計算的手法を用いた構文習得研究の可能性

長谷部　陽一郎

1. はじめに

言語研究において構文（construction）の重要性が説かれるようになって久しい。構文文法（Construction Grammar）にとっての1つの重要な出来事は、Adele Goldberg による1995年の *Constructions* の出版であり、彼女の理論は特に認知言語学の領域で広く受け入れられ、発展してきた（Goldberg 1995, 2006）。また、より古くから存在し、同様に発展を続けてきたのが、Charles Fillmore や Paul Kay らによるいわゆる Berkeley Construction Grammar である（Fillmore *et al.* 1988; Kay and Fillmore 1999）。後者は、主辞駆動句構造文法（HPSG）など、より形式的な言語理論に基づいた分析を行ってきた研究者をもまきこんで、1つの言語研究フレームワークとしての存在感を増している（Sag *et al*. 2003; Fillmore *et al.* 2009）。これらの主要な理論の他、近年ではいくつもの関連した理論や枠組みが存在している（cf. Östman and Fried 2008）。

これらの理論を「構文文法」として1つのカテゴリーにまとめているもの、すなわち、それらすべてに共有されているものは、「記号主義（symbolic thesis）」と呼ばれる言語観である。言語の文法を構成する要素には、形態素や語から句や文に至るまで様々なレベルのものが存在するが、いずれのレベルにおいても、形式と意味とが密接に結びついている。これら形式と意味との関係を記号的関係（symbolic relation）と呼ぶならば、およそあらゆる言語表現は幾重にも重なった記号的関係に基づいて解釈されなければならない。また、これらの記号的関係はレベル間で合成関係を持ちながらも、同時に独立した存在であり、それゆえに、例えば *kick the bucket* のような慣用句は、*kick*, *the*, *bucket* という個々の言語的要素が持つ性質と概念構造から成り立っ

『言語文化』12-2：395－420ページ　2009.

ている一方で、それ自体に固有の意味や機能を有することになる。ここで、このような慣用句を「構文」という名で呼ぶことについては学界において必ずしも統一された見解があるわけではない。しかし、Goldberg を含む複数の研究者が、一般的な意味での「構文」とは一種のプロトタイプに過ぎず、実際には構文があらゆる言語的レベルにおいて存在することを―つまり語や句も構文と呼べることを―認めている（cf. Goldberg 2006; Östman and Fried 2008）。したがって、構文とは基本的に、1 個以上の構文から構成された「再帰的な（recursive）」構造体であると言える。[1]

このような言語観に基づく構文文法が応用され、一定の成果を挙げているのが言語習得の領域である。Noam Chomsky らによる生成文法の枠組みにおいて強調されてきたように、子供は限られたデータと限られた時間の中で、母語話者としての能力を確実に身につけていく。そこに個々の話者の（無）意識下で働くきわめて効率的・効果的なメカニズムが存在していることは疑いない。生得的言語論者が仮定するそのメカニズムとは、言うまでもなく、普遍文法（Universal Grammar）および生得的言語習得装置（Language Acquisition Device）である。対照的に、構文文法家の多くは文法の生得性を仮定しない。なぜなら形式と意味との記号的関係という前提のもとでは、言語習得のただ中にある子供の言語知識が、大人のそれと同様に構造化されているとは限らないからである。例えば、*gimme-some-milk* という表現は、幼い子供の中で $V\text{-}NP_1\text{-}NP_2$ という統語形式のもとに表現として蓄えられている必要はない。それは一定の機能を持った 1 個の独立した記号である。このように、より大きな単位で形式と意味機能とをマッチングさせていくことからはじめ、次第にそれらの中に一定のパターンを見出すとともに下位の記号に対する理解を得るというのが、構文文法が想定する言語習得の道筋である（Tomasello 2005, 2008）。もしこのようなパターンの発見が、発達段階にある子供の認知能力のもとに可能なものであり、また、事実上無限の文を生成できるメカニズムへと限られた時間で発展可能であるなら、普遍文法や言語習得装置の必要性は薄れてくる。

では、子供の言語習得過程において働いていると考えられるパターン発見のプロセスとはどのようなものなのか。間違いなく現在の構文文法研究にお

ける究極的な目標の1つは、この問いに対する答を得ることである。事実、近年では子供の発話データベース CHILDES など、実際の発話に基づいた言語習得研究を行う基盤が整ってきたこともあり、言語習得の観点からの構文研究がますます盛んになっている（cf. Goldberg 2006; Clark and Kelly 2006）。[2] しかし、未だ手つかずの部分も少なくない。まず、構文がネットワーク構造を持つことはよく知られているが、このネットワーク構造が話者の言語知識の一部として実際にどのように構築されるのかについては共通理解が得られていない。子供が周囲の大人たちの発話に触れる中で言語知識のネットワークを拡張・更新していく過程を、擬似的にでも再現する手法が求められる。また、ある構文を（単なる決まり文句としてではなく）完全な形で習得することは、下位パターンの重ね合わせを伴うものであるが、あるパターンを習得するにあたって、どのような下位パターンの習得が必要であり、そのために子供にどのようなデータが与えられているかということも重要な問題である。これらに答えるための定式化された手法は、確立されていると言えない。

以上のような背景のもとに、本稿では、実際の発話事例と計算的な手法を用いて子供の構文習得プロセスを推測するための2つの分析手法を提案する。第一の手法は形式概念分析（Formal Concept Analysis）と呼ばれる数学的手法を言語分析に応用したものである（Ganter and Wille 1999; Ganter *et al*. 2005）。これにより、与えられた言語データに潜む構文ネットワーク構造の可能な姿を推測することができる。第二の手法はパターン・ラティス（Pattern Lattice）分析と呼ばれ、黒田・長谷部（2009）および Kuroda（to appear）で示されたモデルがもとになっている。後者は細部において未だ改良の余地はあるが、構文の「重ね合わせ問題」に取り組むにあたって、有用なツールとなる可能性がある。

これらの手法は、一定のアルゴリズムに基づいて計算機に実装することが可能である点で、構文文法における一般的な分析手法とは異なっている。そのため、一部の研究者にとっては、一見、構文文法の（あるいはそれを包摂するとされる認知言語学の）理念に沿わないものと映る可能性がある。もちろんこれは正しい見方でない。見過ごされがちな事実であるが、構文文法は当初よりある種の計算的手法と密接な関係にある。そこで、次節では本稿で

提案する手法の意義を理解するための基盤として、構文文法の計算的特徴について論じる。その後、3節と4節でそれぞれ形式概念分析とパターン・ラティス分析を用いた構文パターンの発見・抽出手法を示し、5節で全体のまとめを行う。

