6 高機能な構文解析器に向けて

-HPSGのための実用的な構文解析器-

鳥澤健太郎
東京大学大学院理学系研究科

高機能な構文解析器とは何か？

本稿では，我々の研究グループで開発を行っている主辞駆動句構造文法[1)]（Head-driven Phrase Structure Grammar，以降**HPSG**と略す）ベースの文法，および構文解析器の解説を行う．タイトルにもある高機能な構文解析器とは，自然言語アプリケーションが必要とする情報を入力文からシステマティックに抽出できる構文解析器という意味で使っており，我々が現在開発している文法，構文解析器はそのようなシステムを目指すものである．

我々がフレームワークの核に据えたHPSGは1980年代に盛んに研究された「単一化文法」と呼ばれる一群の文法枠組みの1つである．本章ではとりあえず，この単一化文法をはじめとする文法や，構文解析の枠組み一般の研究の経緯と，我々がHPSGに注目した理由を簡単に説明してみたい．

これまでの研究

一般に文法と呼ばれるものには，一群の単一化文法のほかに，文脈自由文法，あるいは正規表現などの正則文法などさまざまなものがあり，それぞれ自然言語処理への応用が試みられてきた．主たる応用は，それらの文法を使った構文解析である．構文解析とは，使用する文法によっても異なるが，入力となる自然言語の文の文法的，あるいは意味的構造を解析するものである．より具体的にこれを見るため，企業などでの人事異動の情報を新聞から抽出するアプリケーションシステムを考えてみよう．おそらく，そのようなシステムでは，次のような文が入力されることが予想される．

[例文1] 社長にはおそらくA氏が就任する．
[例文2] A氏が社長に就任するようだ．
[例文3] 社長に就任するのはA氏の可能性が高い．

これらの文は，それぞれかなり異なった文法的な構造を持っているが，その意味するところはほぼ同じである．実際のアプリケーションでは，これらの多様な文から文全体が「就任」という出来事に関する推量を表し，就任するポストが「社長」で，就任する人が「A氏」であるという情報を得る必要が出てくる．自然言語処理における構文解析といった場合には，本来このような情報を得ることを目的として行われるプロセス，あるいはその一部を指す．

たとえば，文脈自由文法で例文1を構文解析すると，**図-1**にあるような構文木を得ることができる．しかしながら，このような構文木を得ただけでは，アプリケーションが必要とする「就任する人」といった情報は得られない．このような情報を得るためには，構文解析以外に木構造を探索するプログラムを書く必要がある．例文1の場合であれば，そのようなプログラムを書くのは簡単に思えるが，例文3のような場合がほかにも無数にあることを考えると，そのようなプログラムを書くことはかなりの労力が要求されることが分かる．このような状況は，文脈自由文法ではなく，正規表現を用いた場合でも似たりよったりであり，多様な言語表現から意図した情報を抽出するため，大量の正規表現を書き下ろす，いわゆる力づくの作業が必要とされることになる．

一方，単一化文法では，アプリケーションが必要とする情報を得るための道具立てが文法の枠組みに用意されている．この道具立ては意味表現と呼ばれ，単一化文法を使って構文解析を行った場合には，構文木が生成されるのと同時にこの意味表現が生成される．従来，意味表現は論理式の形式で書かれることが多く，たとえば，例文1には‘推量（就任（A氏，社長））’のような意味表現が割り当てられる．この意味表現では，‘就任’の第1引数が就任する人で，第2引数が就任するポストであるということが仮定されているわけである．

仮に例文1～3までのような多様な文に対しても同一の意味表現がシステマティックに生成できるとすれば，アプリケーションサイドが必要とする情報を容易に抽出できるわけだが，単一化文法を適切に書けば，まさにこのようなことが可能になる．

以上のような望ましいシナリオと共に登場した単一化文法ではあったが，1990年代に入ると研究は下火になった．これには下のような原因がある．

1. 現実の文を解析できるような適用範囲の広い文法の開発が非常に困難であった．
2. 構文解析に時間がかかり，実用的なシステムでは使用できなかった．
3. どのようなアプリケーションにも使える万能の意味表現が定式化できなかった．したがって，タスクごとに意味表現の仕様を決め，その意味表現を生成できる文法を開発する必要があったが，これも非常に困難な作業であった．

