意味論における演算についての一考察

三藤　博

1.　はじめに

筆者は三藤(2015)において、現在の生成文法統語論において規定されている統語構造の binary branching（二股分かれ）について、意味論的操作との関連を中心として、人間が行なう演算操作という観点から考察し、統語構造が binary branching でなければならないことを統語論に対しては出力側として対応している意味論的操作の観点から裏付けることは難しく、むしろ逆に、論理結合子に相当する英語の and, or 等を代表例として、一つの演算子が二つの要素を取る構造の方が自然であると見られる場合もあることを論じた。

本稿では三藤(2015)における議論をさらに発展させつつ、統語論と意味論の対応関係についてさらに考察を深めたい。

2.　三藤(2015)

まず、三藤(2015)における議論を簡単に整理しておこう。

生成文法統語論の現在の理論体系であるミニマリズムにおいては基本的統語操作として Merge が規定され、この基本操作 Merge が二つの要素αとβから新たな要素を作り出す操作として定義されていることから、でき上がる統語構造がすべて binary branching で構成されることとなる。ところが、意味論の観点からこの構造の問題を考えてみると、純粋に意味論的な観点からは、すべての節点が binary branching になっていなければならないという必然性は認められないばかりか、むしろ逆に、例えば ternary branching（三つ叉分かれ）があってもなんら不思議はない、と言える。さらに進んで、2 つの要素αとβから新しい要素を作り出す操作としては、代数学にまとめられているような 2 項演算「α・β」の形式の方がむしろより基本的な操作と捉えられるべきである。また、binary branching を意味論サイドに対応させた関数－項の構造「α(β)」はより一般的な構造である「$\alpha(\beta_1, \beta_2, ..., \beta_n)$」において n＝1 とした特殊ケースであり、2 項演算「α・β」のような人間の演算処理におけるいわば特権的な基本演算としての地位を認めることは難しい。そしてこのことは、統語論における Merge の操作とその結果としての統語構造の binary branching を二つの要素α、βに対する最も基本的な操作であるという観点から根拠づけることは難しいことをも示唆している。三藤(2015)では以上のように論じた。

本稿では、三藤(2015)の続編として、特に生成文法統語論の現在の理論フレームワークであるミニマリズムに対応する意味解釈メカニズムとしてどのようなものを考えるべきかについて検討していきたい。

3. 関数適用(functional application)と主要部に関する情報

ミニマリズムの近年の考え方では、統語構造の派生の過程においてフェイズという段階が設定され、派生がフェイズに達するごとに Transfer という操作によってその段階までに構成された統語構造が Conceptual-Intentional System に出力される、と想定されている。意味解釈メカニズムにとってはこの出力結果が意味解釈のための入力となると考えられる。

ミニマリズムは統語論の理論であり、当然のことながらこの出力結果のその後の処理については統語論の枠外の問題となるため、ミニマリズムの文献では具体的な議論はほとんど行なわれていないように見受けられる。そこで本稿では、このミニマリズムにおける Tranfer に対応する意味解釈メカニズムについて具体的に考察してみたい。

まず初めに、Transfer によって Conceptual-Intentional System サイドに送られてくる統語構造が（とりわけ派生の初期の段階においては）断片的な構造に留まっている、という問題がある。(形式）意味論では、真理値が定まる文（命題）を出発点としてそこからトップダウンに各構成素の意味解釈を定めていくアプローチが取られることもあるが、当然のことながら Transfer からの出力を処理する場合にはこのようなアプローチを採ることはできない。したがって必然的に、各構成素の意味解釈からその親ノードの意味解釈を決定するボトムアップのアプローチを採ることになる。もちろん、ボトムアップのアプローチは意味論でも普通に採られているアプローチであり、全く問題はない。ただ、意味解釈メカニズムが入力として受け取る統語構造が文全体ではなくその一部の構成素に留まっていることに関しては、意味解釈メカニズムに対してそれ相応の制約を課すことになってこざるを得ないであろう。

現在ミニマリズムでは、以前のX-bar Theoryに基づく統語構造の枠組み指定のようなものも過剰な規定であったとして廃棄され、さらにはMergeの結果としてできる親ノードのラベルも必要ないとする、いわゆるBare Phrase Syntaxの方向性が強く打ち出されるようになっている。このBare Phrase Syntaxにおいて現在活発な議論の対象となっているのが、親ノードのラベルがなくなった場合に主要部(Head)[1]に関する情報をどのように取り扱うのか、という問題である。