2. 構文の計算的特徴

構文文法は認知言語学の一領域として見なされることが多く、そのため、計算的なシステムとは相容れないものであるという印象を持たれることが少なくない。[3]　これは主に、計算的なプロセスを想定することは、言語の本質を数式のようなものに置き換える「悪しき還元主義」につながるという発想によるものである。ところが実際には構文文法は非常に計算的な性質を持っている。ここで構文文法が「計算的」であるというのは、記述対象としての言語そのものが計算に基づいているという意味では必ずしもない。むしろ、言語の構造を記述する「方法」としての構文文法が計算可能という意味である。したがって、構文文法が計算的な性質を持つことは、この理論が還元主義的であることを意味しない。計算的な性質を持つことは、理論を客観的な評価・判断の対象とし、また計算機上でのシミュレーションを実現可能にする。それゆえ構文文法が計算的な性質を持つことは、メリットでありデメリットではない。以下では構文文法が持つそのような性質について、「オブジェクト指向的」であること、および「関数的」であることという2つの側面に着目し、論じていく。

2.1　オブジェクトとしての構文

Goldberg（1995）は構文文法がオブジェクト指向に基づいた継承リンク（Inheritance Link）のシステムを備えていることについて、次のように述べている。

> 一般化を捉える手法の1つとして継承を用いるという考えは、本来コンピュータ科学で生まれたもので、データ構造をできる限り一般化した形式で表示する方法として用いられた。（中略）低レベルのものが

> 高レベルのものから情報を継承するという抽象的継承関係を仮定することで、情報は効率よく蓄えられ、修正も容易になる（Goldberg 1995: 72, 河上他訳）

オブジェクト指向は、問題領域に存在する様々な事物を「オブジェクト」として規定し、プログラムが提供する様々な機能を、これらオブジェクト同士の相互作用として実装する計算機プログラミングのパラダイムである。より古いパラダイムとしての手続き型プログラミングに対して、オブジェクト指向は、「継承」、「カプセル化」、「多態性」という概念を備えている点で特徴づけられる（cf. Bergen and Chang 2003; Hasebe 2005）。これらのうち、第一の継承（inheritance）については Goldberg が述べている通りである。ある抽象的な機能を備えたオブジェクトを用意し、それを継承したオブジェクトを作成することで、より機能的に豊富な、あるいは一部の機能が異なったオブジェクトを最低限の作業量で実現することができる。第二のカプセル化（encapsulation）とは内部の情報を隠蔽する機能であり、これにより、オブジェクトは、互いの内部的な詳細に立ち入ることなく、一定の手続きを踏むことでメッセージのやり取りを行うことが可能になる。またカプセル化は、不用意な外部からの改変を防ぎ、システムの変更時にも、影響を最小限に抑えることができる。第三の多態性（polymorphism）とは、ある機能を求めているオブジェクトに対し、条件に応じて、自動的に適切な形で―すなわち多態的に―応答する性質を言う。これにより、メッセージを送信する側のオブジェクトは、すべての場合について個別に対応するのではなく、最低限のプロトコルだけを効率的に保持することができる。このようなオブジェクト指向の考え方は、70年代から80年代前半にかけて理論的な発展をみた後、80年代後半から90年代にかけて実際の計算機プログラム開発の現場で浸透していった（Meyer 1997）。

Goldberg がオブジェクト指向と構文との関係について明示的に述べるのは基本的に継承のメカニズムについてのみであるが、カプセル化と多態性についても両者のアナロジー的関係は当てはまる。例えば1節で言及した「死ぬ」という意味を持つ英語の慣用句 *kick the bucket* は、構文／オブジェクトとし

てカプセル化されているため、もはや内部要素である *kick* や *bucket* が持つ意味的内容は問われない。つまり、*the bucket* を *kick* することと「死」との因果的な（あるいはその他の）関係は事実上隠蔽されているのである。また、この表現の使用者もなぜそれが「死ぬ」という意味に結びつくかを知識として持っている必要はない。

さらに、*kick the bucket* が1個の動詞表現として、使用の（統語的）文脈に応じた多態的な振る舞いが可能であることも注目に値する。通常の動詞とは異なった形態的特徴を持つ表現ながら、*kick the bucket* は通常の動詞と同じプロトコルによって扱うことが可能である。つまり、歴史的な由来に起因する内部構造に関わらず、使用者はこれを「動詞」として扱うことができる。例えば *kicked the bucket* と過去形にすることも、*have not kicked the bucket* のように否定完了の形で用いることもできる。これは構文の持つ一種の多態性を表していると言える。

では、なぜこのように、計算機プログラミング言語と自然言語という異なる領域において、同様の仕組みが見い出されるのだろうか。その重要な要因は、心理的な経済性・効率性である。計算機プログラミングにおいては、継承、カプセル化、多態性という仕組みを利用することによって、開発者にとっては複雑なシステム全体の見通しと保守性が向上するという利点があり、プログラムの使用者にとっては自然で簡便なインターフェイスが提供されるという利点がある。ここで、1つのアナロジーとして、言語の習得過程にある子供をプログラムの開発者かつ使用者であるような存在として見なすならば、自然言語においても同様のことが言える。子供は、（一見）複雑を通り過ぎて雑多とさえ感じられる入力データに必要な秩序を与え、これを情報として整理し、所属する共同体において定められたプロトコルに沿った形で言語化する術を身につける。このプロセスには、データからパターンを発見してこれを構造化という開発者的側面と、さらにそれをユーザーとして主体的に利用するというユーザー的側面が存在している。2つの役割をこなしつつ、第一言語習得という重大なタスクを限られた時間と物理的資源のもとでやり遂げるためには、シンプルかつ強力な方策が必要となる。計算機科学の領域では、オブジェクト指向の考え方によって計算機プログラムの開発スピード

と品質が向上した。継承、カプセル化、多態性といった概念が開発者やユーザーの心理的な経済性・効率性を大いに高めたからである。同様の仕組みが言語習得の過程にも存在していると仮定することは理に適う。あるいは別の見方をするならば、むしろ、私たちが現実世界の中で（無）意識下で利用してきたノウハウを再発見して定義したのが計算機科学におけるオブジェクト指向であるとも言えるだろう。いずれにしても、両者の間には偶然ではない共通性があり、このことは構文文法が（少なくともある種の）計算的な方法論と相容れないものではないことを意味している。

2.2 関数としての構文

次に、構文文法が関数的であることについて論じる。ここでは、例として二重目的語構文（V-O_1-O_2 構文）を用いて考えてみたい。Goldberg（1995: 141-142）では (1) のような例を用いて、この構文の着目すべき性質について述べている。二重目的語構文は、最も典型的な場合として (1a) のように *give* などの動詞と融合する他、(1b) の *bake* のように、一見まったく異なった種類の動詞と共起することが可能である。

