自然言語解析システム LangLAB†

徳永健伸†† 岩山真††
上脇正††† 田中穂積††

ロジック・プログラミングに基づく文法記述形式として，これまでに Metamorphosis Grammar, DCG などが提案されている．これらは Prolog の計算機構を利用し，トップダウン・縦型探索で構文処理を行うため，左再帰規則が扱えない．本論文ではボトムアップ・縦型探索で構文処理を行う BUP を左外置処理が扱えるように拡張した BUP-XG システムに基づいた自然言語解析システム LanLAB について述べる．我々はすでに，文法記述形式として DCG と上位互換性を持つ XGS と呼ばれる形式を開発している．使用者の記述した文法，辞書は LangLAB に組み込まれたトランスレータで，それぞれ BUP-XG 節，TRIE 構造辞書に変換される．解析はこの変換結果と Prolog に組み込みの機能を用いて行う．我々は LangLAB に組み込みの BUP-XG トランスレータの高速化と変換結果である BUP-XG 節の最適化を行った．実験結果によると，従来の BUP-XG に比較して構文処理速度がインタプリタで 10 倍，コンパイルすると 4 倍高速になることを確認した．LangLAB で採用している辞書は TRIE 構造辞書と呼ばれるもので，辞書容量の節約，辞書引きの高速化に有効であることを明らかにした．また，TRIE 構造辞書を使うと複雑な熟語も柔軟に扱うことができる．以上のことから LangLAB は今後，自然言語処理システムを開発するための有力なツールとなると考えられる．

1. はじめに

筆者らはこれまで，ロジック・プログラミングの枠組みで自然言語処理の研究を進めてきた．ロジック・プログラミングの枠組みと自然言語処理との整合性が良いと考えたからである[1]．両者の整合性の良さは，フランスの Marseille 大学の Colmeraure の開発した Metamorphosis Grammar[2] において初めて明確に示された．

Metamorphosis Grammar では，拡張文脈自由文法の一形式である Metamorphosis Grammar で書かれた文法規則を，それと 1 対 1 に対応する Prolog プログラム（Horn 節）に変換する．変換後の Prolog プログラムに自然言語の文を与えて実行すると，トップダウンで縦型探索 (depth first) による構文処理を行うことができる．この方法によれば，Prolog インタプリタの機能で，パーサを代用することが可能になり，従来，構文処理をする場合に必要であったパーサの作成が不要になる．

Metamorphosis Grammar のもうひとつの重要な特徴は，自然な拡張によって，構文処理と意味処理とを融合した自然言語処理が可能になる，ということである．これは認知心理学の観点からも好ましいことであるとされている．

その後，イギリスの Edinburgh 大学の Pereira らは，Metamorphosis Grammar の考え方をさらに洗練した Definite Clause Grammar (DCG) を開発している[3]．DCG も Metamorphosis Grammar と同様，（構文処理と意味処理とを融合した）トップダウンで縦型探索の自然言語処理を行う Prolog プログラムに変換されるが，両者の方式にはひとつの大きな問題がある．それは，文法規則中に左再帰規則があると，構文処理の過程で無限ループに陥るという問題である．この問題は左再帰規則を右再帰規則に変換するか，ボトムアップな構文処理を行うことにより避けることができる．前者の方法では不自然な構文処理結果が得られることになるので，後者の方法を採用することが望ましい．

電子技術総合研究所の松本らの開発した方法によると，DCG の形式で書かれた文法規則を，ほぼそれと 1 対 1 に対応する Prolog プログラムに変換し，この Prolog プログラムに自然言語の文を与えると（構文処理と意味処理とを融合した）ボトムアップで縦型探索の自然言語処理を行うことができる[4]．これは，BUP システムと呼ばれている．

その後，東京工業大学の今野らは，BUP システムに左外置変形が扱えるような拡張を施した BUP-XG と呼ばれるシステムを開発している[5]．

† LangLAB: A Natural Language Analysis System by TAKENOBU TOKUNAGA, MAKOTO IWAYAMA (Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology), TADASHI KAMIWAKI (Hitachi Ltd.) and HOZUMI TANAKA (Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology).

