Vol. 33 No. 7　　情　報　処　理　　July 1992

解　説

自然言語処理技術の最近の動向

自然言語の構文解析と文生成の統合†

橋　田　浩　一††

1. 序　論

構文解析や文生成*などの個別的作業に専用のアルゴリズムが自然言語処理の重要な研究テーマであったのは，10年以上前のことである．そもそも，人工知能一般においてゲームとかプランニングなどのための個別的なアルゴリズムが重要な研究対象でなくなってから久しい．知的であることの本質は，個別的で閉じた情報処理にあるのではなく，個別的な処理の柔軟な統合にある．しかし，個別的な問題解決のための処理方法が数多く開発されてきているのに対し，それらの処理を統合することができない．その統合の方法を求めることが，今や自然言語処理のみならず人工知能一般における最大の課題のひとつである．

本稿では，その統合が実は自然言語処理においては言語の理解と産出の統合を含意することを指摘し，特に構文解析と文生成の統合に関する研究の動向について述べる．理解と産出の統合にもさまざまな立場があり，控え目な立場としては，たとえば Kay[20]や McDonald[22]のように，共通の辞書や文法から理解用の辞書や文法，および産出用の辞書や文法をそれぞれ自動生成する，という方法が考えられる．しかしここでは，最も徹底的な立場，すなわち，**辞書や文法を含む知識も処理手続きも，理解と産出で共有する**という立場を取る．後に示すように，柔軟な言語処理機構を実現するには，このような**強い意味での統合**が必要となる．

理解と産出を強い意味で統合するということは，言語処理モデルの中に，理解に専用の手続きとか産出に専用の手続きがないということだから，特に，構文解析アルゴリズムとか文生成アルゴリズムは存在しない．意味とかプランやゴールなどの処理はいずれにせよいわゆる常識を用いた一般的な推論を含まざるをえないから，理解と産出に共通と考えるのが自然だろう．しかし，構文解析と文生成には専用の確立されたアルゴリズムがあるのだから，それを使ったほうが共通の一般的な手続きで処理するよりはるかに効率がよいのではないかと思われるかもしれない．以下ではこれが間違いであることを示す．

2.では，強い意味での統合の必要性について述べ，情報の流れをあらかじめ限定しない設計法としての制約プログラミングの重要性を指摘する．**2.**の内容の補足として，橋田[11]，橋田と竹沢[12]なども参照されたい．**3.**では，制約に基づく文法理論について述べ，特に，計算の中間状態を明示的に措定しない方法を論ずる．**4.**では，個別的なアルゴリズムによって明示された効率のよい処理過程が，実は一般的な処理方法から自然発生(emerge)することを示す．

2. 多様性と制約

いかなる立場にせよ，理解と産出を統合するということにはそれなりの工学的意義がある．たとえ理解用の辞書と産出用の辞書が異なるものであっても，それらが共通の原辞書から機械的に作られる場合には，理解できる語彙と産出できる語彙とが一致することを体系的に保証できるだろう．文法に関しても同様である．理解できる言い回しと産出できる言い回しが同様のものなら，原辞書

† Integration of Parsing and Generation of Natural Language by Koiti HASIDA (Natural Language Section, Machine Understanding Division, Electrotechnical Laboratory).

†† 電子技術総合研究所知能情報部自然言語研究室

* 言語の「理解 (comprehension)」と「産出 (production)」は，意味的な情報や社会的な関係にわたる一般的な情報処理過程を含むとする．これに対し，「構文解析 (parsing)」と「文生成 (generation)」は，それぞれ理解と産出のうちで統語的な情報に関する処理を意味する．特に「文生成」は，本特集の徳永氏の解説にあるようなプランニングを含まず，所与の意味表現から文を生成することである．この用語法は本稿のものであり，一般的ではない．

と原文法をひととおり開発することによって理解の機能と産出の機能を一挙に実現できる.

しかし,理解と産出の統合は,このような意味で単に望ましいだけではなく,設計の複雑さを現実的な範囲に抑えるために是非とも必要であり,特に,人間の言語使用と同様の柔軟性を実現するには,強い意味での統合が不可欠である.人工的に設計されるシステムに関しても,進化や学習を経て自然に設計される人間の認知システムに関しても,理解と産出の統合は,柔軟な言語処理メカニズムを設計するための工学的要請である.それは,理解においても産出においても,どのような優先順序でどのような種類の情報をどのように組み合わせて処理するかという,処理の制御法が,「重要な情報を優先的に処理せよ」というような非常に一般的な形でしか述べられない,ということである.そのような一般的な制御法は,理解と産出に共通のものとならざるをえないだろう.もちろん,理解または産出に限定した形でもっと具体的に制御法を明示することも理論的には可能なはずだが,実際にそれを行おうとすると,設計が過度に複雑になり,システムがモジュール性を失って巨大化し,管理不能になってしまう.

それは一般に,認知過程における情報の流れがきわめて多様だからである.たとえば,言語表現の理解において統語的な情報のほうが意味的な情報より常に優先的に利用されるとは限らない.どの情報が優先されるかは文脈に依存する.ところが,文脈とは参照可能な情報の分布の仕方のことだから,文脈の種類は莫大であり,それに応じて情報の流れ方もきわめて多様である.したがって,たとえば「ここでこの統語的な情報とこの意味的な情報をこう使え」というような具体的な形で情報処理の制御を記述しようとすると,記述の量が爆発し,設計が破綻するのである.逆に言えば,そのような明示的な形で処理の制御を記述し,しかも設計の複雑さを実際的に管理可能な範囲に抑えると,情報の流れが過度に固定されてしまい,十分な柔軟性が実現されない.

こうした多様な情報の流れを現実的な設計の複雑さの範囲内で実現するには,まず,設計において情報が流れる向きや順序を捨象することが必要である.情報の流れを捨象した情報の表現のことを**制約** (constraint) と呼ぶ.言語表現の理解と産出の間の差は情報が流れる向きの違いであるから,制約に基づく設計においては理解と産出は統合されることになる.