(1) (a) *Sally gave her sister a cake.*
(b) *Sally baked her sister a cake.*

bake という動詞自体には「（ケーキなどを）焼く」という意味があるに過ぎない。それにも関わらず、(1b) の文は全体として、*Sally* が *her sister* に *a cake* を「焼いてあげる」という意味を含んでいる。つまり、文字通りの「焼く」という意味の他に、間接目的語から直接目的語へと対象物が移動するという内容が新たに加わっている。Goldberg（や彼女に続く構文文法の研究者たち）は、これを、英語の二重目的語構文が持つ意味機能の働きによるものであると考え、その構造を図1のように規定する。

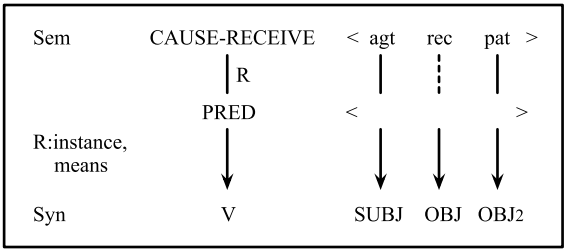

図1 二重目的語構文の構造

ここでは、二重目的語構文の意味（Sem）のコアとなるイベントスキーマがCAUSE-RECEIVEという関数で表現されており、この関数が引数（argument）を3つ取ることが示されている。すなわち動作主（agent）、受容者（recipient）、非動作主（patient）である。また、これらとは別にこの関数は内部に変項を持っており、その値は具体的な動詞タイプを表すPREDによって決定される。また、PREDの性質についての規定を行うのがRであり、ここでRがinstance, meansとなっているのは、PREDがイベントスキーマを具体化し、その行為の方法を述べるようなものであることを要請している。このように抽象的なイベントスキーマと具体的な意味内容を持った動詞タイプが融合することで、全体の意味内容はより制限されたものとなり、引数（argument）となる要素の役割もより特定的なものとして現れてくる。

ここで注目したいのは、構文文法において関数とは、単にある形式から別の形式を導きだすための操作ではなく、複数の意味要素の「重ね合わせ（merge）」を実現するものだということである。CAUSE-RECEIVEという関数はCAUSEとRECEIVEという2つの関数に分解することができる。また動詞自体も一種の関数として捉えることができる。このように複数の関数が連続的に融合するプロセスは計算機科学においてカリー化（currying）と呼ばれているものに他ならない。[4] また、構文文法では関数の連続適用により重ね合わせられる要素が、部分的に重複し合う非排他的な関係にあることも重要である。これらの性質により、構文文法における関数的操作は、必ずしも表現の意味を極小単位の構成要素に還元することにつながらない。したがって、構文文法が「関数的」であるということは、形式主義に立つ言語理

論において統語構造が関数的操作で表現されることとは根本的に異なっている。[5]

2.3 構文が計算的な性質を持つこと

構文文法では、生成文法が規定するような独立した言語部門と生得的な言語アルゴリズムの存在を仮定しない。しかしそのことは、子供の言語習得における計算的プロセス自体を否定するものではない。おそらくこれは、構文文法研究者や認知言語学者の中でも議論の対象となる点であり、また生得主義の考え方に立つ研究者からも誤解されやすい点である。非生得主義、非還元主義は言語的メカニズムの明示的な記述の否定を意味する、という思い込みが一部にあると推測される。しかし、構文文法は当初より計算機科学との接点を意識し、また理論の形式化を重んじてきている。ただしその意図は、言語自体をアルゴリズムとして規定しようとした生成文法のそれとは大きく異なっている。

本節の始めに述べた通り、ここでいう「計算的」とは、言語そのものが一種のアルゴリズムであるということを意味していない。広く論じられている通り、ヒトは複雑な入力からパターンを発見し、経験との参照を行い、整理・ネットワーク化するという能力を持っているが、そのような能力自体は、必ずしも言語だけに特化されたものであるとは限らない。したがって、様々な言語表現の産出を、変形や派生といった手続きによって記述し、それこそが言語のメカニズムであるという考え方は受け容れられない。しかし少なくとも、言語産出の過程を計算的な手法で記述することは可能かつ必要である。それには理論および実際の言語データを部分的にであれ計算機に実装することが求められる。

本節では、まず、構文文法がオブジェクト指向的であり、構文が継承、カプセル化、多態性といった性質を持った一種のオブジェクトとして捉えられることを確認した。構文同士が互いに関連し合い一種のネットワークを形成していることはよく知られている（e.g. Croft 2002; Sag *et al*. 2003; Hudson 2007）。しかし、ネットワークが構成されていく過程についての研究や、言語データからこれを擬似的に構築するという試みは未だ進んでいるとは言え

ない。そこで、次の3節において、形式概念分析を用いた新たな手法を提案する。

また本節では、ある種の関数を連続的に適用していくことで多くの抽象的意味が重ね合わされ、より具体的な意味を持った構文事例が産み出されていくということについて述べた。これは母語習得の過程において観察されることでもある、子供は、周囲とのインタラクションの中で様々な言語パターンに遭遇し、その中から次第に言語的に重要なパターンを断片的に見い出していく。そのようなパターンを実証的に特定できたならば、様々な構文の概念構造や習得に関わる条件がよりはっきりと見えてくる可能性がある。4節では、このような期待のもとに、パターン・ラティスという概念を用いた手法を提案する。

3. 形式概念分析を用いた構文分析

形式概念分析（Formal Concept Analysis）とは、束論と呼ばれる数学理論を基礎としたデータ分析の手法であり、2次元のマトリクス形式にまとめられたオブジェクトと属性それぞれの束から、ある種の概念（concept）を、数学的な厳密さとともに定義することを可能にする（Ganter and Wille, 1999; Ganter *et al.* 2005）。構文文法が理念として持つ記号主義的言語観においては、あらゆるレベルの言語表現は形式と意味との記号的関係に基づいている。形式概念分析を用いると、そのような形式と意味のパターンを「概念」として抽出することができる上、概念間の関係や距離を視覚的にわかりやすい形式で表示できる。これにより、従来は多分に理論的な構築物として扱われてきた構文のネットワークの概念にある種の実体を与えることが可能になる。（Hasebe and Kuroda to appear）。[6]

3.1　形式概念分析の概要

形式概念分析を行うにあたって前提となるのは、概念を構成する要素となるオブジェクト（object）の集合と、それに対応する属性（attribute）である。[7] 形式概念分析では、オブジェクトと属性を含み持つそのような対象領域のことをフォーマル・コンテクスト（formal context）と呼ぶ。そして、オブジェ