†† 東京工業大学工学部情報工学科
††† (株)日立製作所

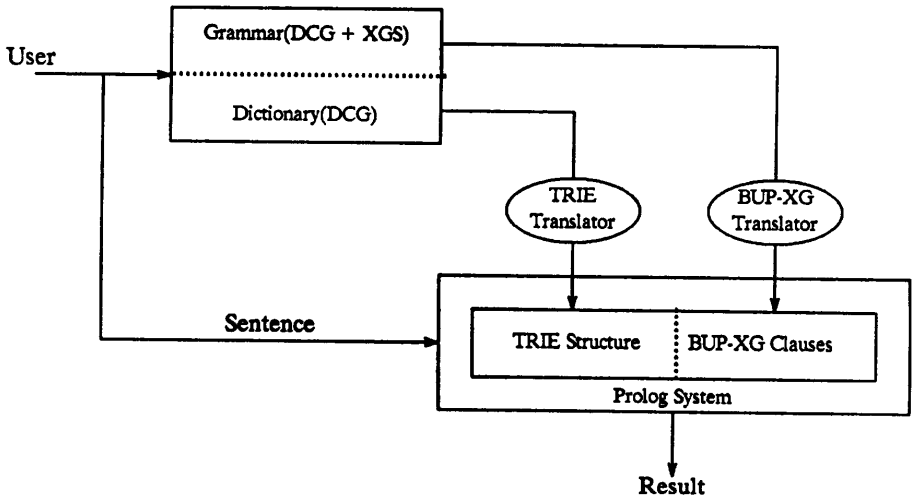

図 1 LangLAB の構成
Fig. 1 Structure of LangLAB.

本稿では，この BUP-XG を構文処理の基礎に置いた自然言語解析システム LangLAB について，構文処理を例にして説明する．LangLAB の構造は図 1 のようになっている．使用者は XGS 形式（後述）の文法と DCG 形式の辞書を用意する．文法，辞書はそれぞれ専用のトランスレータを用いて BUP-XG 節，TRIE 構造辞書に変換された後，Prolog システムに入力される．解析は Prolog に組み込みの機能をそのまま用いて行われる．

2 章では，DCG による文法記述形式と BUP システムの基本的な考え方を簡単に説明し，LangLAB で採用されている XGS と呼ばれる文法記述形式を説明する．さらに，XGS を BUP-XG 節に変換するトランスレータについて，その最適化手法について述べる．

3 章では LangLAB で採用している TRIE 構造辞書について説明する．TRIE 構造辞書を用いることによって，辞書容量の節約，辞書引きの高速化が得られる．

4 章では 2 章で説明した高速化手法の効果を検証するための実験結果を示す．LangLAB における構文処理速度は初期の BUP-XG[5] に比べてインタプリタで約 10 倍，コンパイルすると約 4 倍の高速化が得られている．文献 4)では，BUP システムにおいて，再計算を避けるアルゴリズムを導入することにより，構文処理速度がおよそ 1 桁改善可能なことが報告されているから，再計算を避けるアルゴリズムの導入以前の，最も初期の BUP システムに比べて LangLAB システムは，構文処理に関しておよそ 100 倍の高速化が得られていることになる．

2. XGS と BUP-XG

この章では LangLAB システムで採用している文法記述形式 XGS と構文解析メカニズム BUP-XG について説明する．BUP-XG の説明にはいる前に，その前身である BUP システムの動作原理について簡単に説明する．

2.1 BUP システム

BUP システムでは，DCG 形式で書かれた文法規則（図 2）を，BUP トランスレータによって，BUP 節と呼ばれる DCG 形式の文法規則と Prolog プログラム（後述する link 節，停止条件節など）に変換する（図 3）．この BUP 節は Prolog に組み込みの DCG トランスレータにより最終的には，Prolog プログラムに変換される（図 4）．図 4 のプログラムに現れる各非終端記号に対応した述語と述語 goal には，ふたつの引数が付加されている．これらは，処理すべき文の

```
s --> np, vp.          (2-1)
np --> pron.           (2-2)
pron --> [you].        (2-3)
vp --> [walk].         (2-4)
```

図 2 DCG による文法記述例
Fig. 2 Sample grammar written in DCG.

```
np(G) --> {link(np, G)},
   goal(vp),
   s(G).               (3-1)
pron(G) --> np(G).     (3-2)
dict(pron) --> [you].  (3-3)
dict(vp) --> [walk].   (3-4)
```

図 3 図 3 の文法規則を変換した BUP 節
Fig. 3 BUP clauses translated from Fig. 2.

```
np(G,X,Z) :- link(np,G),
    goal(vp,X,Y),
    s(G,Y,Z).                    (4-1)
pron(G,X,Y) :- np(G,X,Y).        (4-2)
dict(pron, [you|X],X).           (4-3)
dict(vp, [walk|X],X).            (4-4)
```

図 4 図3の規則を変換した Prolog プログラム
Fig. 4 Prolog programs translated from Fig. 3.

```
goal(G,X,Y) :-
  ( wf_goal(G,X,_) ;
    fail_goal(G,X), !, fail), !,
    wf_goal(G,X,Y).              (5-1)
goal(G,X,Y):-                    (5-2)
    dict(C,X,Y),
    link(C,G),
    P =.. [C,G,Y,Z],
    call(P),
    assertz(wf_goal(P)).
goal(G,X,Y) :-
    assertz(fail_goal(G,X)),!
    fail.                        (5-3)
```

図 5 goal 節の定義
Fig. 5 Definition of the goal clause.