まず，最初の問題は，1980年代に主流であったDCGなどの単一化文法の枠組みに原因があった．というのも，これらの文法では，現実的な文法に数百個以上のルールが必要だったのである．とりあえず，数百個のルールを書いてみることはできるが，それらのルールの間で矛盾がないようにすること，また，3．の問題とも絡むが，適切な意味表現が常に生成されるようにするのは至難の技であった．また，2番目の問題と絡んで，以上のような困難な作業を終えても，書かれた文法を用いた構文解析にはかなりの時間がかかり，新聞の文を使ったテストでさえままならないという状況が常であった．さらにタスクに依存しない意味表現の仕様が決まらないという問題は現在でも解決されておらず，筆者などは永久に解決されないのではないかとすら思う．より現実的な解決策は，タスクに依存した意味表現の仕様を素早く決め，それらをやはり素早く文法に反映する技術ということになると考えている．

なぜHPSGか

以上を踏まえると，1980年代における単一化文法の研究の状況は，文法開発の方法論，および構文解析のシステムが，理想に追いつけなかったと要約することができる．我々の研究グループの目的は，以上のような問題点を解決，あるいは回避し，自然言語処理システムを作成するためのインフラストラクチャを構築し，実際にシステムを作成することである．我々はこの目標を達成するために，文法の枠組みとしてHPSGを採用した．HPSGとは言語学者のSagらによって発案された

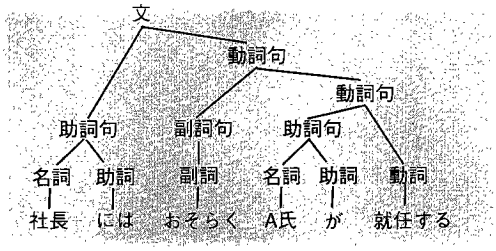

図-1　構文木の例

文法枠組みであり，上述の問題点の解決に有利な性質をいくつか持っている．1つは現実の文を処理できる文法が比較的容易に書けるということである．我々はすでに，意味表現こそ備えていないものの，新聞の文の約98%を妥当な精度で解析できるコンパクトなHPSGベースの日本語文法を開発している．また，解析速度に関しても，高速な抽象機械の作成，文法のコンパイル手法の導入などにより，新聞の文1文の解析が平均250ミリ秒程度でできるところまできている．たとえば新聞に現れた文「円高は，輸出にブレーキをかける役割を果たしているが，外国品が安く簡単に手に入るという利点も目立ってきたわけだ．」といった文に対しては約360ミリ秒で，可能な構文木をすべて列挙することができる．

より詳しく見ていくと，適用範囲の広い文法が容易に作成できたのは，HPSGのアーキテクチャによるところが大きい．先ほど，従来の単一化文法では数百のルールが必要であったと述べたが，HPSGではこれらが高々10個程度しかない．この10個程度のルールは，従来であれば数百のルールに分散して書かれていた言語の原則的性質だけを記述している．HPSGでは，それ以外の情報は，各々の単語に与えられる辞書記述に書かれることになる．ただし，我々の現在の文法では，特殊な振る舞いをするいわゆる機能語にだけ辞書記述を与えており，それ以外の単語に関しては，単語ごとではなく，品詞ごとにその品詞の語の一般的な振る舞いを記述した辞書記述のテンプレートとして与え，記述をコンパクトにしている．機能語の辞書記述とテンプレートの両者を合わせた数は約150個程度である．ここで，文法のルールを数百個書くのと，辞書記述を150個書くのでは，大差ないのではないかというのが当然の疑問として生じると思われるが，我々の経験からすると，一般的なルールを書くよりも，辞書記述，あるいはそのテンプレートを書く方が容易である．これは，辞書記述間の相互作用がルール間の相互作用に比べて非常に限定されたものになるからである．端的な例を挙げれば，従来の単一化文法でルールを