クトの集合から成る外延（extension）と、それに対応する属性の集合から成る内包（intension）の組を1つの概念として定義する。計算処理としての形式概念分析は、このように定義された概念をフォーマル・コンテクストの中から抽出することを指す。

ここで、すべてのオブジェクトから成る集合を O、すべての属性の集合を A とおくと、それぞれの部分集合 O_i と A_i から成る概念（O_i, A_i）は次のような特徴を持つ。

(2) (a) $O_i \subseteq O$
(b) $I_i \subseteq$ A
(c) O_i に含まれるオブジェクトの成員はいずれも A_i に含まれるすべての属性を持つ。
(d) O_i に含まれないオブジェクトはいずれも、A_i に含まれる属性をすべては有していない。
(e) A_i に含まれない属性はいずれも、O_i に含まれるオブジェクトすべてには保持されていない。

コンテクストから抽出された概念は半順序集合を成し、全オブジェクトを外延として含むトップノードと、全属性を内包として含むボトムノードを持ったグラフ構造を構成する。このような構造はコンプリート・ラティス（complete lattice）と呼ばれる。概念間の包摂関係（O_i, A_i）≤（O_j, A_j）は、$O_i \subseteq O_j$ かつ $A_i \supseteq A_j$ であるときに成立する。これに基づき、例えば、図2のコンテクスト・マトリクスを形式概念分析にかけると、図3のようなコンプリート・ラティスが得られる。

	bird	mammal	ape	flying	preying	talking
Ostrich	x					
Sparrow	x			x		
Eagle	x			x	x	
Lion		x			x	
Bonobo		x	x			
Human		x	x			x

図2　コンテクスト・マトリクス

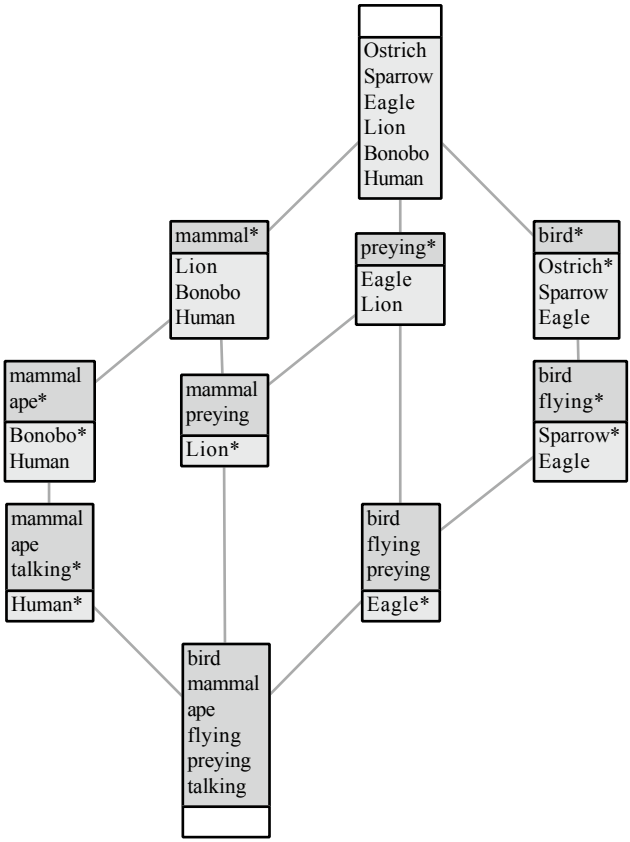

図3　コンプリート・ラティス

図３からは次のような事実が読み取れる。例えば、[bird, flying, preying | Eagle*] というラベルが付けられた概念（下から２番目のランクの右側）には、その外延として Eagle オブジェクトが含まれ、内包としては bird, flying, preying という３つの属性が含まれる。また、その右上の概念は [bird, flying* | Sparrow*, Eagle] となっているが、これは、外延に Eagle と Sparrow が含まれ、内包としては bird という属性が含まれていることを表す（アスタリスクの意味に付いては後述する）。ここで注意すべきことは、すべての概念がグラフ構造中で、オブジェクトと属性の組み合わせに応じた位置付けを与えられていることである。トップから下のレベルへと降りると、段階的に属性が加わっていき、含まれるオブジェクトは制限されていく。反対に、ボトムノードにはすべての属性が与えられており、レベルが上がるにしたがって、属性は限定され、それゆえに含まれるオブジェクトの数は増えていく。

なお、形式概念分析では、通常、各概念のすべての属性とオブジェクトを記すのではなく、ラベルの縮約を行うのが慣例となっている。つまり、属性についてはトップノードから辿って最初に出現する概念にのみ記載することになっている。同様に、オブジェクトについては、ボトムノードから辿り、最初に出現する概念にのみ記載を行う。しかし、ここでは簡潔さより理解の容易さを重視し、このような縮約を行っていない。各概念ノードのオブジェクトおよび属性のうち、アスタリスクの付いたものが、本来のラベル記載項目であり、それ以外は本来縮約の対象となるものであることに注意されたい。

3.2　形式概念分析の言語習得研究への応用

3.1の内容を踏まえ、ここでは、形式概念分析の手法を用いて、子供が言語を習得する過程で構築される構文ネットワークを擬似的に表現する方法を提案する。いわゆる二重目的語構文（double object construction）を具体的な考察の対象とし、入力データには、CHILDES の Brown コーパスから Adam の事例に含まれる母親の発話を用いる。Adam は Brown コーパスで扱われている３人の子供のうちの一人で、データベースには２才３ヶ月から４才10 ヶ月までの合計55回のセッションの結果が含まれている。実際の習得研究に用いるためのデータとしては必ずしも十分でないが、ここでは、言

語習得のただ中にある子供の日々耳にする言語データを分析するための手法の提案を主眼に置いて、この選択を行った。

CHILDES 用の検索プログラム CLAN の実行と手作業による確認の結果、データセットの中に二重目的語構文を含んだ事例は131例あり、その内訳は以下の通りであった（括弧内は出現頻度を表す）。

(3) *give* (65), *show* (22), *tell* (16), *call* (9), *bring* (4), *make* (4), *buy* (2), *read* (2), *serve* (3), *draw* (1), *hand* (1), *pay* (1), *sing* (1)