差分リスト表現になっている．この Prolog プログラムに自然言語の文を与えると，ボトムアップで縦型探索の自然言語処理が行われる．

BUP システムにはボトムアップに構文処理を行うための述語 goal が用意されている．その定義を図5に示す．

図2の文法規則を用いて "you walk." という文を例にとり，構文処理の手順を説明する．

"you walk." という文の処理は，述語 goal の呼び出しにより開始する．

```
?-goal(s, [you, walk], [ ]).
```

これは，[you, walk] の後方から [ ] を差し引いた残りの部分（これを差分リストという），すなわち [you, walk] という文が将来 s というゴールを満たしうる（s という根をもつ構文木ができる）かどうかを調べることを意味している．

述語 goal の呼び出しにより，(5-1) が呼び出されるが (5-1) ではまず，成功結果と失敗結果とが，それぞれ述語 wf_goal と fail_goal として登録されているかどうかを調べ，無駄な再計算を避ける．

この例の場合には (5-1) の実行は失敗するので，つぎの (5-2) が選ばれる．(5-2) のボディでは，述語 dict(C, [you, walk], Y) を呼び出し，入力文の単語のリストXの先頭から辞書引きを行う．これは，図4の (4-3) とマッチするので，文法カテゴリCに pron が，Yには [walk] が返される．

(5-2) のボディの2行目では述語 link(pron, s) が呼び出されるが，これは構文処理が先に進み，構文木が下から上向きに成長したとき，pron からゴール s に到達可能かどうかを調べる述語である．link 節はあらかじめ BUP トランスレータによって文法規則から計算される．今の場合，link(pron, s) が成功するものとしよう．次は文法カテゴリ pron という述語を呼び出す：

```
P =.. [pron, s, [walk], [ ] ], call(P).
```

ここで call(P) は call(pron(s, [walk], [ ])) であることに注意．

pron(s, [walk], [ ]) の実行により，今度は (4-2) が呼び出され，そのボディの np(s, [walk], [ ]) が実行される．これは，np を根とする構文木が完成したことを意味している．np(s, [walk], [ ]) の呼び出しにより，次は (4-1) が呼び出される．(4-1) のボディでは link(s, s) が調べられるが，これは必ず成功する．link 節での検査の後，次の goal(vp, [walk], [ ]) が呼び出される．これは，np までの処理が終わり，次に処理すべき単語の系列が walkで始まり，その満たすべきゴールが vp であることを予測していることになる．

以下同様にしてトップダウンの予測を利用したボトムアップの構文処理が進み，最終的には停止条件節によりgoal 節の実行が成功する．停止条件節については文献 4) を参照されたい．さて(5-2)のボディの最後では，現在実行中のゴールが成功したので成功結果を述語 wf_goal としてアサートしておく．これにより，同じ計算がバックトラックにより繰り返される時には，(5-1) により直ちに成功結果を取り出すことができる．(5-3) は失敗結果をアサートしておくための節である．

2.2 BUP-XG システム

英語の関係代名詞節の埋め込み文は，平叙文中の名詞句がひとつ欠落した構造をしているが，これは先行詞が関係代名詞節の左方に移動してできたと考えられる．このような語句の移動を左外置と呼んでいる．この場合，移動した跡には痕跡を残すと考え，この痕跡をシステムに発見させることを前提にして文法規則を記述すると，文法規則の数や文法カテゴリの数が減り，文法の見通しが良くなるとともに，構文処理の速度も向上することが報告されている[5]．

トップダウンの構文処理システムであるATNG[6),7)] や XG[8)] には，このような痕跡発見機構が組み込まれ

ている．トップダウンに処理を行う場合には，次に処理すべき部分がどの非終端記号にまとめられるべきかが予測できる．したがって，特定の文法カテゴリ（たとえば np）が予測された時だけに痕跡の存在を仮定することができるので，効率良く痕跡を発見することができる．

一方，純粋にボトムアップの構文処理システムの場合には，そのような予測が不可能なので，処理すべき文の単語間のほとんどすべてに痕跡を仮定して処理を進めなければならなくなり効率が悪い．前節で述べたように BUP システムは，ボトムアップを基本としてはいるが，トップダウンの予測も利用している．今野はこの点に着目して，BUP システムに痕跡発見機構を組み入れた BUP-XG システムを開発している[5]．

BUP-XG システムでは，痕跡をサーチし発見するために，Pereira の XG[8] で導入されたXリスト (extraposition list) を用いる．それに伴い goal 節も変更されるが，その詳細は文献5)を参照されたい．LangLAB の使用者は，左外置が容易に記述できるようにDCG を拡張した文法記述形式 XGS を理解するだけで十分である．