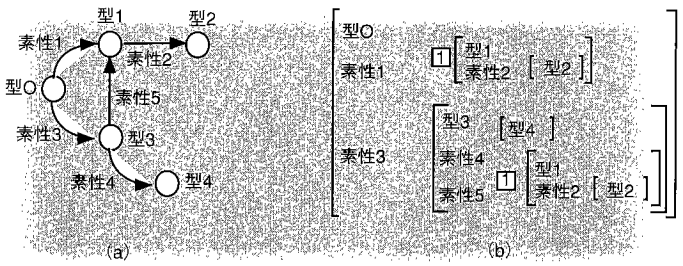

図-2　素性構造の例

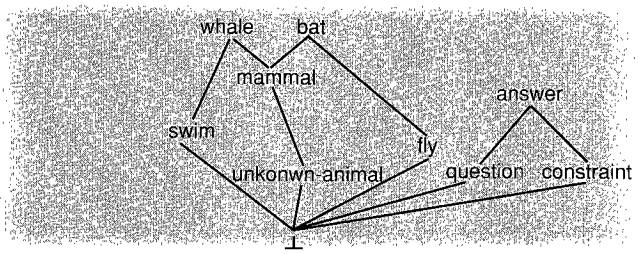

図-3　型階層の例

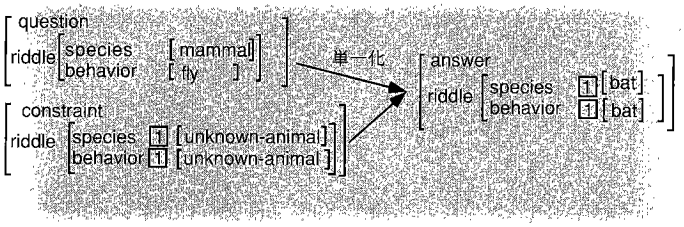

図-4　単一化の例

1つ修正すると，その副作用がどこに現れるかというのは，非常に判りにくく，文法のデバッグが非常にやりにくいことになる．一方でHPSGで辞書記述・テンプレートを修正した場合には，副作用が現れるのはその語，あるいはその品詞の語が現れた場合のみであり，また，10個程度のルールに書かれた原則だけを知っていれば，デバッグは比較的容易に行われる．少々極端なたとえではあるが，このようなHPSGと従来の単一化文法との差は，オブジェクト指向のプログラミング言語と，構造化すらされていないプログラミング言語との差に類するものと捉えることもできよう．

しかしながら，我々の開発したコンパクトな文法は意味表現の生成を行うものではなく，アプリケーション作成時の問題の一部しか解決していない．実際のアプリケーションで使用する際には，意味表現の生成を行う部分をさらに付加するという作業をする必要がある．現在，我々はこの意味表現の付加のプロセスを，例文を使って半自動化し，アプリケーション作成の労力を減らそうと考えている．しかしながら，現時点ではこの半自動化には未解決の問題もあり，本稿の最後で，研究の見通しを述べるにとどめる．そこで述べるように，このような枠組みを考える上でもHPSGの性質は良い方向に作用する．

以下では，HPSGの説明と，これまでに我々が開発したシステムの解説を行い，最後に今後の研究の見通しとして意味表現生成機構作成の半自動化について述べる．

HPSGとは？

HPSGを一口で説明すると，次のようになろう．

- 文法は，principle，ID-schemaと呼ばれる少数のルール群と辞書記述からなる．
- 前章で述べたように，ルール群は自然言語の一般的な性質のみを記述し，詳細な情報はほとんど辞書記述に書かれる．
- 文法の構成要素はすべて型付き素性構造（Typed Feature Structure）[2]という形式を使って記述されている．文の解析，生成は型付き素性構造間の単一化によって行われる．

型付き素性構造というのは，有向グラフで表現される．図-2 (a) の例にあるように有向グラフの各ノードに「型」と呼ばれるラベル，また，各アークには「素性」と呼ばれるラベルがふられている．通常は，図-2 (a) のようなグラフ表記は使わず，図-2 (b) にあるようなAVM形式と呼ばれる表現法で記述される（以下でAVM形式を用いる場合には素性構造の型を省略することがあるので，気を付けてほしい）．図-2 (b) 中の[1]は構造共有のタグと呼ばれるものである．同一のタグがつけられている複数の素性構造は構造共有しているといわれる．この意味は，図-2 (a) から分かるように，タグのついた素性構造がまったく同一であることであり，Prologなどの論理型言語における変数共有と同様の機能を果たす．型に関しては，図-3にあるような型階層が与えられる．この型階層では，型はリンクを上に辿っていくほど詳細な情報を表している．この階層によって，型の最小上界という概念が与えられる．最小上界とは複数の型が与えられたときに，その複数の型より詳細で最も一般的な型である．たとえば，図-3では，型mammalと型flyという型の最小上界はbatということになる．このような型階層は前述した辞書記述のテンプレートのような不確定部分を多く含んだ記述を簡潔に与えるのに役立つことになる．