形式概念分析を行うにあたってまず必要なことは、データ中の何をオブジェクトとし、何を属性と見なすかである。これには具体的な研究の目的により、様々な方法があり得る。今回は、子供に話しかける母親の発話で用いられる二重目的語構文全体の中で、どのような動詞がどのような役割を持つ項と共起しているか、また全体としてどのようなネットワーク構造を成すかを調査するという目的を仮に設定し、(3) に示した動詞タイプのそれぞれをオブジェクトとすることにした。一方、属性については次のような手順で見出していった。まず、二重目的語構文の３つの項 AGT, REC, PAT のうち、REC と PAT の意味属性をすべての事例について特定していった。その結果、REC 項は、PERSON あるいは THING のいずれかとなった。PAT 項については THING, FOOD, STORY, SONG など12種類のラベルを用いてカテゴライズし、代名詞が使用されているなど表現自体から内容が特定できない場合は別に SOMETHING というラベルを用いた。その後、AGT-PAT-REC という項同士の関係を AGT-PAT と PAT-REC に分解し、各オブジェクトが、それぞれ AGT-PAT に対応する属性と PAT-REC に対応する属性の２つを持つものとして、データ整形を行った。[8]　例えば、実際の発話は下の (4a) および (5a) のようなものであるが、これらの例から抽出されるオブジェクトと属性はそれぞれ (4b)、(5b) のようになる。

(4) (a) *can you bring me my pocket book ?*
(b) object: *bring*, attributes: AGT-REC, REC-THING

(5)　(a) *you going to tell her a story ?*

(b) object: *tell*, attributes: AGT-REC, REC-STORY

以上のような作業を経て、得られたコンテクストのマトリクスを形式概念分析にかけたところ、図4のコンプリート・ラティスが出力された。[9]

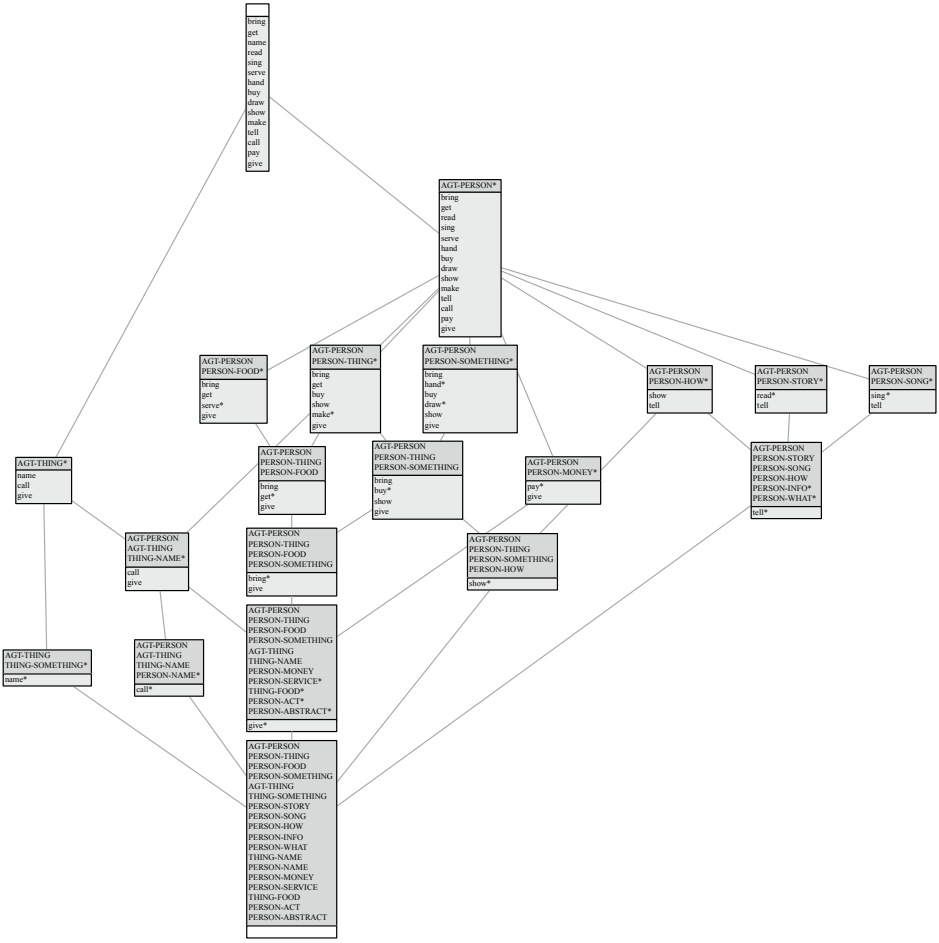

図4　二重目的語構文のコンプリート・ラティス

あくまで予備的な実験ながら、Adam が母親の発話を通じて触れてきた二重目的語構文の事例について、ここから様々なことが読み取れる。まず、用いられる動詞の中で、最も多くの属性を付与されているのが *give* であり、二重目的語構文の中ではこの語が最も多様な使い方をされていることがわかる。グラフ構造のボトムノードに全18の属性が記載されているのに対し、その真上に位置する、give* というラベルが含まれた概念ノードには実に11もの属性が含まれている。このことから、*give* は二重目的語構文に用いられる動詞のプロトタイプとして機能する動詞であると言える。この概念ノードを上に辿っていくと、可能な属性の数は徐々に少なくなっていき、より限定されたオブジェクト（動詞）が見出される。*get* や *buy* がその例である。[10]

その他、グラフの左側を観察すると、AGT-PERSON ではなく AGT-THING という属性を備えたオブジェクト（すなわち動詞のトークン）が非常に少なく、突出して汎用性の高い *give* の他には *name, call* に限られることがわかる。これは Adam の母親にとって、（少なくともコーパス構築のための事例採取のセッションの時間においては）二重目的語構文の REC 項、すなわち第一目的語として人物以外を表す表現を用いることがほとんどなかったことを意味する。*call, name* は明らかに「名を呼ぶ」ことに関する動詞としてグループを成しており、一般にこれらを二重目的語構文の動詞として扱わないことも多いことから、実質上すべての動詞の第一目的語が人物を表していたことになる。

グラフの右側を見ると、*show*, *tell*, *read* などがクラスターを成している。また、これらの語をオブジェクトとして持つ概念の属性の多くは二重目的語構文のプロトタイプである *get* の概念に共有されていない。このことはこれらの語が *call, name* とはまた別の例外クラスを形作っていることを意味する。二重目的語構文は対象の「移動」を基本的な意味として持つと言われるが（Goldberg 1995; Newman 1996; Mukherjee 2005）、これらの語を含んだ二重目的語構文において AGT から REC へと移動する対象の多くが無形の「情報」であることを考えると、これらが例外クラスを形成していることは理に適うものである。