二つの要素αとβが統語操作 Merge によって結ばれた時、αとβとのどちらが新しく構成された統語構造の主要部となるかは、αとβが具体的にどのような性質の要素であるかに依存して Merge 操作のたびごとに決定されるものと考えられるが、そもそも Merge 操作の結果として構成された統語構造においてどちらか一方が主要部というステータスを獲得しなければならないということは、X-bar スキームなどが廃棄された後となってはそれほど自明のこととは言えないであろう。このことに関してはミニマリズムにおいても当然議論の対象となっており、たとえば Narita(2011)では、Is headedness relevant to narrowly syntactic computations or does it arise only at SEM/PHON? という問題設定

[1] Head に対する日本語の用語としては「主辞」も広く使われているが、本稿では「主要部」を用いる。

を行なった上で、これまで headedness との関連が必要だと考えられてきた様々な操作に関して、headedness に関する情報がミニマリズムの精神から純粋に統語的に(purely syntactically)必要であるかどうかを検討している。たとえば Tranfer の操作に対しては、統語構造のどの部分を Tranfer すべきかは構成された統語構造から主要部に関する情報とは全く独立に決定できるとし、主要部に関する情報は不必要であると結論づけている。Transfer 以外の統語操作についても、主要部に関する情報を参照しなければ遂行できないような操作は存在しないとし、さらには Merge はでき得る限り単純な形で定式化されるべきであるというミニマリズムの根本精神に則って、次のように結論づけている。

(1) the notion of headedness is purely interpretive and arises only at SEM/PHON, meaning it is irrelevant to narrowly syntactic computation.

Narita(2011) p. 7.

音声出力の問題は本稿の議論の及ぶ所でないので今は措くとして、(1)の結論を意味解釈の機構に結びつけて考えると、主要部に関する情報は意味解釈のプロセスにおいて必要となる、と理解するのが最も自然な捉え方であろう。このようなミニマリズムにおける議論を受けて、本稿では、意味解釈のプロセスに主要部に関する情報が必要かどうかについて検討してみることとしたい。

意味論における計算(semantic computation)は、関数適用(functional application)が基本である。関数適用の操作は、narrow syntax から Transfer 操作によって送られてきた統語構造に対して、当該統語構造は Merge 操作によって作られたものである以上二つの構成素 α と β から成っているはずなので、その α と β のうちどちらか一方を関数、他方を項(argument)と見なして適用されることとなる。この時に、統語構造上での主要部が関数、補部が項となると考えるのが最も自然であり、実際に形式意味論の処理においても、このことは当然のこととして暗黙のうちに前提されているような所がある。しかしながら、ミニマリズムにおける主要部の指定をめぐる最近の議論を踏まえて改めて考え直してみると、意味論の基本的要請である「統語論と意味論との準同型対応関係」は、統語論的操作（これは現在のミニマリズムでは Merge 操作に一本化されている）と意味論的操作（こちらも基本的には関数適用）との間に準同型の対応関係がつかなければならないということであって、統語論、意味論の双方における操作の具体的内容にまでは立ち入っていない。つまり、統語論における主要部が意味論における関数となり補部が項となる、ということは、自然な対応関係であることは間違いないとしても、決して論理的な（あるいは、意味論的な）必然ではないのである。

このことをより具体的に見るために、(2)のような英語の動詞句とその意味解釈の例について考えてみよう。

(2) a. know John
b. know ⇒ λxλy.know(y, x)
c. John ⇒ j
d. know John ⇒λxλy.know(y, x)(j) = λy.know(y, j)

(2)の意味解釈過程はスタンダードなもので、関数適用の操作において統語的な主要部である know を関数とし、補部である John を項として適用している。ところが、関数適用の可能性としては、(2)とは関数と項との関係を全く逆にして、John の方を関数とし know を項とすることもできる。この場合、意味解釈過程は(3)のように進むこととなる。

(3) a. know John
b. know ⇒ λxλy.know(y, x)
c. John ⇒ λP.P(j)
d. know John ⇒(λP.P(j))(λxλy.know(y, x))= λxλy.know(y, x)(j)
= λy.know(y, j)