さて，上で述べたようにHPSGの文法構成要素はすべて素性構造で記述され，文の解析，生成な

どの操作は，素性構造間の単一化によって行われる．直観的に述べると，単一化とは与えられた2つの素性構造をマージする操作である．素性構造が素性を持っている場合には，同一の素性の値に対して再帰的に単一化を適用することになる．また，マージされたノードの型はオリジナルの2つの素性構造が持つ型の最小上界であり，最小上界が存在しなければ単一化は失敗する．図-4は，図-2の型階層に従った単一化の例である．また，単一化では，構造共有が単一化を引き起こしていることに注意してほしい．HPSGの文生成，解析などは，この構造共有によって引き起こされる単一化によって，情報のチェック，やりとりを行うことになる．

HPSGは，辞書記述と少数のID-schema，principleと呼ばれるルール群からなる．これらはすべて型付き素性構造を用いて記述される．また，HPSGでの構文解析では構文木のノードに素性構造を対応させる．ID-schemaとprincipleはそれぞれ，構文木中の親子間の制約を記述したものということができる．図-5にID-schemaの例を示す☆1．この例から分かるように，ID-schemaは文脈自由文法の書き換えルールの非終端記号を型付き素性構造で置き換えたものである．つまり，左辺の素性構造は構文木中の親ノードに対応し，右辺の2つの素性構造は子ノードに対応する．ボトムアップの構文解析を行う場合にはすでに生成されたノードの素性構造がID-schemaの子ノードに単一化される．このとき，ID-schema中で，子ノードが親ノードのHEAD-DAUGHTERとCOMPLEMENT-DAUGHTERという素性と構造共有を行っているため，親ノードの素性構造からそれらの素性構造を参照することができるようになる．

従来の単一化文法や通常の文脈自由文法では，日本語文法を書こうとすれば数百のルールが必要となるが，現在我々が記述している日本語文法では，以上のようなID-schemaが8個，また，英語の場合にも10個しかない．これは品詞などの詳細な情報がID-schemaレベルでは捨象されてしまっているからである．たとえば，文法を書くときに，英語の目的語を伴う動詞句と前置詞句は異なるルールによって扱うのが通常の考え方であるが，HPSGでは，両方とも図-5にあるID-schemaで扱われることになる．もちろん，動詞句と前置詞句は異なる性質を持っているわけだが，この差は次に説明する辞書記述によって表現されることになる．このようなアーキテクチャは前章で述べた文法記述の容易さにつながることになる．

図-6に英語の動詞‘eat’に対する辞書記述の例を示す．まず，HEAD素性がVという型を持っている．これは，‘eat’が動詞であることを意味している．また，SUBCAT素性の値が要素2つのリストになっていることに注意してほしい．各々の要素はHEAD素性がNという型を持っており，名詞句に対応する素性構造を意味している．これらは，後の解析例でみるように，‘eat’という動詞が主語，目的語としてそれぞれ名詞句をとるということを意味している．また，素性CONTの値は意味表現を表す素性構造である．以上のように辞書記述はさまざまな素性を持つが，これらの素性の値の具体的な使用方法は次に説明するprincipleによって規定される．また，我々が開発した日本語文法で使われている辞書記述のテンプレートも，まったく同じフォーマットで記述することができる．principleは図-7にあるように素性構造で記述される．ボトムアップの解析を考えると，これらの素性構造は，ID-schemaが生成する親ノードの素性