XGS による文法記述例を**図6**に示す．この例のなかで，(6-3) の規則中の "../" (スラッシュと呼ぶ) が新たに拡張された記法である．XGS では，"srel../np" と記述することによって，カテゴリ srel の下に

```
s --> np, vp.                    (6-1)
np --> pron.                     (6-2)
np --> det, noun, srel../np.     (6-3)
vp --> vt, np.                   (6-4)
srel --> relpro, s.              (6-5)
```

図 6 スラッシュ・カテゴリを用いた文法記述例
Fig. 6 Sample grammar with slash category.

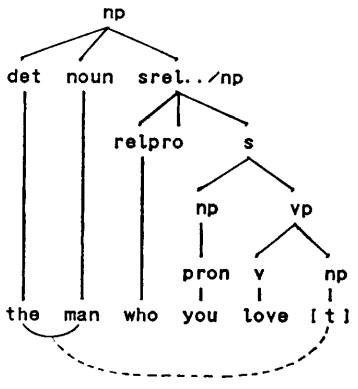

図 7 スラッシュ・カテゴリと痕跡の対応
Fig. 7 Matching between slash category and its trace.

痕跡を支配するカテゴリ np がひとつ存在することを表す．この記法は GPSG[9] の Slash category の考え方を参考にしたもので，XGS でもスラッシュの後のカテゴリをスラッシュ・カテゴリと呼んでいる．

また，規則 (6-3) は，det, noun, srel の3つカテゴリから名詞句 np ができることを表しているが，srel の下の痕跡は，det と noun からなる名詞句が移動してできたものと考えることができる．したがって，構文処理によって痕跡が見つかった場合には，**図7**に示すように埋め込み文中の痕跡 t と移動した語句 (the man) とが対応づけられる．

このように文法記述者は XGS で文法を記述すれば，LangLAB に組み込まれた BUP-XG システムによって，自動的にスラッシュ・カテゴリと痕跡との対応がとられる．

ところで，変形文法には，複合名詞句制約[10]という制約があるが，XGS ではこの制約を文法中に自然な形で記述できるように，次のようにカテゴリを"<"と">"(Pereira にならって，それぞれ open, close と呼ぶ[8]) で囲む記法が用意されている．

```
a --> b, c, <d>.
```

この規則はカテゴリ b，c，d から a ができることを示しているが，d を open, close で囲むことにより，語句の左外置に関しては，b，c の下からは a の外に語句が移動してもよいが，d の下からは，a の外に移動してはならないことを表している．図6の文法が複合名詞句制約に違反した文を受理しないように，規則 (6-3) に open, close を付け加えると次のようになる．

```
np --> det, noun, <srel../np>.
```

これにより，srel 中の（スラッシュ・カテゴリ np に支配される）痕跡は，det と noun とからなる名詞句以外とは対応づけができなくなる．

そのほかに，XGS には二重矢印 (==>) の記法，X リストを明示的に記述する記法（これらを使うと coordinate structure の記述が容易にできる）などがあるが詳細は文献5)を参照されたい．

2.3 BUP-XG トランスレータ

XGS で記述された文法は BUP-XG トランスレータによって BUP-XG 節に変換される．この時，BUP と同様に link 節，停止条件節も生成される．LangLAB では初期の BUP-XG システムに比べ最適化された BUP-XG 節を生成するようにトランスレータに改良が加えられている．この最適化に加え，使用者の

使いやすさを考慮して構文木の情報を自動付加できるように機能を拡張した．変換速度も規則数約200の文法に対して約3秒と非常に高速である．以下では，これらの改良について説明する．

2.3.1 link 節の変更

文法規則の数が多くなるにつれて，トランスレータにより生成される link 節の数は著しく増大する．ちなみに筆者らが使用している英語の文法規則の数はおよそ200であるが，トランスレータにより生成される link 節の数は約700に達している．したがって，link 節の検索時間の短縮は解析速度の短縮につながる．

link 節の呼び出しが起こるのは，BUP-XG 節のボディの先頭と goal 節の中である．いずれの場合も述語 link の2つの引数はアトムであることが保証されているので，カテゴリ a から b への到達可能性を表す，次の link 節

```
link(a, b).
```

を

```
a(b) :- !.
```

という形に変更することができる．これにより，述語名をキーとしたハッシュ検索が行われるので到達可能性の検査のための時間が大幅に減少する．LangLAB の BUP-XG トランスレータはこのような形式で link 情報を生成するように改良されている．