ただし、(3c)における P は、know を項として取れるように<<e,<e,t>>,<e,t>>というやや複雑なタイプの述語であると規定する必要があるが、これは(3c)が全体として「John が内項となり得るような二項述語全体の集合」と同等（より厳密に言えば、この集合の特徴関数(characteristic function)）の意味解釈を持つために必要となることで、より基本的な(2c)からのタイプシフトによって(3c)が得られると考えることができる。

このように、意味論サイドには必要に応じてタイプシフトによって関数と項との組み合わせ方を入れ替えることができるメカニズムが組み込まれているため、原理的には、どのような二股分かれ統語構造に対しても、どちらの構成素も関数となって他方の構成素を項として取ることができるわけである。

もう一つ英語から具体例を考えてみよう。

(4) a. Mary knows John.
b. knows John ⇒λy.know(y, j)
c. Mary ⇒ m
d. Mary knows John ⇒(λy.know(y, j))(m)＝know(m,j)
e. Mary ⇒ λP.P(m)
f. Mary knows John ⇒ (λP.P(m))(λy.know(y, j))＝(λy.know(y, j))(m)＝know(m,j)
g. everyone ⇒ λQ.∀x(person(x)→Q(x))
h. everyone knows John ⇒ (λQ.∀x(person(x)→Q(x)))(λy.know(y, j))
＝∀x(person(x)→λy.know(y, j) (x))＝∀x(person(x)→know(x,j))

(3)で組み上げられた know John が外項（主語）と結合して文を完成させる段階である。統語論サイドでは Tense に関連するレベル等が間に介在しているが、当然のことながらいずれかの段階で必ず外項を Merge することとなるので、(4)はそれが意味論サイドに Tranfer されてきた段階における意味論での演算の可能性を見ているものである。(4)においても(3)と全く同様に、動詞句 knows John の方を関数とし主語 Mary を項とするスタンダードな(4b)から(4d)の関数適用と並んで、逆に主語 Mary の方を関数とし動詞句 knows John を項とする(4e)から(4f)の関数適用も可能である。これに対して、文 Everyone knows John.に対応する関数適用の場合には、一般化量化子としての everyone の方が関数となり動詞句が項となる(4g)から(4h)のプロセスが圧倒的に自然なプロセスである。もちろん原理的にはこの場合も動詞句を関数にするような関数適用も考えられないわけではないが、そのような関数適用においては動詞句 knows John のタイプが非常に複雑なものとなってしまい、(4b)の自然な意味解釈（ジョンを知っている人の集合）に比べてあまりにも不自然である。つまり、(4g)から(4h)にかけての関数適用の過程では、一般化量化子である everyone の方を関数とする方が、余剰なタイプシフト操作を必要としない分だけいわば計算コストが低いと考えることができる。

以上見てきたように、組み上げられた統語構造が意味論サイドに Transfer されてきたとき、主要部に関する情報は意味解釈において必ずしも必要ではないことが分かる。ただし、タイプシフトによって不必要に複雑なタイプを構成しても、関数適用の結果が同じものになるという場合には、そのようなタイプシフトは経済性の原理に反すると言える。すなわち、タイプシフトの適用に関しては次のような原則があるものと考えられる。

(5) 関数適用に際して、出力の意味解釈に影響を与えないような（高階方向への）タイプシフトは、操作の経済性に反しており、適用されない。

(5)のような趣旨の制約は、これまで先行研究でもしばしば提起されてきたものである。(5)が実際の意味解釈過程において及ぼす効力は、次のように考えられる。たとえば上で見た know John の場合、(2)のように関数適用を進めて(2d)のように適正な出力を得ることができる。このような場合にわざわざ(3)のような関数適用を行なうためには、直接目的語の John に対して(3c)のような意味表示(denotation)を与える必要があるが、この意味表示(3c)は明らかにより単純な(2c)を高階（より複雑な）方向にタイプシフトさせたものに当たっている。このような場合には、(5)の効果が働いて、余剰なタイプシフトを必要とするような(3)のような関数適用は発動されない、と考えるのである。(4a)から(4f)の主語（外項）と動詞句との結合においても全く同様のメカニズムが働き、(4b)から(4d)に至る関数適用が適正な出力(4d)を与える以上は、(4c)を高階方向にタイプシフトした(4e)を必要とする(4f)のような関数適用は発動されないこととなる。それに対して、主語が一般化量化子と