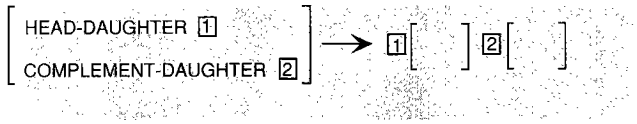

図-5 ID-schemaの例

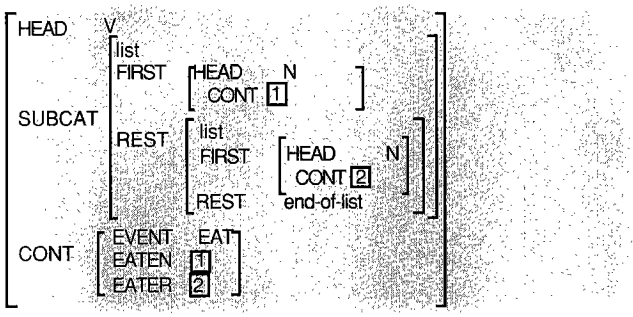

図-6 辞書記述の例

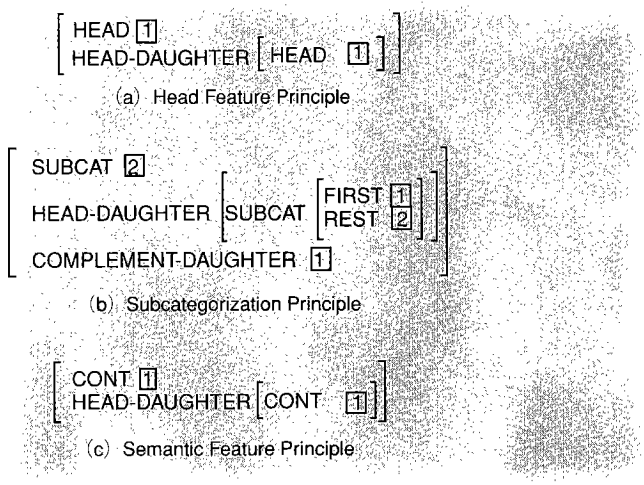

図-7 principleの例

☆1 説明を簡略化するため，文献1）と若干異なるID-schemaの定式化を与えている．詳細に興味のある方は参考文献をあたってほしい．

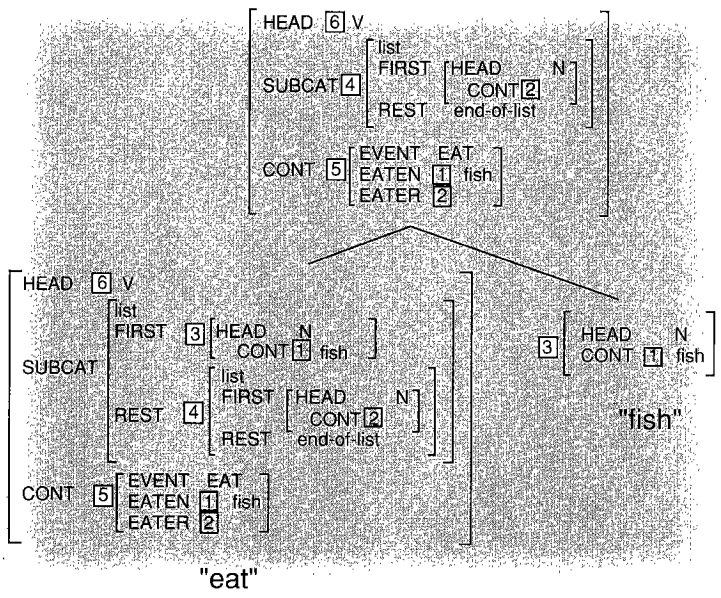

図-8　動詞句eat fishの解析例

構造に単一化される．ID-schemaの親ノードと同様に，principleの素性構造も，HEAD-DAUGHTERなどの素性を持っている．また，それらの素性の値の間で構造共有を行っているが，これらの構造共有で引き起こされる単一化によって，構文解析に必要な品詞などのチェック，情報のやりとりが行われることになる．また，これらのprincipleには品詞などといった詳細な情報がいっさい書かれていないが，そのような情報はすべて先の辞書記述に書かれることになる．