情報の流れ方を設計の中に明示しないで,多様な文脈に応じた多様な情報の流れを実現するには,たとえば,システムと環境との接面およびシステムの内部における情報の「活性度」の分布として文脈をとらえ,おのおのの情報の活性度を他の関連する情報の活性度に依存させ,さらにおのおのの処理をそれが参照する情報の活性度に応じて発火させる,というような制御の方法[15]が必要と考えられる.しかし,その方法の詳細は本稿の範囲を越える.以下の議論は,それとは一応無関係に,構文解析と文生成に限定した範囲での情報処理の制御法について述べる.

3. 文法と制約

言語使用を制約によってとらえる文法理論☆を,制約に基づく (constraint-based) 文法と言う.これに対立する概念は,手続きに基づく (procedure-based) 文法である.GPSG (Generalized Phrase Structure Grammar)[9], LFG (Lexical Functional Grammar)[11], HPSG (Head-Driven Phrase Structure Grammar)[25], JPSG (Japanese Phrase Structure Grammar)[10] などは,文や談話の構造を一種の論理プログラムとして記述する文法と考えられ,制約に基づく文法の色合いが強い.これに対し,変形文法 (Transformational Grammar)[2)~4)]は変形 (transformation) によってひとつの構造から別の構造を派生するという手続き的な記述を用いており,手続きに基づく度合が高い.

ただし,A という構造から B という構造を得る変形とは,実は A と B との関係についての制約であり,言語処理において A が B に時間的に先行することを要請しない.つまり,変形文法において一貫して主張されてきたとおり,変形は情報が流れる向きを限定しないのである.

しかし,それでもやはり変形は言語処理モデルの設計において望ましくない.それは,変形が,計算の中間状態に明示的に言及するという意味においてやはり手続き的だからである.一連の変形を用いた説明は,変形の入出力となる多くの中間的な構造に言及することになる.そのような中間

☆ 文法とは,統語論,意味論,語用論を含む概念である.

状態を明示しない説明のほうが単純で優れた説明であるから，それが可能な場合には変形の妥当性は低い．変形を用いて記述されてきた主要な現象として，非有界依存（unbounded dependency）*があるが，GPSG と LFG では非有界依存は中間状態を使わずに説明されており，GPSG の手法は HPSG や JPSG にも継承されている．その簡単な解説としては橋田と竹沢[12]を参照されたい．そういうわけで，変形を本当に実装した自然言語処理システムは皆無に近く，著者の知るかぎりではただひとつ[8]しかない．

しかし，制約に基づく文法として上にあげた諸理論においても，中間状態を用いる説明がしばしばあり，その中には中間状態を使わなくても済む場合がある．たとえば，句構造（phrase structure）または構成素構造（constituent structure）は，不要かもしれない中間構造の一例である．つまり，**図-1** のような句構造において，節点 S と VP は係り受けを処理するための中間構造であり，不要である可能性もある．実際，句構造を用いない統語論のアプローチとして，依存文法（dependency grammar）[26]がある．依存文法とはいわゆる係り受け文法のことであり，図-1 の S や VP のような節点を V とは別のものとして措定せず，'Tom' の NP も 'Mary' の NP も直接 V に係ると考える．しかし，自然言語の文法に句構造が本質的に必要かそれとも依存文法でよいかはまだ決着していない問題である．

また，'every' や「ほとんど」などの量化子（quantifier）の作用域（scope）を処理するための方法として量化子保管（quantifier storage）[5]というものがあり，HPSG などでも用いられている．ところが，量子化保管とは，構文木を下から上へ辿りながら見た場合，まだ作用域の定まっていない量化子を一時的に保管しておくということであり，ボトムアップな計算の中間状態の明示的な表現にあたる．

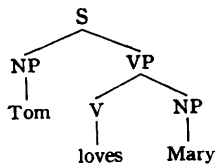
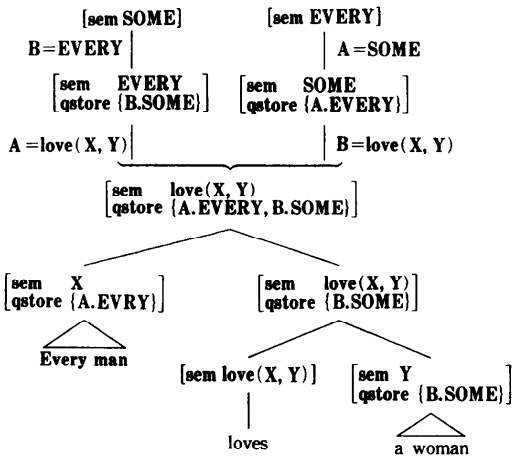


図-1 句 構 造

たとえば，*Every man loves a woman.* という文は量化子保管を用いて**図-2** のように分析される．説明の便宜のため，この図をボトムアップな処理過程として見てゆこう．まず，'every man' と 'a woman' の所で，量化された式 **EVERY** と **SOME** が導入される．これらの式の形は図の下端に示したとおりである．**EVERY** と **SOME** において，この段階ではその中で **A** と **B** の形が未定であることを示すために **A.EVERY** および **B.SOME** という形式で，それぞれ 'every man' および 'a woman' の範疇*の量化子保管（つまり，素性 **qstore** の値である集合）に収納される．構文木中の枝分かれ（branching）においては，娘範疇**の量化子保管の和集合が母親の量化子保管となると考えると，上から 3 段目の範疇ができる．枝分かれのない局所木（local tree）においては，娘の量化子保管の要素 $\alpha.\beta$ を取り出して α を娘の意味（素性 **sem** の値）として β を母親の意味とすることができるとしよう．すると，図の上部 2 段に示したとおり，ここでどの順序で要素を取り出すかに関して左右 2 とおりの場合がある．左側は，まず **A=love(X, Y)** として **A.EVERY** を



EVERY = $\forall X\{\mathbf{man}(X) \rightarrow\} A$
SOME = $\exists Y\{\mathbf{woman}(Y) \wedge B\}$

