラムをうまく作成すればよい．機械翻訳であれば，得られた格構造を翻訳先の言語の構造にうまく置き換えてやればよい．データベースの検索を自然言語で行うための自然言語インタフェースであれば，解析結果を対象としているデータベースの検索用言語に変換してやればよい．たとえば，**1.2.2** でふれた T. ウィノグラードの SHRDLU システムは，図-14 のプログラムの考え方を高度にしたものである．db で扱う対象として“積木の世界”を考え，英文の指示に従って，積木の世界の状態を変更させ，会話を行った．

4.3.2 単一化文法を用いた意味解析

次に UG と呼ばれる手法（制約に基づく文法と呼ぶ場合もある）を紹介しよう．これは，前項で示した文法をより拡張性のあるものにする方法であり，また，構文的曖昧性をうまく解消する方法でもある．

素性構造　UG は，PSG の構文カテゴリを“素性（そせい）構造”(Feature Structure) というものに拡張した文法である．FS は，素性名とその値（素性値）の組の集合である．動詞 ‘drives’ を FS で表現すると次のようになる．

```
FS1 : | sem : | predicate : drive                    |
      |       | agent : Fi | marker : human          |
      |       | object : Fj | marker : vehicle       |
      | agr : singular                                |
      | subj : | sem : Fi |                           |
      | obj : | sem : Fj |                            |
```

sem, agr などが素性名で，‘:’ の右がその値である．たとえば agr (agreement, 数の一致) は，単独の値 singular（単数）をもつ素性である．それに対し sem (semantic, 意味構造) は，FS を値とする．このように，素性値は，また，FS であってよい．

FS では，さらに“素性値の共有”という考え方を用いる．FS1 であれば，F_i により subj（構文上の主語）と agent（格構造上の動作主）の値が，F_j により obj（構造文の目的語）と object（格構造上の対象）の値が共有されることが示されている．FS は，実は**図-16** のように，グラフの構造をしているのである．このメカニズムにより，

> drive という動詞では，構文上の主語，目的語と格構造上の動作主，対象は，それぞれ同一のものである

という“構文構造と意味構造の対応”が直接記述できる．

単一化　構文カテゴリをこのように拡張したので，次は，カテゴリが“一致”するかどうかというチェックを拡張した“単一化 (unification)”という操作を考えよう．単一化は，二つの FS を“融合”する操作であり，条件チェックだけでなく結果の計算も同時に行うものである．単一化は以下のようにして行う．

1. 一方の FS にのみ存在する素性名とその値は，計算結果にそのまま取り込む．

2. 両方の FS に共通に存在する素性名については，

（a） その値が単独の値の場合は，一致すれば計算結果に取り込む．一致しなければ単一化は失敗．

（b） その値が FS の場合は，それらをさらに単一化してみる．単一化に成功すればその計算結果を取り込む．失敗すれば，全体としても失敗．

少し複雑であるが，要は，図-16 のようなグラフを二つ重ね合わせて新しいグラフを作る操作である．

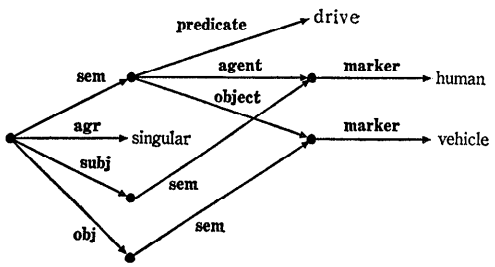


図-16 グラフで表現した素性構造

次の二つの FS の単一化を考えよう.

FS 2 : | lex : **r_sports_car** |
　　| marker : **vehicle** |

FS 3 : |marker : **vehicle**|

FS 2 と FS 3 は単一化でき，結果は，前者の FS 2 と同じになる．一方の FS にのみに存在する素性名は lex であるので，これはそのまま取り込み，共通の素性名である marker は値が一致するのでやはり取り込む.

[問 4] 'drive' に対する FS 1 と目的語 (obj) としての 'sports-car' を表現する

FS 4 : | obj : | sem : | lex : **r_sports_car** |
　　| | | marker : **vehicle** | | |

とは単一化できるか．素性の共有に注意せよ.