2.3.2 差分リストのインデックス化

2.1 節で述べたように，文の処理対象部分は差分リストとして表現される．また，構文処理過程で生じる中間的な成功結果と失敗結果とは，それぞれ wf_goal と fail_goal としてアサートされる．述語 wf_goal の最後の2引数は，処理対象部分文の差分リストになっているので，処理する文の長さが長くなると，wf_goal は長いリストを引数として持つことになる．また，文法規則の数が増えるとアサートされる wf_goal と fail_goal の数が増加するので，大きな記憶容量を必要とする．この記憶容量は，差分リストをインデックス化することで節約できる．また計算結果を再利用する場合にも，検索時間が短縮されるため，構文処理速度が改善される[11]．

具体的には，たとえば "you walk." という文が入力された段階で，

```
text(0, [ ]).
text(1, [walk]).
text(2, [you, walk]).
```

をアサートする．辞書引きはインデックスの対から述語 text を呼び出すことにより（処理対象部分文の）差分リストを復元してから行う．

2.3.3 wf_goal, fail_goal の分割

一般に解析する文章が長くなると，解析の途中で生成される wf_goal, fail_goal の数も増加し，その検索に時間を要する．wf_goal は引数として，解析に成功した（fail_goal では失敗した）入力文の部分単語列のインデックス（初期の版では差分リスト）とその解析結果を持つ．goal 節の中では，これから解析しようとする部分単語列のインデックスをキーとして，まず wf_goal, fail_goal を検索していた．LangLAB では，wf_goal, fail_goal という述語ではなく，解析に成功した部分単語列のインデックスを述語名として計算結果をアサートする．これにより，ハッシュを利用した検索を行い，検索時間を短縮している．

2.3.4 構文木情報の自動付加

入力された文章を解析するとき，解析結果として構文木を得たい場合が多い．BUP，および初期の BUP-XG システムでは使用者が文法を記述するときに各カテゴリの引数として構文木の情報を付加していた．しかしながら，構文木の情報は，機械的に付加することが可能であり，文法記述時には本質的なものではないので，トランスレータが自動的に付加してくれる方が望ましい．LangLAB の BUP-XG トランスレータは特に指示しない限り変換の際に構文木の情報を自動的に付加する．

3. TRIE 構造辞書

この章では LangLAB で採用されている TRIE 構造の辞書について説明する．TRIE 構造の辞書を用いることによって，辞書に必要な記憶容量の節約，辞書引きの高速化，柔軟な熟語処理が可能となる[12]．

3.1 TRIE 構造

TRIE 構造の辞書とは木の形をした辞書のことで，TRIE の名前は reTRIEval に由来する[13]．たとえば，**図8**に示す DCG 形式で記述された辞書を TRIE 構造に変換すると**図9**のようになる．これを図示すると**図10**になる．図10において根の t1-[c1] は終端記号 t1 がカテゴリ c1 に属することを表し，終端記号列 t1, t2 がカテゴリ c2 に属することを表している．なお，中間の枝に [c4]-[ ] と (prog)-[c3] というノードがあるが，このように文法カテゴリやプログラムなども [ ] や ( ) で囲むことにより TRIE 構造中に埋め込むことができる．また，他の語へのポインタ

```
c1 --> [t1].
c2 --> [t1, t2].
c3 --> [t1], c4, [t3], {prog}.
c5 --> [t1, t4, t5].
c6 --> [t1, t2, t6].
```

図 8 DCG 形式の辞書記述例
Fig. 8 Sample dictionary written in DCG.

```
dicta(t1,[[c1,[ ]]],[
    [[c4],[ ][
        [t3,[ ],[
            [(prog),[[c3,[ ]]],[ ]]]]]],
    [t2,[[c2,[ ]]],[
        [t6,[[c6,[ ]]],[ ]]]],
    [t4,[ ],[
        [t5,[[c5,[ ]]],[ ]]]]]).
```

図 9 図8を TRIE 構造に変換したもの
Fig. 9 TRIE structure translated from Fig. 8.

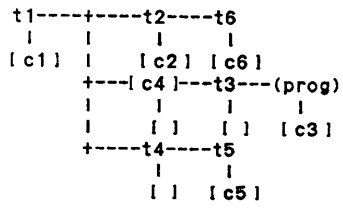

図 10 図9を図式化したもの
Fig. 10 Schematic representation of Fig. 9.