図-2 量化子保管による作用域の処理

* たとえば，*Who did you say that Mary loves* ε? において，*Who* と ε とは非有界依存の関係にある．共有界とは，単文の境界を越えることができるという意味である．

* 言語学では，非終端記号およびそのインスタンスである構文木中の節点のことを範疇（category）と言う．

** 木の枝分かれにおける子節点のことを娘（daughter），親節点のことを母親（mother）と言う．

量化子保管から取り出し，次に **B=EVERY** として **B.SOME** を取り出した場合で，'some' の作用域が 'every' のそれよりも広い解釈に当たる．つまり，**A=love(X, Y)** かつ **B=EVERY** とすれば，結局

SOME=∃**Y**{**woman**(**Y**)∧
∀**X**{**man**(**X**)→**love**(**X**, **Y**)}}

となる．右側は逆の場合である．量化子保管の要素をどういう場合にこのように取り出すかなどの詳細について，実際の定式化はこれとは異なるが，ここでの議論には特に関係しないので立ち入らない．

このように，量化子保管は計算の中間状態の表現である．そして，そのような中間状態に明示的に言及せずに，量化などの作用域を処理することは可能である．それには，作用域の大小関係を，量化された式の間の順序関係に関する制約としてそのまま処理すればよい．たとえば，'Every man loves a woman' の場合には，作用域の大小関係に関して図-3 のような制約が生ずる．図中の点線は，両端の二つの式のうちの下の式に 0 回以上の量化を施して上の式が得られるという制約を表わす☆．**LOVE** は文全体の意味を表わす式，**ManLove** は 'every man' によって量化された式，**LoveWoman** は 'a woman' によって量化された式である．なお，名詞に対する修飾句が量化子を含む場合などは話がやや複雑になる．詳細は Hasida[14] などを参照されたい．

この方法においては，量化子保管のような中間構造は，図-3 のような制約を処理する際に過渡的に生ずるに過ぎず，設計の段階では明示されない．これに対し，そのような中間構造に明示的に言及する理論を用いると，当然ながら記述も処理も格段に複雑になる．

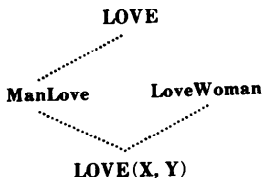


図-3 制約に基づく作用域の処理

☆ **LOVE** と **LoveWoman** の間にこの関係が仮定されていないのは，'Some people believe that every man loves a woman' のように，'a woman' の作用域が 'every man loves a woman' を越えることを可能にするためである．ちなみに，'a woman' のような不定名詞を量化子とみなさない立場もあるが，その場合も，ここでの議論は本質的にそのまま成立する．

4. 処理手続きの統合

言語処理過程において，常識や比喩や類推を含む推論は，非常に一般的なものとなるから，先に述べたように，理解と産出にわたって統合されて当然と考えられる．また，その一般性のゆえに人工知能一般の問題となり，本解説の範囲を越える．したがって，以下では特に，構文解析と文生成に関する処理手続きの統合について論ずる．

4.1 演繹としての構文解析と文生成

Earley 演繹[24]は，文脈自由言語の構文解析法である Earley のアルゴリズム[7]をホーン節論理における推論規則として一般化したものである．Earley のアルゴリズムは，チャート構文解析 (chart parsing)[19]の一種である．チャート構文解析では，チャート (chart) というデータ構造を用いる．チャートは，$A \rightarrow i\alpha j\beta$ という形の項目の集合である．この項目は，図-4 に示すように，入力文中の位置 i から j までの部分が，規則 $A \rightarrow \alpha\beta$ 中の非終端記号列 α として認識された，ということを意味する．ここで，英大文字は非終端記号，英小文字は入力文中の位置，ギリシャ文字は非終端記号の列を表わす．Earley のアルゴリズムは，図-5 に示す二つの推論規則として定義できる．入力文字列中の語 w の位置 i から j における生起がすべて $A \rightarrow iwj$ の形の項目として与えられている状態で Earley のアルゴリズムを実行した結果 $S \rightarrow k\alpha l$ という項目がチャートの中に生成されていれば，構文解析が成功したことになる．S は文の表わす非終端記号，k は文の先頭の位置，l は末尾の位置である．チャートの中で同一の項目を重複させなければ，この計算の空間計算量は $O(n^2)$，時間計算量は $O(n^3)$ である．n は入力文中の語の個数とする．

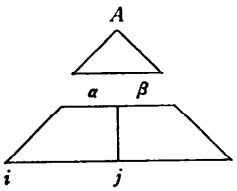


図-4 チャートの項目 $A \rightarrow i\alpha j\beta$ の意味

$$\frac{A \rightarrow iajB\beta \quad B \rightarrow \gamma}{B \rightarrow jj\gamma} \qquad \frac{A \rightarrow iajB\beta \quad B \rightarrow j\gamma k}{A \rightarrow iaBk\beta}$$

図-5 Earley のアルゴリズム

非終端記号をリテラルとみなし，文脈自由規則をホーン節とみなすことにより，図-5の規則をホーン節論理における推論規則に一般化したものが，Earley 演繹である．その場合，チャートの項目 $A \rightarrow i\alpha j\beta$ における α は処理済みのリテラルの列，β は未処理のリテラルの列となる．Earley 演繹を文脈自由文法に適用した結果は Earley のアルゴリズムと等価である．また，Earley 演繹においてホーン節の負リテラル（右辺のリテラル）の処理順序の設定を変えることにより，Earley のアルゴリズム以外のチャート構文解析も実現できる．

Earley 演繹は基本的にトップダウンの推論法であるが，ボトムアップの推論法を用いた構文解析法としては，Kanamori[18]のアレクサンダ・パーサ（Alexander Parser）がある．この方法では，文脈自由文法を論理プログラムに変換する際にさまざまな構文解析の戦略を組み込み，できたプログラムをボトムアップな仕方で解釈実行する．変換の際にトップダウンの戦略を組み込むこともできるので，トップダウンとボトムアップを組み合わせた構文解析が実現できる．