解答例：FS 1 と FS 4 は，単一化でき，

FS 5 : | sem : | predicate : **drive**
	agent : $\mathbf{F}_i$	marker : **human**
	object : $\mathbf{F}_j$	lex : **r_sports_car**
		marker : **vehicle**	
agr : singular
subj :	sem : $\mathbf{F}_i$
obj :	sem : $\mathbf{F}_j$

になる．一方の FS にのみ存在する素性名は sem, agr, subj であるので，これらはそのまま取り込む．共通する obj については，その値を単一化するが，sem も共通であるので，さらにその値の FS を単一化する．この場合，F_j により object と共有されているので，実際に単一化するのは，前の例の FS 2 と FS 3 であり，最終的に FS 5 が得られる.

これに対して，FS 1 と目的語としての 'taro' を表現する.

FS 6 : | obj : | sem : | lex : **r_taro**
　　| | | marker : **human** | | |

であれば，marker の値が一致しないため，単一化は失敗する.

素性構造と単一化を用いることにより，構文的および意味的な制限を自然に表現することができる．先の例では，**2.2.2** で述べた drive の対象格 (object) は意味マーカ (semantic marker) が "＋乗り物" (vehicle) であるという選択制限が自然に表現できている．ただし，[＋乗り物，－食物，－衣類] といった複雑な選択制限や意味素性の階層関係を扱うためには，別の処理が必要になる.

システム　ER 1 に相当する文法を記述した例を**図-17** に示す．このシステムでは，書き換え規則に，補強項で FS 間でどのように単一化を行えばよいかを表現している．なお，この記述は，単一化を行うプログラムを呼び出すようにした DCG 形式に変換して実行する必要があるので，残念ながら，このままでは，通常の Prolog システムに入力しても実行できない.

F などは変数であり，解析木のかわりに FS を受け渡しするようになっている．たとえば 5-6 行は，v と np から vp を作る点は ER 1 と同じであるが，その際，5 行目の F=F 1 で，vp に対する FS F は v の F 1 をそのまま引き継ぎ，6 行目の F@obj=F 2 で，属性名 obj の値と np の FS F 2 を単一化することを記述している．これは問 4 で考えた単一化を行うための記述である．なお，FS は 13-18 行目のように素性名を @ で区切ることにより表現している．この部分は FS 1 に対応している.

この規則を用いて 'taro drives a sports-car' を解析した結果は，**図-18** のようになる．図中の F 2 などは，共有構造を示している．この結果，解析によって意味構造 (格構造表現) の agent (動作主) が subj (主語) の r_taro であることが得られていることが分かる．UG についてより詳細に知りたい読者はたとえば参考文献 3)，4) を参照し

```
 1  s(F) --> np(F1),          {F@subj = F1},
 2           vp(F2),          {F = F2,
 3                             F1@agr = F2@agr}.
 4  np(F) --> prpn(F1),       {F = F1}.
 5  vp(F) --> v(F1),          {F = F1},
 6            np(F2),         {F@obj = F2}.
 7  np(F) --> det(F1),        {F = F1},
 8            n(F2),          {F = F2}.
 9
10  prpn(F) --> [taro],       {F@sem@lex = r_taro,
11                             F@sem@marker = human,
12                             F@agr = singular}.
13  v(F) --> [drives],        {F@sem@predicate = drive,
14                             F@sem@agent@marker = human,
15                             F@sem@object@marker = vehicle,
16                             F@agr = singular,
17                             F@subj@sem = F@sem@agent,
18                             F@obj@sem = F@sem@object}.
19  det(F) --> [a],           {F@sem@det = undet,
20                             F@agr = singular}.
21  n(F) --> [sports_car],    {F@sem@lex = r_sports_car,
22                             F@sem@marker = vehicle,
23                             F@agr = singular}.
```

図-17　素性構造を考慮した文法規則

てほしい.

4.4 文脈解析

これまでは，ある一文を解析する方法について考えてきた．しかし，実際には文が単独で用いられることはなく，ある脈絡をもった文の集り，すなわち文章中で用いられたり，ある実際の場面で用いられたりする．例として，図-15 の文をもう一度考えてみよう．最後の“Who drives a sports-car?”は，この1文だけを考えているかぎり，答は分からない．しかし，その前に文章があるので，答が Taro であることが分かる．また，実際に Taro が車を走らせているのを見ている場面であっても答は分かる．このように，それ以前の文や文が用いられている状況を考慮して解析を行うのが**文脈解析** (discourse もしくは context analysis) である．文脈解析は，現在研究中の分野であり，確立した手法はない．そこで，以下では，文脈解析上重要とみられるトピックをいくつか考えてみよう．

