ICOT Technical Memorandum: TM-0805

TM-0805

言語に関する制約の処理

橋田浩一

September, 1989

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

言語に関する制約の処理

橋田 浩一

1 情報の部分性

ある問題を制約充足問題 (constraint satisfaction problem) として見るということは、その問題を、制約 (constraint)、即ち何らかの対象の構造に関する情報によって規定するということである。その場合、伝統的な意味において「問題を解く」とは、その制約を満足するような対象の構造を求めることである。計算機を用いた情報処理技術が成功を収めて来たのは、完全情報問題 (total-inoformation problems)、即ち、関係する情報を完全に参照できるような問題の領域においてであった。ここで、情報を完全に参照できるとは、第1に情報が完全に与えられており、しかも与えられたその情報を現実的な計算量の範囲内で完全に処理し尽くせるということである。

ところが、世の中の実際的な問題のほとんどは、部分情報問題 (partial-inoformation problems) である。つまりそこでは、関係する情報を部分的にしか参照することができない。このことを情報の部分性 (partiality of information) と言う。情報の部分性には、関係する情報を完全には知り尽くせない (知識の部分性)、および、知っている情報を完全には処理し尽くせない (処理の部分性)、という2つの側面がある。知識の部分性とは、たとえば、明日の天気がわからない、ということであり、処理の部分性とは、極端な例を挙げれば、将棋の規則を知っていても必勝手順がわかるわけではない、ということであり、どちらも日常的に普通に生じるごく当り前の事情である。

ロボットにせよ人間にせよその他の生体にせよ、莫大な情報に満ちた世界の中に置かれた、情報処理能力の限られた行為者は、いずれも情報の部分性から逃れられない運命にある。このような意味において、AIの問題は全て部分情報問題である。計算機科学の観点に立った場合、情報の部分性はさまざまな困難の源泉である。その困難とは、曖昧性 (ambiguity, vagueness)、文脈依存性 (context sensitivity) などであり、これらは自然言語に限らずAIの問題領域における困難のほとんどを占める。

以下ではまず、これらの困難に対処するためにプログラミング・パラダイムが満たすべきいつくかの一般的な要請について論じ、その中で、何らかの制約プログラミングの枠組が必要であることを指摘する。次に、特に自然言語の処理に関して制約の考え方を用いたさまざまな方法を、これらの要請に照らしながら紹介する。

2 制約プログラミング

AI、特に自然言語の問題領域における情報の部分性を計算モデルによって扱うためには、少なくとも以下の4つの要請を満たすプログラミング・パラダイムが必要である。

(I) 制約の内容と情報の流れとが互いに独立に記述される。
(II) 計算は制約の変換である。
(III) 組合せ的 (combinatorial) な構造を持つ対象を扱う。
(IV) 計算の制御はヒューリスティクス (heuristics) による。

要請 (I) は、制約の内容の記述の中に処理手順の明示的な指定を含めないということである。たとえばPrologの場合には、ホーン節の本体が左から右へと処理されることが決まっているから、この要請が満足されない。制約の内容と処理の手順を混ぜて指定する方法では、部分情報問題を扱うために実現すべき情報の多様な流れを実現できない、というのが、(I) を要請する理由である。ここで情報の流れとは、情報の表現のある部分を参照することによって他の部分の状態が変化することである。たとえば、変数Xの値が2であるという情報を参照し、Y:=X+1という代入によって変数Yの値を3とした場合には、XからYに情報が流れたことになる[1]。

問題を規定する制約の内容を記述する際にそれを処理する際の情報の流れをも同時に指定するプログラミング法を、手続き型プログラミングと呼ぼう。(これは一般に考えられている手続き型プログラミングと大体同じものである。) たとえばソーティングの手続きには、求めるべき構造に関する (与えられたいくつかの数値を要素とするリストであって、しかもある大小関係に関してそれらの要素が昇順に並んでいる、という) 制約と、その制約を満足する構造を得るための処理手順とが、渾然一体となって埋め込まれている。このようなプログラミングの方法が有効なのは、関連する情報のどの部分が参照可能であるかが決まっており (たとえばソーティングにおいては、普通は順序付けるべき値の集合が所与である)、従って、問題を解く際の情報の流れ方を狭く限定できる場合であり、計算機で扱われてきた問題の多くは、この条件を満たすように作られた完全情報問題である。