Shieber[27]は，文生成にも使えるように Earley 演繹をさらに一般化している．文生成の場合には，第1に，処理済みの句が文中のどこに現われるかが不定であることが多いので，チャートの項目 $A \rightarrow i\alpha jB\beta$ の i と j は変数である．第2に，計算の初期状態においては，入力である意味表現の各部分について，それを意味にもつ語 ω があれば，$A \rightarrow \omega$ という規則から作られる項目 $A \rightarrow i\omega j$ をチャートの要素とする．入力の意味表現と同じ意味表現をもつ文ができれば生成が成功したことになる．初期状態の与え方から，正しい文生成が行われるためには，おのおのの句の意味構造が空でなく，かつその句を含む句の意味構造の一部でなければならない（このような条件を満たす文法を意味的に単調（semantically monotonic）であると言う）．この点においてこの方法の一般性には問題がある．

Earley 演繹もアレクサンダ・パーサも，論理プログラムの解釈実行方式という一般的な形をしてはいるが，固定した実行順序や文法の変換方式の中に文脈自由言語という領域や構文解析または文生成という作業に依存する，強解法（strong methods）を組み込んでいる．したがって，構文解析と文生成の統合とか，意味的な推論などを含む一般的な認知過程との統合には不十分である．

4.2 主辞駆動

そこで，領域や作業によらない一般的な処理の戦略，つまり弱解法（weak methods）に関する研究を紹介しよう．「同じ情報を共有するもの同士は同時に処理せよ」というのは，一般的戦略として妥当であろう．文の句構造において，同じ情報を共有するものの組として典型的なのは，句とその主辞（または主要部：head）である．たとえば，「大きな車」の主辞は「車」であり，'Tom loves Mary' の主辞は 'loves' である☆．

Shieber[28]は，**意味主辞駆動生成**（semantic-head driven generation）という文生成のアルゴリズムを提案している．ある句の**意味主辞**（semantic head）とは，その句に含まれる句のうち，意味（図-2に即して言えば **sem** の値）を共有するものである．意味主辞駆動生成とは，ある意味表現をもつ句 P を生成しようとする際に，その意味主辞のうちで極小のもの（これをピボット（pivot）と言う）を先に生成し，そこから P に至る経路をボトムアップに探索しながら，その経路上に現われる他の句を再帰的に生成してゆく，というものである．

たとえば，**call_up(john, friends)** という意味をもつ文の生成は，**図-6** のように行われる．ここで，各局所木に付けたラベルは，その局所木が何番目に処理されるかを示す．まず最初に，根の範疇 **v**([ ]) : **call_up(john, friends)** のピボットとして，'call' の範疇を見付け，次にそこからボトムアップに構文木をたどって行く．**n** : **friends** や **p** : **up** のピボットはそれぞれ自分自身であるからこの場合は簡単だが，そうでなければ，ピボットの探索とボトムアップな構文木の構成という処理が再帰的に適用されることになる．この方法だと，意味的に単調でない文法でも扱えることに注意されたい．たとえば，'up' の意味 **up** が **call_up(john, friends)** の一部でないにもかかわらず，'up' は出力文に含まれる．これは，母親の意味構造に含まれない意味構造をもつ範疇が句構造規則の適用によって得られ，それに基づいてピボット（多くの場合は単語）の探索が起動されるためである．

☆ 文の主辞を動詞と考えない立場もある．

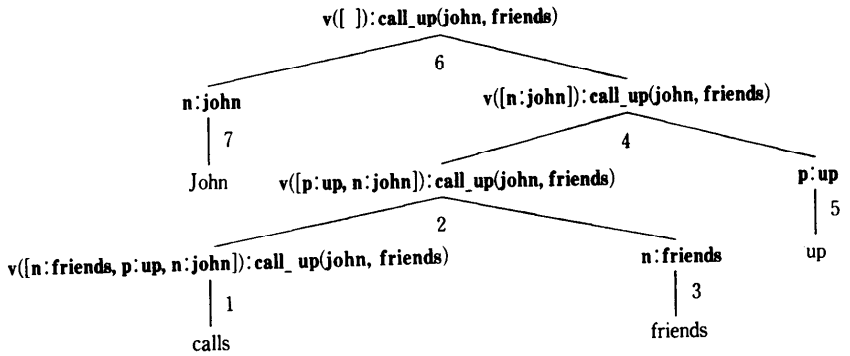


図-6 意味主辞駆動生成

van Noord[29]は，主辞駆動を構文解析にも応用し，それを連結でない構成素を許容する文法に応用している．主辞駆動という考え方のもとである「同じ情報を共有するもの同士は同時に処理せよ」という一般的な戦略をほぼそのまま用いることによって主辞駆動生成に近い処理が自動的に行われることを後に示す．

4.3 一般的ヒューリスティック

これまで述べてきたさまざまな方法は，構文解析と文生成を統合する方法としては一般性に欠けるものであった．非常に一般的な弱解法を用いて構文解析と文生成とを統合しようとする研究としては，Hobbs ら[16]，Zajac[30]，伝ら[6]，Hasida[13]などの仕事がある．これらのうち，伝らと Hasida は，従来提案されてきた専用のアルゴリズムによる効率のよい処理を創発的属性 (emergent property) としてもつようなメカニズムを探求しており，特に Hasida の方法は，Earley のアルゴリズムや主辞駆動生成に匹敵する処理効率をもつ．ここでは，Hasida[13]を例題に即して解説しよう．

一般に，構文解析は，ホーン節プログラムとして表現される制約を満たす変数の束縛を求める問題として定式化できる．この性質を利用して Prolog によって構文解析を直接的に定式化したものが，標準的な DCG[23]である*，一方，文生成ではさらに，与えられた意味表現をちょうど覆うような意味表現をもつ文を生成することが要請されるが，この要請は，ホーン節プログラムの制約の下での変数の束縛を求める問題としては直接的に表現できない．以下に示す方法は，これら二つの側面を共に扱うものである．

まず，構文解析を考える．

* 標準的でない DCG としては，BUP[21]などがある．

$P \rightarrow a$

$P \rightarrow PP$

という文法の下での構文解析は，次のような制約によって定式化される．

:-**p**($\mathbf{A}_0$, **B**), $\mathbf{A}_0$=[**a**|$\mathbf{A}_1$], $\mathbf{A}_1$=[**a**|$\mathbf{A}_2$], ….