情報を持たせることによって，不規則変化動詞の変化形から原形の情報を参照することも可能である．辞書をこのような構造にすると，同一の単語を先頭に持つ複数の単語列はひとつの節にまとめられるので，記憶容量の節約になると共に，辞書引き時にも節レベルでのバックトラックが無くなり，辞書引きが速くなる．

3.2 TRIE 構造辞書による熟語処理

初期の BUP では，非終端記号や Prolog プログラムを含むような熟語は，その先頭の終端記号の部分と残りの非終端記号またはプログラムの部分とを分割し，前者を辞書項目，後者を文法規則として扱っていたため，記述方法が煩雑であった．しかし，本来，これらは分割して扱う必要はない．TRIE 構造辞書を用いることによって，このような熟語はすべて辞書項目として扱うことが可能となる．

具体例をあげて辞書の記述形式を説明する．**図 11** は動詞 get およびそれを含む熟語の記述例，**図 12** はそれを TRIE 辞書トランスレータによって変換したものである．このように，使用者は図 11 のように単に DCG 形式で熟語を含む辞書項目を記述すればよい．さて，図 11 において，左辺の第1引数，第2引数はそれぞれ辞書項目の構文

```
v(syn(get), sem(get)) --> [get].
v(ref(get, [[[vf|ed]],[ ]]),_) --> [got].
v(ref(get, [[[vf|en]] [ ]]),_) --> [gotten].
v(syn(get_up), sem(get_up)) --> [get, up].
v(syn(get on), sem(get_ on)) --> [get, on].
```

図 11 熟語を含む辞書の例
Fig. 11 Sample dictionary including idioms.

```
dicta(get, [[v,[syn(get), sem(get)]]],[
    [on, [[v,[syn(get_on), sem(get_on) ]]],[ ]],
    [up, [[v,[syn(get_up), sem(get_up) ]]] [ ]]]).
dicta(got, [[v,[ref(get, [[[vf|ed]],[ ]]),_]]],[ ]).
dicta(gotten, [[v,[ref(get,[[[vf|en]],[ ]]),_]]],[ ]).
```

図 12 図11の変換結果
Fig. 12 TRIE structure translated from Fig. 11.

情報，意味情報である．これを強調するために，辞書項目＊の構文情報，意味情報をそれぞれ，syn(＊)，sem(＊) と書いている．この中で辞書項目 got と gotten の第1引数は ref という構造体になっているが，これは構造体 ref の第1引数（この場合は get）へのポインタを意味している．不規則変化動詞の原形と変化形のように一部の情報が異なるような辞書項目を記述するときにはこのように記述する．構造体 ref の第2引数のリストはこの辞書項目（got または gotten）とポインタの先の辞書項目（get）との差分情報で，リストの最初が構文情報の差分（vf：動詞の変化形がそれぞれ ed：過去形，en：過去分詞形であることを示している），次が意味情報（この場合空リスト）の差分である．このような機構を設けることによって，多くの熟語をつくる不規則動詞について，その変化形に対する熟語をすべて登録する必要がなくなる（規則変化動詞の場合は，形態素解析プログラムが処理するので辞書項目には登録する必要がない）．

熟語の中には not only～but also～ のように熟語の一部に文法カテゴリを含むものもある．このような熟語の記述例を**図 13** に，またその変換結果を**図 14** に示す．図 14 のようにトランスレータは辞書項目中に Prolog プログラム（この例では {join(Np 1, Np 2, Np)}）があるとこれを（ ）で囲んだ形（(join(Np 1, Np 2, Np))）に変換する．辞書引きのプログラムの中では，（ ）で囲まれた Prolog プログラムを呼び出す

```
adj([ ] [ ]) --> [not, only],adj(_,_),
                 [but, also],adj(_ _).
np(Np, [ ]) --> [not, only], np(Np 1 _),
                 [but, also], np(Np 2, _), {join(Np 1, Np 2, Np)}.
```

図 13 右辺にカテゴリ，プログラムを含む辞書の例
Fig. 13 Sample dictionary with nonterminal symbols and programs in the rule body.

```
dicta(not, [ ], [
    [only, [ ], [
        [ [adj, _, _], [ ], [
            [but, [ ], [
                [also, [ ], [
                    [ [adj, _, _], [ [adj, [ [ ], [ ] ] ] ] [ ] ] ] ] ] ] ] ] ],
        [ [np, Np 1, _], [ ], [
            [but, [ ], [
                [also, [ ], [
                    [ [np, Np 2, _], [ ], [
                        [(join(Np 1, Np 2, Np)), [ [np, [Np, [ ] ] ] ], [ ] ] ] ] ] ] ] ] ] ] ] ] ] ]).
```

図 14 図 13 の変換結果
Fig. 14 TRIE structure translated from Fig. 13.