なっている(4h)の場合には、今度は everyone に対する意味表示(4g)が最も基本的なタイプのものとなっているため、主語の方を関数とする(4h)の関数適用が実行されるのである。このような意味解釈メカニズムが計算機構のあり方としても自然であると考えられるのは、(5)を適用するに当たって二つ以上の関数適用を行なってみた上で出力の意味解釈が異なるのかどうか、もしも異ならないとすればどちらの適用過程の方がより単純か、というような、二つ以上の意味解釈メカニズムの間での比較衡量を行なう必要が全くないからである。すなわち、まず各構成素に対して最も基本的なタイプの意味表示を対応させ、それを用いて関数適用を行なって適正な出力が得られたのであれば、その適用過程自体を適正な過程と判定することができるのである。

このように考えてくると、意味解釈過程においては、各構成素に対して最も基本的なタイプの意味表示を対応させるメカニズムさえ備わっていれば、関数適用においてどちらの構成素を関数として扱いどちらの構成素を項として扱うかは意味論サイドで独自に決定できることになる。残るのは、各構成素に対して最も基本的なタイプの意味表示を対応させるメカニズムがどのようになっているのかという問題であるが、これは自然言語の意味論の内部だけで完結できない問題（人間の外界に対する認識一般の問題と密接に関連していることなども考えられる）である可能性もあり、意味論の演算の基礎についてさらに考察を深めていく中で今後の研究課題としたい。

4. Merge 操作の binary branching 再考

第 2 節でも簡単に振り返ったとおり、筆者は三藤(2015)において、統語論における Merge の操作とその結果としての統語構造の binary branching を二つの要素 α、β に対する最も基本的な操作であるという観点から根拠づけることは難しいことを示唆した。三藤(2015)を執筆した後、この統語構造の binarity について非常に興味深い考え方に接したので、ここでその考え方を取り上げて考察してみたい。

その考え方とは、藤田耕司氏が提案している「ポット方式併合(Pot-Merge)」と「サブアセンブリ方式併合(Sub-Merge)」との区別、という考え方である。この考え方は、元々は Greenfield の「行動文法」における区別に遡り、類人猿など知能の発達した動物と人間の行動パターンを分類したものであったが、藤田氏はこれを生物言語学的観点から自然言語の統語構造を組み上げる操作としての Merge に発展させて適用しているものである。たとえば形は同じであるがサイズが大、中、小と異なる 3 つの容器があったとして、これらを重ねて 1 つにまとめる、という操作を考えてみよう。このとき、「ポット方式」とは、操作の対象を一貫して 1 つの対象に固定して操作を適用していく過程で、具体的には、一番大きな容器を操作の対象として固定し、まず中サイズの容器を一番大きな容器に重ね、次に一番小さな容器をその上に重ねて全体を 1 つにまとめる、という手順である。これに対して、「サブアセンブリ方式」とは、名称の示すとおりいわば「部品」を組み上げていく様式であり、今の例に当てはめると、まず先に中サイズの容器の上に一番小さな容器を重ね

て一つの「部品」を作り、次にこの「部品」全体を一番大きな容器に重ねて全体を 1 つにまとめる、という手順である。結果としてでき上がるものは同一であるが、そこに至る手順が異なっているのである。

藤田氏はこの「ポット方式」と「サブアセンブリ方式」を統語操作としての Merge に発展させてそれぞれ Pot-Merge と Sub-Merge とし、この両者の区別を用いて、日英両語の統語構造の対照研究をはじめ広範な分野において分析を行なっているのであるが、本稿の観点から極めて興味深いのは、この Pot-Merge と Sub-Merge の概念から Merge 操作の binarity に極めて自然な説明が与えられる可能性が見えてくるという点である。