(**Φ**)**p**([**a**|**X**], **X**).

(**Ψ**)**p**(**X**, **Z**) :-**p**(**X**, **Y**), **p**(**Y**, **Z**).

ここで，**p**(α, β) において α と β の組は差分リストであり，**p**(α, β) は，α に始まり β で終わるリスト（文字列）がこの文法の下での文であることを意味し，節 **Φ** と **Ψ** はそれぞれ文脈自由規則 $P \rightarrow a$ と $P \rightarrow PP$ を表わす．この制約は，**図-7** のようなネットワークとみなすことができる．ここで，各閉曲線で囲まれた領域は節を表わし，• は項（変項）を表わし，項の間の曲線は等式を表わす．節の中の太い矢印は，正リテラルとして参照されている要素式を指す．要素式の間の曲線は**単一化結線** (unification link) と呼ばれ，両端の要素式が単一化可能であることを示す．ここで要素式は束縛も含む．単一化可能な任意の二つの要素式は単一化結線で結ばれているとする．ただし，以下では簡単のため，原則として定義された述語を

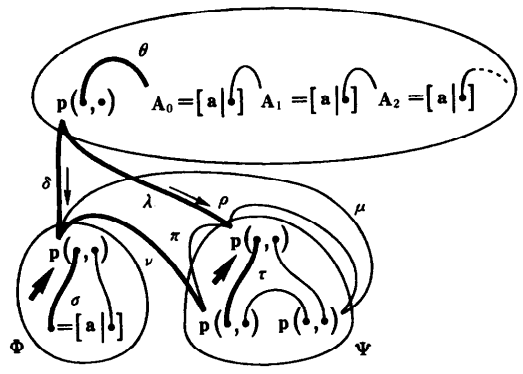


図-7 構文解析（1）

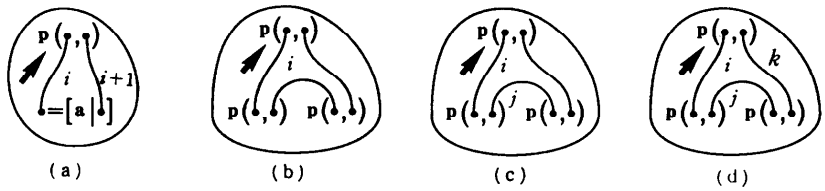


図-8 構文解析によって作られる節

もつ要素式の間の単一化結線のみを明示する．n 項述語をもつ要素式の間の単一化結線は，対応する項の間の n 本の等式を含む．$\bullet=[\mathbf{a}|\bullet]$ は $\bullet=[\alpha|\bullet]$ と $\alpha=\mathbf{a}$ とを合わせたものの略記である．

以下で述べる処理によって作られる節を図-8に示す．(a)の型の節は，節 Φ のインスタンスであり，他は Ψ のインスタンスである．等式のラベル l は，その等式が $\mathbf{A}_l$ によって**包摂** (subsume) されることを示す．項または要素式 α が項または要素式 β を包摂するとは，α の基礎インスタンスの集合が β の基礎インスタンスの集合を含むことであり，項または要素式 α が等式または単一化結線 δ を包摂するとは，α が β の両端の項または要素式を包摂することである．(a)および(b)では $0\leq i<n$，(c)では $0\leq i<j<n$，(d)では $0\leq i<j<k<n$ が成り立つ．詳しく論ずるゆとりはないが，これらの節のおのおのを作るのに要する処理の計算量は定数であるから，この構文解析の計算量は時間・空間ともに $O(n^3)$ である．また，ここでは簡単のため節を単位としてネットワークの複写（基礎インスタンスの集合の分割）を行っているが，複写の単位を要素式とすれば，実は一般の文脈自由文法に対して時間計算量 $O(n^3)$，空間計算量 $O(n^2)$ で構文解析ができる．

以下，図-8に示した形の節が作られる様子を示す．構文解析のように，ホーン節プログラムとして与えられる制約を満たす変数の束縛を求めることに帰着できる問題の場合には，処理はもっぱら**依存関係** (dependency) によって起動される．二つの項の間に依存関係があるとは，それらが共に束縛されており，かつ，**強い等式** (strong equations) からなる，巡回路 (cycle) を含まない経路（**依存経路**：dependency path）によって結ばれているということである．強い等式とは，原則として，節内の等式であるか，または，定義された述語をもつ要素式の間の単一化結線に含まれる等式である．図-7においては，$\mathbf{A}_0$ と Φ の $\bullet=[\mathbf{a}|\bullet]$ の第1項の間に，太線で示された2本の依存経路 $\theta\delta_1\sigma$ と $\theta\lambda_1\tau\nu_1\sigma$ がある．ここで，単一化結線 ξ の両端の要素式の第 i 項同士を結ぶ等式を ξ_i で表わす．依存関係において，依存経路の両端の項に対する束縛が単一化可能なものであれば，**包摂化** (subsumption) と呼ばれる演算を依存経路に沿って行い，節を適当に分割してゆくことにより，一方の項に他方の項のインスタンスを包摂させる．この処理は次のヒューリスティックに従う．