省略と照応　文脈に依存した言語現象としては，表現の一部を省略するという現象がある．以下の教授と学生の対話を調べてみよう．

A教授：おっ，(Bが)$_1$ いたいた．
　　(Bは)$_2$ ここにいたのか．
B学生：あっ，A先生．
A教授：いや，今，(Aは)$_3$ 君の部屋にいってみたんだけど，(Bが)$_4$ いなかったので，ひょっとすると (Bは)$_5$ ここじゃないかなと (Aは)$_6$ 思ってね．
B学生：あ，それで，何か (ご用でしょうか)$_7$.

‘( )’ 内を省略するのが，ごく標準的な会話であろう．しかし，これを英語に翻訳したり，内容に関する質問に答えたりするためには，省略されている内容が分からなければうまくいかない．人間ならば，( )内のことを，1，2 はその場の状況から，3-6 は，話し手と聞き手に関する状況から，7 は，“探しに来た”という行為から推測して理解するであろう．計算機にこのような推定を行わせる方法が課題となっている[3].

```
1  | ?- s([taro, drives, a, sports_car], [], F), fprint(F).
2  F0| subj: F1| sem: F2|  lex: r_taro    |  |          |
3    |          |         |marker: human|  |          |
4    |          | agr: singular             |          |
5    |sem: F3| predicate: drive                        | |
6    |         | agent: F2                             | |
7    |         | object: F4| marker: vehicle         | | |
8    |         |           | det: undet              | | |
9    |         |           | lex: r_sports_car       | | |
10   | agr: singular                                     |
11   | obj: F5| sem: F4      |                           |
12   |         | agr: singular |                          |
13 yes
```

図-18　属性構造をもたせた規則の実行結果

省略と類似の現象として，“それ”や“彼”といった代名詞を用いて，すでに述べられた表現を参照するというものがある．この現象は**名詞的照応** (nominal anaphora) と呼ばれている．文章を解析する場合には，代名詞が何を指し示しているかを決めてやらなければならない．照応現象のより複雑な例としては，以下のような例がある．

車$_1$ が2台駐車してあった．1台$_2$ は太郎のスポーツカーで，もう1台$_3$ は次郎の RV である．

この文章で，2，3の“1台”という表現は，1の“2台の車”のそれぞれを指していることが分からなければならない．このような解析を行う手法については，たとえば参考文献 5) に述べられている．

なお，照応を始めとする文脈に依存した言語現象を扱うための基礎理論として，“ディスコース表示理論” (discourse representation theory)，“状況意味論” (situation semantics) などいろいろなものが提案されている．これらの理論を理解するためには，論理学などの基礎知識が必要であり本稿の範囲を越える．詳細にについては，参考文献 6) を参照していただきたい．

文章構造　文章は単なる文の集合ではなく，その中になんらかのまとまりがある．その一つは，起承転結や序論・本論・結論といった“文章構造”である．文の構造を解析すれば個々の単語の役割が明確になるのと同様に，文章の構造を解析すれば個々の文の役割・意味が明確になる．また，長文の要約を行ったり，文章の推敲支援にも利用できる[4].

このような考え方は，たとえば D. ランメルハートにより“物語文法” (story grammar) として提案された[5]．この手法では，

物語→起　承　転　結
起→状況設定　登場人物の設定

のように文章の構造を PSG の規則で記述し，文を解析するのと同じように文章を解析する．この考え方は人間の心理モデルとの関係からも研究が行われたが，解析の立場からは，品詞に相当する文の特徴を抽出することが容易ではない，文章構造は

文構造に比べて制限が強くないなど，問題も多い．

常識の利用　物語文法は，文章を構文的に扱おうとするものであるが，文章や対話の内容を解析するには，“常識”などのいろいろな言語外の知識が必要になることも多い．空港の案内所での次の会話を考えてみよう．

客　　：7時10分の東京行きなんですが．
案内人：8番搭乗口です．
　　チェックインはお済みですか．

これだけの会話を理解するためであっても，(1)飛行機で旅行をする場合の一般的手順，(2)飛行機に乗るという目標，(3)この目標を達成するためには，チェックインしてから搭乗口に行くという行動計画，などの知識が必要である．初期の研究としてR. シャンクは，これらをそれぞれスクリプト (script)，ゴール (goal)，プラン (plan) という考え方で扱おうとした[6]．