まず，図-7 (b) のSubcategorization Principleを見てほしい．この素性構造では，HEAD-DAUGHTER中のSUBCATのリストの要素が，COMPLEMENT-DAUGHTERの値と構造共有している．たとえば，先の‘eat’の辞書記述の場合だと，この構造共有による単一化によって，COMPLEMENT-DAUGHTERにくる目的語が名詞句であるかどうかというチェックが行われることになる．図-8は動詞句‘eat fish’の解析例である．解析は，eatとfishの辞書記述を図-5のID-schemaの子ノードにそれぞれ単一化し，さらに親ノードの素性構造にprincipleをすべて単一化することによって行われる．先ほど述べたように‘eat’の目的語が名詞であるという制約を‘fish’が満足しているかのチェックが単一化によって行われている．さらに‘eat’の素性構造のCONTの中に，名詞fishのCONTの値が取り込まれ，食べられたのがfishであるという意味表現が作られている．さらに，図-7 (a) (c) にあるprincipleによって，‘eat’の意味表現，品詞が句‘eat fish’に対する意味表現，品詞に受け継がれている．

さて，以上で述べたようなHPSGの特性を適切に利用すると，他の文法枠組みでは困難だったことがいくつか可能になる．その1点目は，本稿の冒頭でも述べたように辞書記述に大半の情報が記述されているための，文法記述の容易さである．2点目は，やはり本稿の冒頭で述べた意味表現を生成できる文法の半自動的な生成につながるもので，例文を用いた未知語に対する辞書記述の推論である．たとえば，文‘I eat fish’といった文があったときに，‘I’‘fish’に対しては辞書記述がある一方，‘eat’が未知語であったとする．ここで，‘eat’に記述をまったく含まない空の素性構造を与え，構文解析をすると，‘I eat fish’を1つの文とするのに必要な情報がもともと空であった‘eat’の辞書記述に単一化によって流れ込み，図-6にあるのと似た辞書記述を得ることができる．また，‘eat’が動詞であることがあらかじめ分かっていれば動詞の一般的な振る舞いを記述した辞書記述のテンプレートを空の素性構造の代わりに割り当て，前述の手法でテンプレートを‘eat’の性質に合うよう特殊化することもできる．これはHPSGでの解析が単一化だけで行われ，また，単一化という操作が双方向性を持っていることに起因する．さらに辞書記述に情報が集中しているため，多様な例文からの辞書記述の推論が可能である．以上，HPSGの解説と特徴の説明を簡単に行った．次章ではHPSGのための構文解析器と文法開発の現状を解説する．

構文解析器と文法の開発状況

本稿の冒頭で述べたように，我々の研究グループでは，今までのところHPSGの高速な構文解析器と適用範囲の広い文法を開発してきた．HPSGでの解析速度が遅い第一の理由は，型付き素性構造の単一化が時間を要する操作であるということである．我々はこの単一化を高速に実行できるようにするため，まず文法記述・構文解析器開発用のプログラミング言語LiLFeS[3]を設計，その処理系を開発し，さらに，その上で構文解析器を実現した．LiLFeSとは，Prologの項の代わりに型付き素性構造を使用できるようにした言語で，その処理系はGUIや高効率な構文解析器を実現するための種々の組み込み述語を備えている．また，処理系には2種類のものが開発されている．1つはProlog実行のための抽象機械WAMとCarpenterらによって提案された素性構造処理のための抽象機械を融合し，さらに拡張したものであり，実行はバ

イトコードエミュレータによって行われる．もう1つは同じくPrologのネイティブコードコンパイラを参考に作成したIntel Pentium用のネイティブコードコンパイラである．このコンパイラでは抽象解釈などの導入によりさらに性能向上を図っている．後者のネイティブコードコンパイラによって出力されたコードの実行は前者のバイトコードエミュレータでのコードの実行よりも2倍程度高速である．しかし，そのバイトコードエミュレータも既存の素性構造処理エンジンALE，ProFITに対して4倍から2倍程度の実行速度を出している．

しかしながら，LiLFeSを使って常識的な構文解析アルゴリズムを記述しただけでは応用に必要な速度を実現するのは難しい．さらなる高速化を図るため，部分評価によってHPSGを文脈自由文法にコンパイルし，その文脈自由文法を使って高速に構文解析を行う手法や，素性構造のメモリ中での表現を正規化し，単一化の処理からスタック操作を除去する手法，さらには，選言を表現できる素性構造を用いて複数の素性構造をパッキングする手法などを開発している．特に文脈自由文法へのコンパイル手法は，後述の日本語文法を使用して，新聞に現れる文を平均250msec程度で解析することを可能にしている．この解析は可能な構文木をすべて数えあげる全解探索である．今後，解析時の枝刈りの導入により解析時間はさらに数分の1となろう．