(H1) 依存経路の端点のうち一方だけが先頭節にあれば，その依存経路上の包摂化はその端点から他方の端点へ向かって進む．

また，詳細は省略するが，束縛に関する矛盾が生じた場合は節を適当に消去する．依存経路の一方の端点 ω に関するその依存経路上の等式 ξ に沿った包摂化とは，ω が ξ を包摂するように ξ の端点の一方を含む節を分割することである．この場合，ξ のもう一方の端点はすでに ω に包摂されていなければならない．今考えている例題では，全ての依存関係は $\mathbf{A}_i$ と Φ（のインスタンス）の中の $\bullet=[\mathbf{a}|\bullet]$ の第1項の間に生じ，(H1) によって $\mathbf{A}_i$ から包摂化が始まる．

図-7においては，細い矢印で示したように，δ と λ に沿って2通りの包摂化が生じ，Φ は図-9の Φ_1 と Φ に，Ψ は Ψ_1 と Ψ に分割される．また，図-7の単一化結線 μ は図-9の中の μ' と μ'' に分割される．ν，π および ρ も同様に分割される．Φ_1 の左側の等式は $\mathbf{A}_0$ に包摂されているので，Ψ_1 は図-8の(b)で $i=0$ の場合である．

次に，図-9に示された二つの包摂化が生ずる．ν' に沿った包摂化が作るべき節は，もとの Φ のインスタンスであって，$\mathbf{p}(\bullet,\bullet)$ の第1項が $\mathbf{A}_0$ に包摂されているような節である．ところが，Φ_1 はそのような節だから，ここで新しい節を作る必要はなく，ν' を図-10のように付け換えればよい．π' に関しても同様である．一方，$\mathbf{A}_0=[\mathbf{a}|\bullet]$

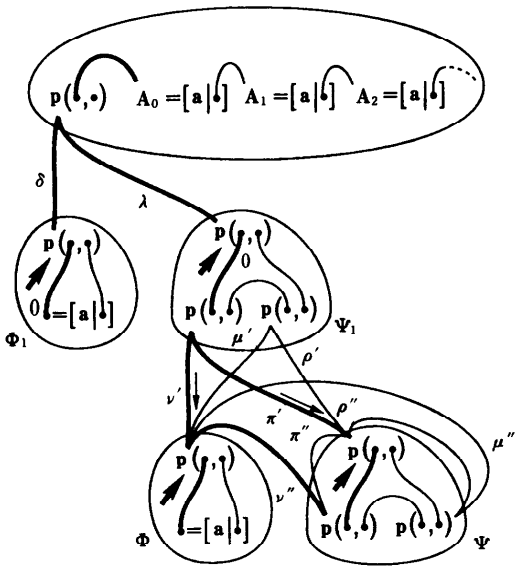


図-9 構文解析（2）

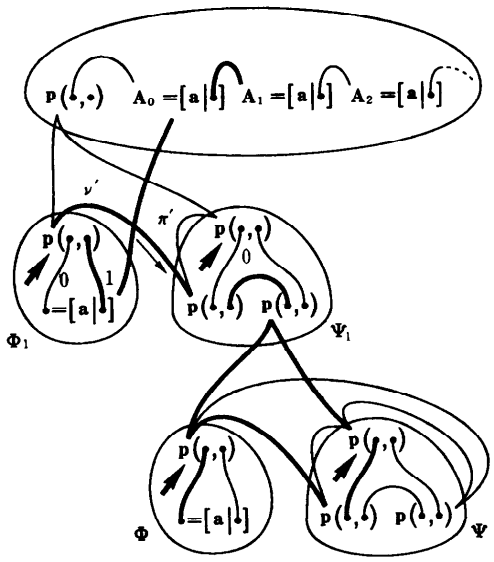


図-10 構文解析（3）

は Φ_1 の •=[a| •] を包摂している．このような場合，両者の間の単一化結線に含まれる等式は強い等式になると考える．これによって Φ_1 が図-8 の(a)の形の節となる一方，A_1 と Φ の第1項の間に太線で示されるような依存関係が生ずる．そこで同様に包摂化を進めると，Ψ_1 は Ψ_1 と Ψ_2 に分割され，図-11 のような状態に至る．Φ_2 は(c)の形の節である．詳細は省略するが，さらに処理を進めることにより，Ψ_2 のインスタンス

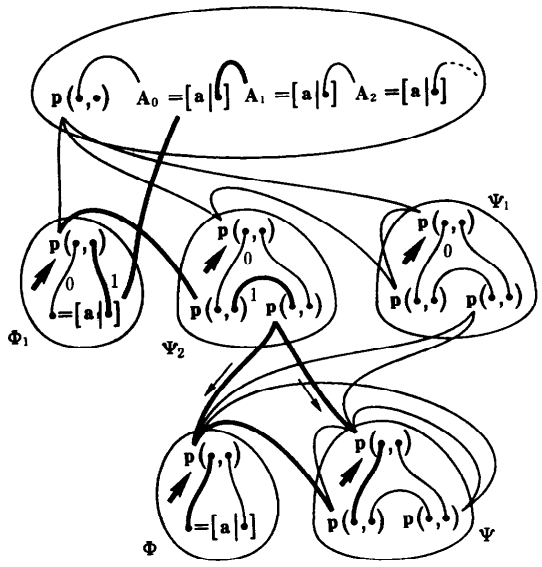


図-11 構文解析（4）

として(d)の形の節が作られる．一般に，Ψ のインスタンスとして(b)ができ，そのインスタンスとして(c)，さらにそのインスタンスとして(d)ができる．

ここで，作業や領域に依存しない一般的なヒューリスティック（H1）によって処理が制御されていることに注意されたい．これに対し，先に述べた Earley 演繹に基づく方法によって Earley のアルゴリズムを実現するには，節の本体のリテラルに関する左から右へという処理の順序をあらかじめ固定する必要がある．また，Earley 演繹ではこのように節の中の処理の順序が定まっているため，その順序を動的に変えなければ効率的に処理できないような問題には向かない．たとえば，文脈自由文法の下での構文解析では処理の順序を左から右へと固定してもよいが，内部構造をもつ非終端記号に関する処理では，必ずしも，固定した順序に従って左から右へと処理を進めればよいわけではなく，Earley 演繹では効率が悪い．アレクサンダ・パーサなどの場合も同様である．

次に，生成の例をみよう．以下の節を含むプログラムによって，'Kim runs' という英語の文の生成が行われる．

(A) **s(RUN, W0, W1) :-run(RUN)$,**
agt(RUN, kim)$, ….