スクリプトには，“飛行機旅行”といった個別の人間の行為ごとに，“予約する，航空券を買う，チェックインする，搭乗口に行く”といった標準的な出来事を記述しておく．これにより，スクリプトの一部に対応する文から次にどのような文が現れやすいかが予想できる．これを文脈解析に役立たせようというものである．しかし，人間の行動はかならずしも予定どおりには進まないので，ゴールとそれを達成するためのプランも知識として利用する必要がある．このように手法はAI志向の研究として現在も行われているが，膨大な個別的知識が必要である．このため，知識をどのようにして獲得するかが課題となっており，大規模なシステムの開発には至っていない．

今回は解析の方法について説明した．次回はこれとは逆に文や文章を作り出す方法について考えてみよう．

文　献

1) Pereira, F. C. N. et al.: Definite Clause Grammar for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence, Vol. 13, pp. 231-278 (1980).
2) Nakamura, J. et al.: Grammar Writing System (GRADE) of Mu-Machine Translation Project and its Characteristics, Proc. of COLING 84, pp. 338-343 (1984).
3) 堂坂浩二：対話登場人物を指示する日本語ゼロ代名詞の同定，情報処理学会シンポジウム「談話理解モデルとその応用」資料 (1989).
4) 中島　靖，渡川智浩，中村順一，吉田　将：表層表現の解析に基づく論文改訂支援方法について，電子情報通信学会言語理解とコミュニケーション研究会，NLC 92-6 (1992).
5) Rumelhart, D.: Notes on a Schema for Stories, Representation and Understanding: Studies in Cognitive Science (Bobrow and Collins Eds.), New York, Academic Press (1975).
6) Schank, R. C. and Abelson, R. P.: Scripts, Plans, Goals, and Understanding, Hillsdale, N. J., Lawrence Erlbaum (1977).

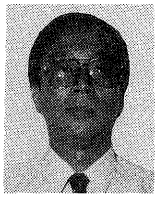
参考文献

1) Winograd, T.: Language as a Cognitive Process, Vol. 1, Syntax, Addison-Wesley (1983).
自然言語の構文処理に関する詳しい解説がある．特にATNについても詳細が示されている．
2) 田中穂積：自然言語解析の基礎，産業図書 (1989).
自然言語の構文解析手法に関する各種手法が第2章で示されており，さらに，第4章でPrologとDCGの紹介が詳しく説明されている．
3) 野村浩郷：自然言語処理の基礎技術，電子情報通信学会 (1988).
構文解析の各種手法が示されている．また，UGについても第6章で紹介されている．
4) Sells, P.: Lectures on Contemporary Syntactic Theories, CSLI Lecture Notes Number 3 (1985).
郡司隆男他訳：現代の文法理論，産業図書 (1988).
統率・束縛理論，一般化句構造文法，語彙機能文法が詳しく解説されている．中でもUGの初期の理論である語彙機能文法が参考になる．
5) Mellish, C. S.: Computer Interpretation of Natural Language Description, Ellis Horwood Limited (1985). 田中穂積訳：自然言語意味理解の基礎，サイエンス社 (1987).
照応の問題など，自然言語の意味理解をどのようにしてシステムとして構築すればよいかが具体的に示されている．
6) 白井賢一郎：自然言語の意味論，産業図書（平成3年).
自然言語の意味の扱い方に関する理論を概説している．特に第4章と第5章で文脈を扱うための理論が詳しく説明されている．

(平成5年8月23日受付)

岡田　直之（正会員）

1964年東海大学工学部卒業．1966年九州大学大学院工学研究科修士課程修了．同年同工学部助手，1976年大分大学工学部助教授，1978年同教授を経て，現在九州工業大学情報工学部教授．工学博士．人工知能の研究に従事．電子情報通信学会，人工知能学会各会員．



中村 順一（正会員）

1979年京都大学工学部卒業．1982年同大学院工学研究科博士後期課程中退．同年京都大学工学部助手，1989年九州工業大学情報工学部助教授．工学博士．自然言語処理，音楽情報処理の研究，計算機ネットワークの管理に従事．電子情報通信学会，ソフトウェア科学会，日本認知科学会，Association for Computational Linguistics 各会員．