文法に関しては，日本語，英語のための文法を開発している．日本語のための文法[4]はHPSGの特徴をフルに活用した形で開発した文法である．つまり，前述したように少数のID-schemaやprincipleは十分詳細なものを与えるが，辞書記述については，各品詞に属する単語の一般的な振る舞いを記述したテンプレートと，ごく少数の特殊な振る舞いをする機能語の辞書記述しか与えていない．数としては，両者合わせて約150個程度である．現在の性能は**表-1**にある．また，現在の構文解析器では，可能な構文木をすべて数えあげ，その中から，単純なヒューリスティクスによって，最も良い構文木1つを選ぶということを行っている．表中には，この方法で選ばれた構文木の中で，いわゆる係受けがどの程度正確であるかという数値も載せた．表中の文法Aは，解析精度を上げるための付加的なprincipleを含み，精度は高いが，解析に失敗する文が多い．また，文法Bでは付加的なprincipleを使用しておらず，解析に失敗する文は少数であるが，解析精度は低い．これらの精度に関しては，既存の統計的構文解析や，長期間にわたって文法，辞書をチューニングした構文解析器

文法	解析可能な文の割合	解析可能な文での係受け精度
文法A 付加的principleあり	93.5％	80.4％
文法B 付加的principleなし	98.3％	75.1％

実験対象：EDRコーパス中からランダムに選び出した20295文
（各文は主として新聞の文からなり，平均21.3語からなる）

表-1　日本語文法の性能

よりも数％から10％低い値が出ているが，大多数の語に辞書記述のテンプレートしか与えていないこと，および構文木選択手法の単純さを考えるとまずまずの値であると考えている．今後，統計的手法の併用や，次章で述べる辞書記述の学習，あるいは，既存の辞書を利用して辞書記述テンプレートをより詳細化し，具体的な辞書記述を作成するなどの手法によって，徐々に改善されていくものと考えている．

英語文法は日本語の場合とは異なり，ペンシルバニア大学で開発された英語文法XTAGをHPSGのフレームワークに移植することで作成した．現状では適用範囲，解析速度など満足のいく性能でないが，日本語文法の開発で試した手法を導入することによって今後改善を図っていく予定である．以上説明した我々のシステムのほかには，素性構造処理エンジンとして，LiLFeSとの比較でも言及したALE，ProFITや，LKB，PAGE，FF-DDMなどといったシステムが各国の研究機関で開発されている．我々のシステムはこれらの処理系よりも高い効率を誇っているが，文生成器などそれらのシステムから採り入れるべき要素も多々ある．文法についても，我々の文法より小規模ではあるが，Stanford大学の英語文法ERGOなどが知られている．これはHPSGの理論的側面に焦点を当てた文法であり，意味表現のフォーマリズムなど今後参考にしていく予定である．

今後の展望：文法の半自動的なチューニングに向けて

以上では，これまでの我々の研究の現状について述べた．本章では，今後に向けて，意味表現の取り扱いの研究での見通しについて述べてみたい．本稿冒頭で述べたように，現状では，アプリケーションごとに意味表現を設計し，その意味表現が生成されるように文法に手を加える必要がある．我々はこのプロセスを半自動化することによって，アプリケーション作成に必要な労力／コストを減らそうと考えている．ただ，この手法の実現には，