(B) **s(SEM, X, Z) :-np(SBJSEM, X, Y),**
vp(SEM, SBJSEM, Y, Z).

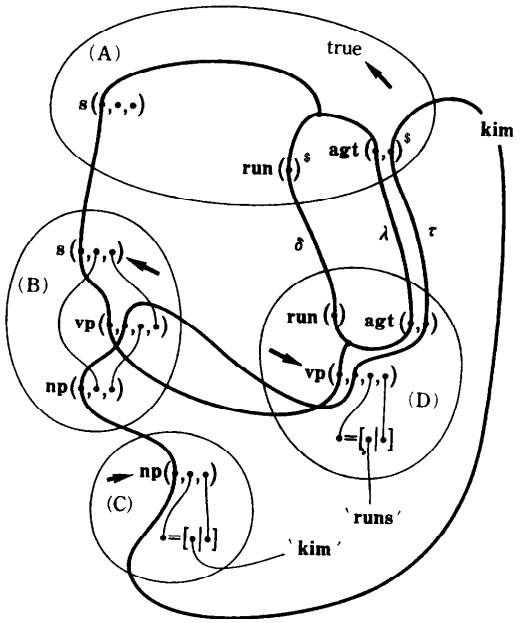


図-12 文 生 成

(C)　**np(kim, X, Y)** :-**X**=[**'kim'** | **Y**].

(D)　**vp(R, AGT, X, Y)** :-**X**=[**'runs'** | **Y**],
run(R), **agt(R, AGT)**.

(A)の **RUN** は「Kim が走る」という事象を表わし，(A)の中の二つの '$' 付き要素式を合わせたものは，そういう事象が存在することを表わす．図-12 にこれらの節を図示する．**kim** や **'kim'** などの定項への束縛は，おのおのの定項ごとに一つしかなく，節には含まれないと考える．また，定項に束縛された項はその定項と同一視する．上記の構文解析の例に現われたような依存関係はここには存在しないから，項の間の包摂化は生じない，ということに注意されたい．

要素式の右肩の '$' は**コスト** (cost) を表わす．これは，Hobbs ら[17]の仮説コストと同様，要素式がそのままでは仮説し難いことを表わす．より正確には，ある要素式がコストをもつということは，その要素式のすべてのインスタンスが仮説し難いということである．逆に言えば，要素式がコストなしのインスタンスをもてば，その要素式はコストをもたない．したがって，要素式 α と包摂関係にある別の要素式がコストをもたなければ，α はコストをもたない．そこで，コスト付きの要素式とコストなしの要素式との間に包摂の関係が成り立てばそのコストは消去されると考える．

要素式の間の包摂関係を作るには，要素式の間の包摂化を行う．要素式 α から β への包摂化とは，β のインスタンスで α に包摂されるものを作ることである．もちろんこのとき，β を含む節がコピーされる．要素式の間の包摂化は次のヒューリスティックによって制御されるとする．

(H2)　二つの要素式の間の包摂化は，それらの対応する項が依存経路によって結ばれているとき，特に，そのような項の組の割合が大きいほど，優先度が高い．

このヒューリスティックは，「対応する項同士が等しい要素式は等しい」という，等号に関する公理に基づく．また，項の間の依存関係がある場合は，(H2)において要素式が束縛であって第1項同士が依存経路で結ばれている場合に相当するが，そのときは，要素式の間の包摂化ではなく，先に述べたような項の間の包摂化を行う．

さて，図-12 において，太線のうち δ，λ および τ だけが弱い（強くない）等式であり，他の太線はこれらの等式の端点を結ぶ依存経路を示す．二つの **run**(•) の間の包摂化により δ は強い等式になり，二つの **agt**(•, •) の間の包摂化により λ と τ も強い等式になるとする☆．これらの包摂化は，(A)の **s**(•, •, •) の第1項をとおる依存結線が引き起こしたものであり，この項は生成すべき文の意味内容であるから，この処理は，意味主辞駆動生成におけるピボットの同定に相当する．ここで，τ が強い等式になったことにより，τ を含む太線の巡回路は **kim** とそれ自身を結ぶ依存経路となる．そこで，(H1)に従って **kim** から左回りに項の間の包摂化を行うと，生成すべき文の構造に含まれる(B)と(C)のインスタンスができる．この処理は，意味主辞駆動生成における主辞以外の生成にほぼ相当する．ここで，**kim** が定項でなくてコストをもつ意味表現である場合は，再帰的に意味主辞の探索が行われる．

5. 結　論

辞書や文法を含む知識も処理手続きも理解と産出で共有する，という強い意味での統合が，柔軟な自然言語処理モデルの設計に必要であることを

☆ 実は，前者の包摂化が先に行われて (D) がコピーされることにより λ と τ もコピーされ，次の包摂化は (D) のコピーのうち δ が繋がったほうに含まれる agt(•, •) に関して行われると考えられるが，詳細は省略する．

指摘し，言語理論におけるその意義を例によって述べた上で，その統合への第1段階として，構文解析と文生成を同一の知識と処理手続きの下に統合する研究について紹介した．特に，文脈自由文法の構文解析と意味構造からの表層文の生成に関しては，従来用いられていた効率のよい強解法による処理過程が，一般的な弱解法から自然に生ずることを示した．自然言語処理において用いられている強解法としては，他にとりたてて工夫に満ちた効率のよいものはないから，特別な強解法は自然言語処理において不要だと言って差し支えないだろう．

実際，統合ということを考えた場合，自然言語処理固有の問題として扱えるのは構文解析と文生成までであり，そこから先はまったく一般的な認知過程だろう．それは，構文解析や文生成よりもはるかに自由度の高い処理過程であり，その設計には，2.の最後で触れたような制御のメカニズムが必要と考えられる．しかし少なくとも，本稿で解説した構文解析と文生成に共通の処理の戦略は，それ自身領域や作業に依存しない普遍的なものであるから，一般的な推論システムの中に自然に埋め込むことができるはずである．