ことになる．また，辞書項目中の文法カテゴリ（たとえば，np((Np 1, _)) はリストの形に変換されて辞書中に埋め込まれる．辞書引きプログラムの中では，このようなリストの形に対しては，goal 節（この場合は，goal(np, [Np 1, _], X, Y)) を呼び出す．

以上のように，LangLAB では辞書として TRIE 構造辞書を採用した結果，熟語の扱いが非常に柔軟に行えるようになった．

4. 実 験

この章では，2.3 節で述べた BUP-XG 節の最適化の効果を確かめるために，付録に示す 10 の例文について実験を行ったので，その結果について考察する．実験の環境は以下のとおりである．

・使用計算機：Sun 3/260 ワークステーション
・Prolog　：Quintus-Prolog Release 1.6
・文　法：XGS 形式で 163 規則
・使用例文：付録参照

実験は各例文について，すべての構文処理結果が得られるまでの時間（単位はミリ秒）を以下の 5 つの場合に分けて測定した．

(1) 最適化する前の BUP-XG 節
(2) (1)＋link 節の変更 (2.3.1 節)
(3) (2)＋差分リストのインデックス化 (2.3.2 節)
(4) (3)＋wf_goal, fail_goal の分割 (2.3.3 節)
(5) (4)に形態素解析を含めたもの

このうち(1)から(4)までは，形態素処理を含まない構文処理だけに要した時間である．

表 1 にインタプリタによる結果，**表 2** にコンパイルした場合の結果を示す．表には，(1)最適化する前，(4)最適化した後，(5)最適化して形態素処理も含む場合，および，(1)と(4)の比，(4)と(5)の比をあげてある．2 つの表より，最適化前の BUP-XG 節に比べ，LangLAB の最適化 BUP-XG 節では，インタプリタで約 10 倍，コンパイルすると約 4 倍の高速化が得られていることがわかる．また，形態素解析を行うと処理時間が約 50% 増加している．この表からはわからないが，最も効果が大きいのはインタプリタに

表 1 インタプリタ・コードによる解析速度
Table 1 Analysis time using interpreter code.

例文	構文木	(1) 最適化前 [m sec]	(4) 最適化後 [m sec]	(5) 最適化後 (形態素解析) [m sec]	(1)/(4)	(5)/(4)
1	14	80415	8552	13565	9.40	1.59
2	4	18868	2700	4216	6.99	1.56
3	3	46700	4983	5999	9.37	1.20
4	1	30900	3600	5433	8.58	1.51
5	3	39634	4050	5767	9.79	1.42
6	4	95933	9550	17184	10.05	1.80
7	9	323167	26183	48146	12.34	1.84
8	2	87550	9349	12017	9.36	1.29
9	4	180300	15816	19367	11.40	1.22
10	1	116284	12083	15283	9.62	2.09
				平均	9.69	1.55

表 2 コンパイル・コードによる解析速度
Table 2 Analysis time using compiled code.

例文	構文木	(1) 最適化前 [m sec]	(4) 最適化後 [m sec]	(5) 最適化後 (形態素解析) [m sec]	(1)/(4)	(5)/(4)
1	14	20485	4134	6500	4.96	1.57
2	4	2467	1299	1550	1.90	1.19
3	3	4783	2284	2250	2.09	0.99
4	1	2884	1566	2217	1.84	1.42
5	3	4383	1917	2283	2.29	1.19
6	4	18768	4500	6949	4.17	1.54
7	9	127400	14000	26600	9.10	1.90
8	2	13450	4450	5050	3.02	1.13
9	4	59468	8216	8682	7.24	1.06
10	1	23650	5801	10900	4.08	1.88
				平均	4.07	1.39

おける link 節の変更であった．生成される link 節の数(この文法では約 700)を考えれば納得のいく結果である．コンパイルするとインタプリタほどは最適化の効果が顕著でなくなるが，システムのデバッグ作業がインタプリタで行われることが多いことを考えると，この最適化は十分意義がある．

また，他のシステムとの比較としては，奥西らによる SAX[14] と LangLAB との比較がある[15]．これによるとインタプリタでは，LangLAB の方が SAX に比べ 6〜10 倍速いが，コンパイルすると SAX の方が逆に 6〜16 倍速くなることが報告されている．同一の文法でも変換すると SAX の方が多くの節を生成するが，コンパイルによりハッシュが効く要素が多いため，このような結果になるものと考えられる．SAX は，現在デバッグ環境を整備中なので，その整備が終われば，文法規則の開発は LangLAB 上でインタプリタで行い，デバッグが終了した時点で，SAX の上でコンパイルして実行する，ということが考えられる．

5. おわりに

自然言語解析システム LangLAB について，その構文処理の道具立てについて説明し，処理速度の高速化手法およびそれを検証するための実験結果を示した．構文処理速度に関しては，筆者らは十分満足できるレベルに達したと考えている．LangLAB システムは，今後，自然言語処理システムを開発するための有力なツールとなるだろう．残された問題としては，文法規則開発用の特別なデバッガの開発が挙げられる．現在，文法規則のデバッグは Prolog のデバッガを使っているため，不必要に細かい部分までトレースをしてしまうなど，柔軟性に欠ける面がある．