形式概念分析によりコンプリート・ラティスを作成することで、構文を構

成するパターンの構造が「概念のネットワーク」として浮かび上がってくる。また、そのような構造間の関係を、主観的、印象的なスケッチとしてではなく、数学的に厳密な手続きのもとに形式化することができる。これにより、二重目的語構文など特定の構文のデータから、プロトタイプ的な下位構文や周辺的な下位構文を視覚的に発見できるようになる。もちろん、形式概念分析を待つことなく、単なる事例の観察から得られることも多い。しかしそのプロセスを形式化し半自動化することは、利便性はもとより、研究成果の信頼性を増すためにも重要なことである。そして何より、形式概念分析は、構文文法において従来から論じられてきた「構文ネットワーク」の考え方に実体を与えてくれる。

4. パターン・ラティスを用いた構文の分析

パターン･ラティス（pattern lattice）は黒田･長谷部（2009）で提案された、自然言語のパターンを検証するための手法である。パターン・ラティスを用いた分析は、きわめてシンプルなものであり、基本的に、複数の文ないしは発話からパターンの集合を作り出し、その中で複数回出現する共通パターンに重み付けを行うに過ぎない。前節で見た形式概念分析は事例タイプのバリエーションに重きを置き、トークン頻度に関する情報は切り捨てられていたが、パターン・ラティスを用いた分析は、タイプのバリエーションに加えて事例のトークン頻度を重視したものとなる。この手法を用いることにより、子供が環境の中で得る言語データの中で、どのようなパターンが特徴的であり、したがってその子供の言語発達に影響を与えやすいかを、視覚的に明らかにすることができる。

4.1 パターン・ラティスの概要

具体的なパターン・ラティス構築の手順は次のようになる。まずは入力されたデータを、適切と考えられる記号レベルで分節化する。例えば、子供の発話における動詞や名詞の正確な活用形の習得が興味の対象であれば、形態素のレベルで細かくデータを切り分ける必要がある。一方、二重目的語構文のように大きなレベルのパターンを見るのであれば、句のレベルで分節化す

ることになる。それに加えて、パターンの過剰な生成を抑えるため、属性の抽象化を行う。例えば名詞句に対してなら、THING, PROCESS, PERSON といったラベルを用いて、個々の要素の属性を設定する。

次に、各入力データに含まれる要素集合から冪集合（power set）を作る。冪集合とは、ある集合のすべての部分集合を要素とした集合である。例えば、$S = \{x, y, z\}$ の冪集合は、空集合を含む $P(S) = \{\{\}, \{x\}, \{y\}, \{z\}, \{x, y\}, \{y, z\}, \{x, y, z\}\}$ となる。また、冪集合中の要素集合それぞれについて、内部構成要素を定項とし、新たな要素中に含まれなかった構成要素（の痕跡）を変項とする。その際、構成要素の並びはもとの集合のそれを維持し、2個以上の変項の連続は1個に縮約する。これにより、新たにできるパターンの集合は、不要な要素を除外しつつも、もとの発話・文における要素の相対的な並びと境界を保ったものとなる。

パターン集合は冪集合と同様、半順序集合となる。したがって、グラフ構造を持ち、ハッセ図として描くことができる。ただし、パターン集合での集合順序は通常の包含関係ではなく、パターン同士のマッチングにより決定される。図5左が冪集合のハッセ図であるのに対して、パターン集合のハッセ図は図5右のようになる。なお、この図には現れていないが、パターン集合では、たとえ同一の構成要素を含む集合同士であっても、変項の数や位置が異なると別の要素と見なされる。したがって、例えば _ x と x _ は別の要素である。

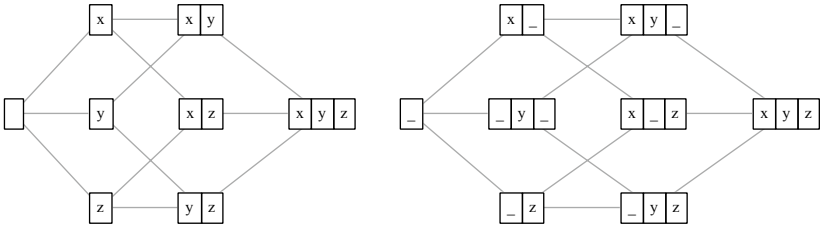

図5　冪集合（左）とパターン集合（右）のハッセ図

パターン・ラティスを用いた分析には、入力データに含まれる構成要素が増えると、集合が指数関数的に増大するという問題がある。具体的には、要素の数が7を超えると、計算量とグラフの大きさが爆発的に増してしまう。ただし、パターン集合をヒトの言語知識構造を模したモデルとして捉えるならば、現実的に扱えるパターンの大きさに限界があることは、むしろ現実の構造を忠実に反映しているとも考えられる（cf. Miller 1956; Kuroda to appear）。

4.2 パターン・ラティスの言語習得分析への応用

では、パターン・ラティスの手法は実際の言語発達研究においてどのように利用できるだろうか。ここでは、前節と同様に、CHILDS Brown コーパスの Adam サブセットを用いることにする。ただし、前節では母親の発話から二重目的語構文の事例を抽出したデータを用いたが、ここでは *give* が用いられた事例を具体的な構文の別に関わりなく採取して用いることにする。これにより、子供が *give* という動詞に触れる際、どのような項構造およびイベントフレームを情報として共に得ているかを（大まかにではあるが）知ることができる。データセット中の事例数は106であり、現実的な調査のためには少ないが、手法を提案するというここでの目的には適う。

CLAN を用いて (6a) や (7a) のような事例を抽出した後、(6b) や (7b) のようなパターン・データを付け加える。なお、今回はグラフ構造のサイズを抑えるため、動詞句の核となる部分についてのみパターンの抽出を行い、付随する主語や副詞句などは取り除くことにした。

(6) (a) *give me your cup* .
(b) *give* PERSON THING

(7) (a) *oh* (.) *that is Ursula's* (.) *you give it to Ursula* .
(b) *give* THING *to* PERSON

106個のデータから成るリストを用いて行ったパターン・ラティス分析の

結果は次のようなものであった。[11]