藤田氏の Pot-Merge と Sub-Merge は、単に Greenfield の「行動文法」における「ポット方式」と「サブアセンブリ方式」をミニマリズムにおける Merge に「当てはめた」というようなものではなく、氏が提唱されている Merge 操作の「運動制御起源仮説」に基づいて、生物進化の観点から操作として連続性を有するものと想定されている点に注意しなければならない。また、操作の複雑さの観点からも分かるとおり、「サブアセンブリ方式」は「ポット方式」よりも後発でより進化したストラテジーであり、基本的には人間に固有の操作能力であると考えられている。さて、Pot-Merge と Sub-Merge という統語操作を考えてみると、元となっている「ポット方式」と「サブアセンブリ方式」における上に挙げた 3 つの容器を 1 つにまとめる例が非常に分かりやすく示しているとおり、どちらの操作においても binarity がごく自然に生じてくることが分かる。Pot-Merge 操作においては、一貫して操作の対象となる要素が 1 つ固定されているのであるから、たとえば 3 つの要素 α、β、γ があって γ が操作の対象として固定されているという状況を考えると、全く当然のことながら最初の操作の対象としては残る α か β かのいずれかしかなく、このどちらかが γ に対して操作され、その次に最後に残った方が操作される、という手順になる。このことを，上に挙げたサイズの異なる 3 つの容器の例でより詳しく考えてみると、ちょうどサイズの小さい順に α、β、γ とおくことができ、最初に β と γ を操作し、次に残った α を操作すれば全体を 1 つにまとめることができるのに対して、逆に最初に α と γ を操作してしまうと次に β を操作することができなくなる。このことはちょうど、それぞれ統語構造の組み上げにおける派生の収束(converge)と破綻(crash)に対応していることが分かる。言うまでもなく、Sub-Merge に関しても全く同じことが成り立つ。同じく上の 3 つの容器の例で考えると、最終的に全体を 1 つにまとめるためには、「部品」としては α と β との組み合わせしかないことが分かる。

三藤(2015)では、数学（代数学）や論理学における演算との比較に基づいて、二つの要素 α と β に対する操作として Merge 操作が最もシンプル、基本的なものとは必ずしも言えず、このことと直接関連して、Merge 操作の結果が binary branching となることも必ずしも経済性の原理等からは帰結されないことを議論した。これに対して、以上見てきたように、「運動制御起源仮説」に立脚して高い知能を有する類人猿などの行動パターンである「ポット方式」、「サブアセンブリ方式」から進化生物学的に発展させられた藤田耕司氏が

提案している Pot-Merge と Sub-Merge は、自然な形で binarity を出すことができる点で極めて注目に値する。三藤(2015)では、統語部門における Merge や意味部門における関数適用を数学や論理学における演算、計算との類推によって考えたため、二つの要素αとβとに操作を及ぼすには演算子を介してα・βという形となる（数学における「α＋β」や論理学における「α∨β」など）のがむしろ最も基本的な形式ではないかと論じたり、関数適用においても引数（項）を一つ取る関数 f(x)がたとえば引数（項）を二つ取る関数 g(x,y)に比べてより基本的であるということは数学的には言えない、という議論を立てたりしたのであるが、このような議論は Merge を演算、計算ととらえるあまり、「演算」、「計算」という用語に言わば引かれる形で Merge を数学や論理学における演算、計算と基本的には同種のものとしてとらえていたのであった。これに対して、藤田氏が提案されている Pot-Merge と Sub-Merge は、類人猿などの知能の高い動物が容器を操作する際に見られるような行動パターンから出発して「運動制御起源仮説」によって統語操作としての Merge に結びつけたものであり、「演算」、「計算」よりもむしろ「操作」という側面を前面に立てた考え方として、人間の持つ言語能力の根幹をなす Merge 操作の解明に向けて極めて注目すべき考え方であると言えよう。

5. おわりに

本稿では、三藤(2015)に引き続いて、統語論と意味論における主要部決定の問題について考察し、意味論における基本操作である関数適用には統語論における主要部の情報は必要ないことを論じた。また、三藤(2015)では数学（代数学）や論理学における演算との比較に基づいて Merge 操作の binarity について演算操作の単純性、基本的性質といった観点からは必ずしも導出されないことを論じたが、藤田耕司氏が提案している Pot-Merge と Sub-Merge という概念によれば binarity が極めて自然な形で導出されることを指摘した。

参 考 文 献

藤田耕司(to appear)「受動動詞の日英比較―生物言語学的アプローチの試み―」in藤田耕司・西村義樹（編）『日英対照・文法と語彙への統合的アプローチ―生成文法・認知言語学と日本語学―』開拓社。

三藤　博(2013)「意味論の基礎についての一考察」『自然言語への理論的アプローチ』（大阪大学言語文化共同プロジェクト2012）, pp. 41-48.

三藤　博(2015)「「理論の意味論的捉え方」と言語学」『自然言語への理論的アプローチ』（大阪大学言語文化共同プロジェクト2014）, pp. 41-48.

Narita, H.(2011) Remarks on the Nature of Headedness and Compositionality in Bare Phrase Syntax, *Proceedings of Sophia University Linguistics Society 26*.