今後は構文処理のレベルを越えた自然言語処理（たとえば意味処理）をロジック・プログラミングの枠組みの中で行うことを考えている．

参考文献

1) 田中穂積，松本裕治：自然言語処理における Prolog，情報処理，Vol. 25, No. 12, pp. 1396-1403 (1984).
2) Colmeraure, A.: Metamorphosis Grammar, in Bolc, L. (ed.): *Natural Language Communication with Computers*, pp. 133-190, Springer-Verlag, Berlin (1978).
3) Pereira, F. and Warren, D.: Definite Clause Grammar for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artif. Intell.*, Vol. 13, No. 3, pp. 231-278 (1980).
4) Matsumoto, Y. et al.: BUP—A Bottom-up Parser Embedded in Prolog, *New Generation Computing*, Vol. 1, No. 2, pp. 145-158 (1983).
5) 今野 聡，田中穂積：左外置を考慮したボトムアップ構文解析，コンピュータソフトウェア，Vol. 3, No. 2, pp. 115-125 (1986).
6) Woods, W. A.: Experimental Parsing System for Transition Network Grammar, in Rustin, L. (ed.): *Natural Language Processing*, Algorithmic Press, New York (1971).
7) Winograd, T.: *Language as a Cognitive Process*, Vol. 1: Syntax, Addison-Wesley, Reading (1983).
8) Pereira, F.: Extraposition Grammar, *Am. J. Comput. Linguist.*, Vol. 7, No. 4, pp. 243-256 (1981).
9) Gazdar, G. and Pullum, A. F.: Generalized Phrase Structure Grammar: A Theoretical Synopsis, Indiana University Linguistics Club (1982).
10) 新英語学辞典，研究社，東京 (1982).
11) 今野 聡，奥村 学，田中穂積：ボトムアップ構文解析システム BUP の高速化，日本ソフトウエア科学会第 1 回大会論文集，3 A-2 (1984).
12) 上脇 正，田中穂積：辞書の TRIE 構造化と熟語処理，*Proc. of Logic Programming Conference '85*, ICOT, pp. 329-340 (1985).
13) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: *Data Structures and Algorithms*, Addison-Wesley, Reading (1983).
14) 松本裕治，杉村領一：論理型言語に基づく構文解析システム SAX，コンピュータソフトウェア，Vol. 3, No. 4, pp. 4-11 (1986).
15) Okunishi, T. et al.: Comparison of Logic Programming Based Natural Language Parsing Systems, *Proc. 2nd International Workshop on Natural Language Understanding and Logic Programming*, Vancouver, pp. 90-102 (Aug. 1987).

付録 例文

1. She was given more difficult books by her uncle.
2. Be careful not to break the vase when you put it down.
3. The structural relations are holding among constituents.
4. It is not tied to a particular domain of applications.
5. Diagram analyzes all of the basic kinds of phrases and sentences.

6. This paper presents an explanatory overview of a large and complex grammar that is used in a sentence.
7. The annotations provide important information for other parts of the system that interprets the expression in the context of a dialogue.
8. For every expression it analyzes, diagram provides an annotated description of the structural relations holding among its constituents.
9. Procedures can also assign scores to an analysis, rating some applications of a rule as probable.
10. These properties provide important pieces of information for the other part of the system that interpret the phrase when it is used in a discourse.

(昭和62年11月 4 日受付)
(昭和63年 5 月10日採録)

徳永 健伸（正会員）

1961年生．1983年東京工業大学工学部情報工学科卒業．1985年同大学院修士課程修了．同年(株)三菱総合研究所入社．1986年東京工業大学大学院博士課程入学．1987年より同大学工学部情報工学科助手．自然言語処理，知識表現に興味を持つ．日本ソフトウェア科学会，認知科学会各会員．

岩山 真（学生会員）

昭和39年生．昭和62年東京工業大学工学部情報工学科卒業．現在，同大学院修士課程に在学中．自然言語処理に興味を持つ．日本ソフトウェア科学会会員．

上臨 正（正会員）

昭和37年生．昭和60年東京工業大学工学部情報工学科卒業．昭和62年同大学修士課程修了．同年(株)日立製作所入社．現在日立研究所に勤務．マルチプロセッサの研究に従事．

田中 穂積（正会員）

昭和39年東京工業大学理工学部制御工学科卒業．昭和41年同大学修士課程修了．同年電気試験所（現，電子技術総合研究所）入所．昭和58年東京工業大学工学部情報工学科助教授．昭和61年同大学教授となり現在に至る．工学博士．人工知能，自然言語処理の研究に従事．電子情報通信学会，認知科学会，日本ソフトウェア科学会．人工知能学会，計量国語学会各会員．