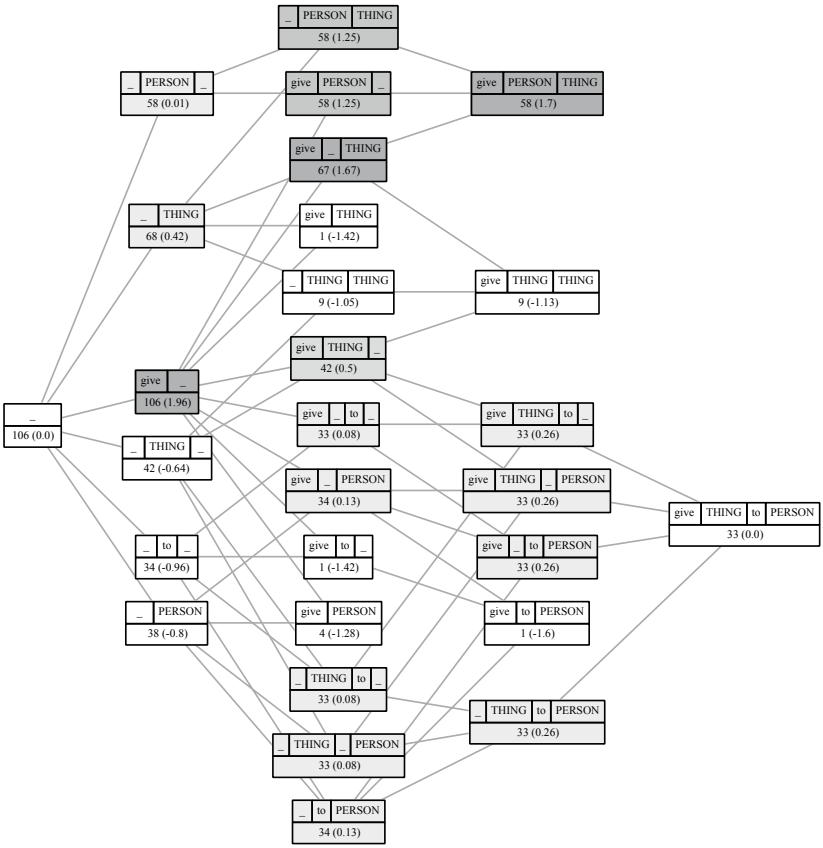

図6　*give* のパターン・ラティス

対象としたのが *give* という単一の動詞タイプに関する事例ということで、パターンのバラエティは限られている。しかし、パターン・ラティスについて興味深いのは、あらゆる入力パターンがランク化されている点である。各ランクは同じ数の定項を持つパターンで構成されており、したがって、基本的に同じランクに属するパターンは形式上は等価である。しかし、2つの要

因により、同一ランクのパターンには際立ちの差が生じている。第一に、パターンとしての生産性が異なる。例えば、図6の左から数えて3つ目のランク上で、_ PERSON THING というパターンは単一の子パターンしか産み出していないのに対し give THING _ パターンは3つの子パターンを持っている。当然ながら後者の方がタイプ頻度が高いパターンであるということになる。第二に、各パターンが事例として実現される頻度もそれぞれに異なる。このことを視覚的に反映させるため、図では同一ランク（縦の列）における事例数の標準化スコア（z-score）を産出し、グラフの各ノード下段に表示している。また、正の標準化スコアを持つパターンについては、数値の大きさに応じた濃さのシェードを与えている。

このようにパターン・ラティスを生成することで、言語データ内の様々なパターンがどのようなバラエティを持ち、各パターンが子供の言語習得環境においてどれだけの際立ちを持つかを明確にすることができる。2節では、構文は下位の構文の再帰的な重ね合わせであることについて触れたが、パターン・ラティス分析により得られるパターンの数々は、ある構文を構成するため重ね合わされる下位構文の候補群と考えられる。タイプ頻度とトークン頻度の高いパターンを特定し、それらを精査することで、ある構文を完全に習得するための前提を特定し、さらに、下位構文間の関係を見出すことができる。先に述べた通り、パターン・ラティスは構成要素の数が一定以上になるとノードの数が爆発的に大きくなる性質があり、大規模なデータを用いたプロジェクトでの使用には難しい部分もある。しかし、活性度の低いパターンを自動的に除外するなどの方法により、今後さらに改良を進めることが可能と考えられる。

5. まとめ

本稿では、構文文法を言語習得研究に応用するにあたり、計算的な手法を用いることの有用性を論じた。2節では、理論的な前提として、構文文法が計算的な性質を持つことについて述べた。構文文法がオブジェクト指向的であること、また関数的であることについて触れ、構文ネットワークや構文の重ね合わせといった概念がこれらの特徴と密接に関わっていることを示し

た。次に3節では、形式概念分析と呼ばれる手法を導入し、これが構文文法とそれに基づいた言語習得研究に応用可能であることを示した。形式概念分析は、従来、多分に理論的な構築物として扱われてきた構文ネットワークの概念に実体をもたらしてくれる。最後に4節では、パターン・ラティスによる構文データ分析の可能性を示した。パターン・ラティスによる分析の研究はまだ新しく、解決すべき問題もあるが、子供の構文パターン習得の過程をモデル化するための枠組みとして非常に有望と考えられる。

1節で述べた通り、認知言語学の一部としての構文文法に計算的な手法は馴染まないと考える人々は少なくない。しかしながら、構文文法はそもそも、計算機科学におけるオブジェクト指向のパラダイムと密接な関係にあり、計算的な手法を用いることは決して本来の理念に反することでない。[12]　今後、この点に対しさらに考察の光を当てていくことで、新たな構文研究および言語習得研究の地平が開けてくると期待される。

謝　辞

本稿の内容の一部は独立行政法人情報通信研究機構の黒田航氏との共同研究によって得られた知見に基づいている。氏の協力なしに本稿の研究はあり得なかった。なお、本研究の一部は日本学術振興会科学研究費補助金若手研究(B)（課題番号：21720179）の助成を受けて行われた。

注

1　したがって、前述の *kick the bucket* は、例えば [CONST[VP[V *kick*] [CONST[NP *the bucket*]]]] のように、構文の入れ子構造として記述することができる。

2　CHILDES とは the Child Language Data Exchange System の略で、Brian MacWhinney によって提唱された、第一言語習得における子供の発話（両親など周囲の発話者によるものを含む）を集積したデータベースである。採取されたデータは電子化された後、文法や発話行為の観点からタグ付けされ、研究者の様々なアプローチに応えられるものになっている。現在、英語をはじめとする20以上の言語において展開されており、基本的に研究者は誰でも自由に利用可能である（MacWhinney 2000）。

3 本稿で論じるように構文文法が計算的な性質を持つことがまったく知られていないというわけではもちろんない。近年、統計的な手法を用いて特定の構文の性質を浮き彫りにする試みが盛んに行われている（Gries and Stefanowitsch 2006）。さらに、構文文法を計算機に実装するプロジェクトも存在する（Bergen and Chang 2003; Steels and De Buele 2006）。

4 現在「関数型」と呼ばれているタイプのプログラミング言語（Miranda, Haskell, 等）の理論的基盤である「組合せ論理」の完成に貢献した数学者 Haskell Curry の名に因む術語である。

5 重ね合わせ（merge）という用語は生成文法のミニマリスト・プログラムにおいても用いられているが（Chomsky 1995）、本稿におけるそれとは定義が異なる。