謝辞 本稿を準備するにあたり，SONY CSLの長尾確氏と査読者から有益なコメントをいただいた．ここに記して感謝する．

参考文献

1) Bresnan, J. W., editor: *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Massachusetts (1982).
2) Chomsky, N.: *Aspects of the Theory of Syntax*, MIT Press, Cambridge (1965).
3) Chomsky, N.: *Lectures on Government and Binding*, Foris, Dordrecht (1980).
4) Chomsky, N. (内田 平訳): 派生と表示の経済性に関する覚書，認知科学の発展，Vol. 2, pp. 11-56 (1990).
5) Cooper, R.: *Quantification and Syntactic Theory*, volume 21 of *Synthese Language Library*, Reidel, Dordrecht (1983).
6) 伝 康晴：認知的制約プログラミングと統合的自然言語処理—統語解析と統語生成の統合について—，コンピュータソフトウェア，Vol. 8, No. 6, pp. 38-50 (1991).
7) Earley, J.: An Efficient Context-Free Parsing Algorithm, *Communication of ACM*, Vol. 13, pp. 94-102 (1970).
8) Fong, S.: Principle-Based Parsing and Type Inference, In *Proceeding of the 3rd Workshop on Natural Language Understanding and Logic Programming*, pp. 38-56 (1991).
9) Gazdar, G., Klein, E., Pullum, G. K. and Sag, I. A.: *Generalized Phrase Structure Grammar*, Basil Blackwell, Oxford (1985).
10) Gunji, T.: *Japanese Phrase Structure Grammar*, Reidel (1987).
11) 橋田浩一：制約と言語，コンピュータソフトウェア，Vol. 6, No. 4, pp. 16-29 (1989).
12) 橋田浩一，竹沢寿幸：自然言語処理における統合の諸相，コンピュータソフトウェア，Vol. 8, No. 6, pp. 511-524 (1991).
13) Hasida, K.: Common Heuristics for Parsing, Generation, and Whatever…, In *Proceedings of the Workshop on Reversible Grammar in Natural Language Processing*, pp. 81-90, Berkeley, 1991, held in connection with 29 th Annual Meeting of the Association for Computational Linguistics.
14) Hasida, K.: Reducing Complexity of Constraint-Based Grammars, In Barwise, J., Gawron, J. M., Plotkin, G. and Tutiya, S., editors: *Situation Theory and Its Applications, volume II*, pp. 405-423, CSLI Publications (1991). CSLI Lecture Notes Number 21.
15) Hasida, K.: Dynamics of Symbol Systems: An Integrated Architecture of Cognition, In *Proceedings of FGCS '92*, Tokyo (1992).
16) Hobbs, J. and Kameyama, M.: Translation by Abduction, In *Proceedings of the 13th International Conference on Computational Linguistics, Volume 3*, pp. 155-161 (1990).
17) Hobbs, J., Stickel, M., Appelt, D. and Martin, P.: Interpretation as Abduction, Technical Note 499, SRI International (1990).
18) Kanamori, T.: Alexander Parser, Technical Report 548, Institute for New Generation Computer Technology (1989).
19) Kay, M.: Algorithm Schemata and Data Structures in Syntactic Processing Technical report, XEROX Palo Alto Research Center, Palo Alto, California (1980).
20) Kay, M.: Parsing in Functional Unification Grammar, In Dowty, D., Karttunen, L. and Zwicky, A. M., editors, *Natural Language Parsing : Psychological, Computational, and Theoretical Perspectives*, Cambridge University Press (1985).
21) Matsumoto, Y., Tanaka, H., Hirakawa, H. et al.: Bup: A Bottom-Up Parser Embedded in Prolog, *New Generation Computing*, Vol. 1, pp. 145-158 (1983).
22) McDonald, D. D.: Reversible NLP by Deriving the Grammars from the Knowledge Base, In *Proceedings of the Workshop on Reversible Grammar in Natural Language Processing*, Berkeley (1991).

23) Pereira, F. C. N. and Warren, D. H. D.: Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence*, Vol. 13, pp. 231-278 (1980).
24) Pereira, F. C. N. and Warren, D. H. D.: Parsing as Deduction, In *Proceedings of the 21st Annual Meeting of ACL*, pp. 137-144 (1983).
25) Pollard, C. and Sag, L. A.: *Information-Based Syntax and Semantics, Volume 1*, CSL Lecture Notes Number 13, CSLI Publications, Stanford (1987).
26) Sgall, P. and Panevovă, J.: Dependency Syntax —A Challenge, *Theoretical Linguistics*, Vol. 15, pp. 73-86 (1989).
27) Shieber, S. M.: A Uniform Architecture for Parsing and Generation, In *Proceedings of the 12th International Conference on Computational Linguistics*, pp. 614-619 (1988).
28) Shieber, S. M., van Noord, G. and Moore, R. C.: A Semanitic-Head-Driven Generation Algorithm for Unification-Based Formalisms, In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pp. 7-71 (1989).
29) van Noord, G.: Head Corner Parsing for Discontinuous Constituency, In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pp. 114-121 (1991).
30) Rémi Zajac: A Uniform Architecture for Parsing, Generation and Transfer, In *Proceedings of the Workshop on Reversible Grammar in Natural Language Processing*, Berkeley (1991).

(平成4年4月13日受付)

橋田 浩一 (正会員)

1958年生. 1981年東京大学理学部情報科卒業. 1986年同大学院博士課程修了. 理学博士. 同年電子技術総合研究所入所. 1988年から1992年まで(財)新世代コンピュータ技術開発機構に出向. 現在, 電子技術総合研究所自然言語研究室. 自然言語処理, 認知言語学, 人工知能の研究に従事. 日本認知科学会, 日本ソフトウェア科学会, 人工知能学会, ACL, AIUEO 各会員.