まだ解決しなければいけない問題が残っており，現段階ではあくまで「シナリオ」として読んでもらいたい．とりあえず，本稿冒頭の例文に戻る．

[例文1] 社長にはおそらくA氏が就任する．

[例文2] A氏が社長に就任するようだ．

[例文3] 社長に就任するのはA氏の可能性が高い．

前述したように，例文1～3はほぼ同じことを意味していると考えることができ，アプリケーション作成を容易にするためには，例文1～3から同一の意味表現が生成されることが望ましい．とりあえず，説明を単純化するため，例文1から生成されるであろう意味表現が決まっており，例文に共通して現れている「社長」「A氏」「就任する」といった語の辞書記述には例文1からその意味表現を生成するだけの記述が与えられていると仮定する．ここで例文2，3から例文1と同一の意味表現を生成させるのは，例文1に現れていない単語である「ようだ」「可能性」「高い」といった語に新たな辞書記述を与えることによって可能になる．この新たな辞書記述を推論するには文法の説明で述べた未知語に対する辞書記述の推論と同様の手法が使える．より具体的には，「ようだ」のような辞書記述を推論したい単語に対して，辞書記述のテンプレートを与えた上で，既存の文法を使って例文2と例文3の構文解析を行う．その結果得られた素性構造に例文1から得られる意味表現を単一化すれば，単一化の双方向性によって，必要な素性構造がそれぞれの単語に割り当てられたテンプレートに流れ込むことになり，意味表現を生成するのに必要な辞書記述を得ることができる（実際にはもう少し工夫が必要だが，その工夫の基本的な中身に関してはすでに研究がなされている）[5]．推論される「ようだ」や「可能性」などの辞書記述には，例文1の意味表現から，例文1～3に共通して現れている単語「社長」「A氏」「就任する」の意味表現を差し引いたものが含まれ，「おそらく」の意味表現に相当するものが含まれることになる．また，構文解析に必要な文法的な性質の記述や，意味表現を組み上げるために必要な構造共有も含まれることになる．

一見すると，このように例文を与えて，要求する意味表現が生成されるように辞書記述を推論させていくのは，それなりの労力が必要であり，いっそのことHPSGなど忘れて，正規表現などを書き下ろしても作業量はそう変わらないのではないかと思われるかもしれない．しかしながら，以上のように推論された辞書記述は，文法の他の要素と組み合わされて，多様な文を処理することができる．たとえば，「B氏を課長に選ぶ」と「B氏が課長に就任する」を含む他の例文の集合から，「選ぶ」の辞書記述が推論されたとする．この辞書記述は，先に推論した「ようだ」の辞書記述やHPSGのルール群に書かれた語順に関する原則に従って，与えられた例文にない文，たとえば「おそらく社長にA氏が選ばれる」や「社長にA氏が選ばれるようだ」などからも例文1と同じ意味表現を生成させることができる．この点は必要な例文の削減につながり，また，正規表現などでは不可能な点でもある．何よりも，例文の作成は自然言語処理の専門家でなくても十分に可能であり，この点も正規表現などを書き下ろす手法とは大きく異なる．また，いったん推論された辞書記述のうち一般的なものについては，異なるアプリケーションで再利用することも可能であろう．

もちろん，辞書記述の推論の段階で，自然言語の曖昧さによって引き起こされる構文解析の誤りなどにより，不適切な辞書記述が生成される可能性もある．しかしながら，こういった問題は可能な限り短い例文を作成する，または，必要に応じて例文作成者への問合せを行うなどといった手段で解決できると考えている．また，意味表現以外の文法的な部分についても例文から情報を得ることができるため，例文から得られた辞書記述を使って，構文解析の精度向上を図ることも可能であろう．

以上，本稿ではHPSGという文法枠組みを説明し，我々が開発した構文解析器，および文法の現状を解説した．また，今後の研究の方向として，辞書に与える記述を例文から推論し，必要な文法のチューニングを行う枠組みについて述べた．最終的なゴールである実用的なアプリケーションをローコストで作成することの実現には，まだ解決すべき問題が多いが，徐々に解決していこうと考えている．

謝辞 議論および推敲に付き合ってくださった東京大学大学院理学系研究科辻井研のメンバに感謝いたします．

参考文献

1) Pollard, C. and Sag, I.: Head-driven Phrase Structure-Grammar, The University of Chicago Press (1994).
2) Carpenter, B.: Logic of Typed Feature Structures, Cambridge University Press (1992).
3) Makino, T., Yoshida, M., Torisawa, K. and Tsujii, J.: LiLFeS - Towards Practical HPSG Parsers, in Proceedings of Coling 98 (1998).
4) Mitsuishi, Y., Torisawa, K. and Tsujii, J.: HPSG-style Underspecified Japanese Grammar with Wide Coverage, In Proceedings of Coling 98 (1998).
5) Torisawa, K. and Yonezawa, A.: Lexicion Acquisition in HPSG, Techinical Report 94-10, Department of Information Science, University of Tokyo (1994).

（平成11年3月10日受付）