6 独立行政法人情報通信研究機構の黒田航氏と筆者は、本節および次節において提案する言語分析の手法の実践に利用できるコマンドラインプログラム RubyPLB および RubyFCA を開発し、公開している。詳しくは http://kotonoba.net/rubyfca を参照されたい。

7 ここでいうオブジェクトは、前節で論じた「オブジェクト指向プログラミング」におけるオブジェクトの概念と必ずしも一致するものではないが、対象領域に存在する（多くの場合、何らかの実体を持った）事物を表現するものであるという点では共通しており、ここで違いを明確にすることはしない。

8 関数の分解と重ね合わせについては、2.2節を参照のこと。

9 データ処理とグラフ構造図式の出力には形式概念分析ツール RubyFCA を用いた。

10 図4のようなコンプリート・ラティスに記載された属性の束は積集合（AND）ではなく和集合（OR）であることに注意されたい。したがって、属性の数が増すほど、オブジェクトの具体性・個別性が高くなる。

11 データ処理とグラフ構造図式の出力にはパターン・ラティス分析ツール RubyPLB を用いた。

12 もちろん、計算処理の結果から言えることの範囲を厳しく限定することは必要である。

参考文献

Bergen, Benjamin K. and Nancy Chang. 2003. “Embodied construction grammar in simulation-based language understanding,” in Östman, Jan-Ola and Mirjam Fried eds. *Construction Grammars: Cognitive Grounding and Theoretical Extensions*. Amsterdam: John Benjamins, 147-190.

Chomsky, Noam. 1995. *The Minimalist Program*. Cambridge, MA: MIT Press.

Clark, Eve V. and Barbara F. Kelly. eds. 2006. *Constructions in Acquisition*. Stanford: CSLI.

Croft, William. 2002. *Radical Construction Grammar: Syntactic Theory in Typological Perspective*. Oxford: Oxford University Press.

Fillmore, Charles J., Paul Kay, Laura A. Michaelis, and Ivan A. Sag. 2009. *Construction Grammar*. Chicago: University of Chicago Press.

Fillmore, Charles J., PaulKay, and Catherine O'Connor. 1988. "Regularityand idiomaticity in grammatical constructions: The case of *let alone*," *Language*,Vol. 64, 501-538.

Ganter, Bernhard, Gerd Stumme, and Rudolf Wille. eds. 2005. *Formal Concept Analysis: Foundations and Applications*. Berlin: Springer.

Ganter, Bernhard and Rudolf Wille. 1999. *Formal Concept Analysis: Mathematical Foundations*. Berlin: Springer.

Goldberg, Adele E. 1995. *Constructions: A Construction Grammar Approachto Argument Structure*. Chicago: University of Chicago Press,（河上誓作・谷口一美・早瀬尚子・堀田優子訳,『構文文法論―英語構文への認知的アプローチ』東京：研究社, 2001年）.

Goldberg, Adele E. 2006. *Constructions at Work: The Nature of Generalization in Language*. Oxford: Oxford University Press.

Gries, Stefan Th. and Anatol Stefanowitsch. eds. 2006. *Corpora in Cognitive Linguistics*. Berlin: Mouton de Gruyter.

Hasebe, Yoichiro. 2005. "Computer analogy reconsidered froma perspectiveof cognitive linguisticsand object-oriented programming," in *Proceedings of the Fifth Annual Meeting of the Japanese Cognitive Linguistics Association*, 126-36.

Hasebe, Yoichiro and Kow Kuroda. to appear. "Extraction of English ditransitive constructions using Formal Concept Analysis," in *Proceedings of PACLIC 23: The 23rd Pacific Asia Conference on Language, Information and Computation*.

Hudson, Richard. 2007. *Language Networks: The New Word Grammar*. Oxford: Oxford University Press.

Kay, Paul and Charles J. Fillmore. 1999. "Grammatical constructions and linguistic generalizations: the *what's X doing Y?* construction," *Language*, Vol. 75, 1-33.

黒田航・長谷部陽一郎 . 2009.「Pattern Lattice を使った（ヒトの）言語知識と処理のモデル化」,『言語処理学会第15回大会発表論文集』670–673.

Kuroda, Kow. to appear. "Pattern lattice as a model for human linguistic knowledge and performance," in *Proceedings of PACLIC 23: The 23rd Pacific Asia Conference on Language, Information and Computation.*

MacWhinney, Brian. 2000. *The CHILDES Project: Tools for Analyzing Talk*. Mahwah, NJ: Lawrence Erlbaum, 3rd edition.

Meyer, Bertrand. 1997. *Object-Oriented Software Construction*. Englewood Cliffs, NJ: Prentice Hall.

Miller, George A. 1956. "The magical number seven, plus or minus two," *The Psychological Review*, Vol. 63, No. 2, 81-97.

Mukherjee, Joybrato. 2005. *English Ditransitive Verbs: Aspects of Theory, Description and a Usage-based Model*. Amsterdam: Rodopi.

Newman, John. 1996. *Give: A Cognitive Linguistic Study*. Berlin: Mouton de Gruyter.

Östman, Jan-Ola and Mirjam Fried. eds. 2008. *Construction Grammars: Cognitive Grounding and Theoretical Extensions*. Amsterdam: John Benjamins.

Sag, Ivan A., Thomas Wasow, and Emily M. Bender. 2003. *Syntactic Theory: A Formal Introduction*. Stanford: CSLI, 2nd edition.

Steels, Luc and Joachim De Beule. 2006. "Unify and merge in Fluid Construction Grammar," in *Symbol Grounding and Beyond: Proceedings of the Third International Workshop on the Emergence and Evolutionof Linguistic Communication*, 197-223.

Tomasello, Michael. 2005. *Constructing A Language: A Usage-Based Theory of Language Acquisition*. Cambridge, MA: Harvard University Press.

Tomasello, Michael. 2008. *Origins of Human Communication*. Cambridge, MA: MIT Press.

Computational Methods for Analyzing Language Acquisition Data in the Framework of Construction Grammar

Yoichiro HASEBE

Keywords: construction grammar, language acquisition, formal concept analysis, cognitive linguistics

This paper argues for the benefits of adopting computational methods in studying language acquisition mechanisms within the framework of Construction Grammar. First, the paper discusses the object-oriented and function-oriented facets of the computational nature of Construction

Grammar. Upon this theoretical foundation, two new methods of language data processing are introduced. The first method, a technique utilizing Formal Concept Analysis (FCA), makes it possible to visualize the conceptual network hidden inside the real data. The second technique utilizes Pattern Lattice Theory to clarify the multiple overlapping sub-constructions that are supposedly acquired by a child before he or she masters a construction. Through these discussions and proposals, this paper argues against the myth that the framework of Construction Grammar is in conflict with, or is not compatible with, computational methodologies.