しかし、自然言語の処理を含む多くの部分情報問題においては、情報のどの部分がどの程度参照可能となるかが場合によって異なり、その場合の数は、少くとも、関係する制約全体の複雑さの指数関数である。そして、情報の流れは情報が「多い」所から「少ない」所へと向かうから、そのような問題を扱う際の情報の流れの多様性は制約全体の複雑さをはるかに凌ぐ。仮に、「水ちょうだい」という発話の理解を考えてみると、この発話がたとえば台所でなされたという情報が参照可能かどうか、話者が花瓶を持っているという情報が参照可能かどうか、あるいは発話の各部分に関する情報がどの程度参照可能か (たとえば、どの程度はっきり聞き取れたか) など、関連する情報の参照可能性について、潜在的には莫大な数の組合せがあり、それに応じて、この発話を解釈する際の情報処理は、多様な種類の制約のモジュールの間の多様な方向の情報の流れを含むことになる。

この場合に関係する制約のモジュールとしては、語彙、形態論的制約、統語論的制約、意味論的制約、語用論的制約、言語外的な制約があり、さらに、言語外的な制約は、水に関する知識や料理に関する知識などのモジュール群からなる。それ自身がこのように十分複雑な制約の複雑さを遥かに凌ぐ複雑さを持つ情報の流れを明示するプログラムは、実際問題として人間の手に負えないほど複雑なものとなろう。

しかもこの複雑さは、プログラムがモジュール構造を欠くという、たちの悪い複雑さである。プログラムにモジュール構造を持たせ、管理可能にするためには、制約のモジュール構造がプログラムのモジュール構造に反映されなければならない。しかし、手続き型プログラミングにおいてそれを行なうと、情報の流れが制約のモジュール構造に沿うように狭く限定されてしまい、かと言って多様な情報の流れを実現しようとすると、今度はプログラムのモジュール構造が崩れてしまう。たとえば、手続き型プログラミングを用いた文の理解を考えた場合、制約のモジュール構造をプログラムのモジュール構造に単純に反映させると、まず統語的な解析を行ない、その後で意味解析を行なうようなプログラムができる。しかし、たとえば意味的な情報を構文的な情報よりも優先的に用いるべき場合も実際には多いから、このようなプログラミングは情報の流れを不当に狭く限定していることになる。そこで、たとえば構文解析の途中で意味的な情報の一部を参照するというようなプログラムを作ることになるが、これはプログラムのモジュール構造を崩

[1] または、情報の表現 X=2 から Y=X+1 へ情報が流れて後者が Y=3 という形になったとも言える。

すことだから、この路線を推し進めて行くと、システムはじきに複雑過ぎて管理不能なものとなる。こうして、手続き型プログラミングによって本格的なAIシステムを開発しようという企ては全て破綻する。機械翻訳システムやエキスパートシステムなど、大きなAIシステムにおいて、処理の範囲を少し広げようとするとたちまちプログラムが複雑化し、拡張はおろか保守することすらできない代物になってしまう、という話をしばしば耳にするが、それはこのような事情による。

この複雑化の原因は、制約の内容と情報の流れとを混ぜて記述したことにある。従って、多くの部分情報問題の処理において、プログラムのモジュール構造を保証しつつ多様な情報の流れを実現するには、制約の内容と情報の流れとが互いから独立に記述されなければならない。以上をまとめると、どのように情報を流せばよいかを予め限定できる場合には手続き型プログラミングによって制約の各部の内容に対する情報の流れを時前に指定しておくことができて、その方が実行の効率がよいが、自然言語処理のよううに情報の流れを時前に限定できない場合には、制約の記述には情報の流れの指定が含まれないようにしておき、実際に処理するときに情報の流れを動的に決定せざるを得ない。

こうして要請(I)が導かれたが、そこから直ちに要請(II)が得られる。即ち、(I)により、いわゆる知識や信念などの、世界に関する情報が常に制約の形で記憶されるから、システムに内在化された情報の処理とは、新しい局面により効率的に適応できるような形に制約を変換することである。情報の部分性により、対象の構造を一般には一意に決定できない、ということも考えると、部分情報問題において処理されるべきものは制約であって対象ではない。情報処理はその次の場面においてより適切に行動するためになされるものだから、そこで第一義的に重要なのは、対象の構造ではなく、(思考を含む)行動に課せられるべき制約である。対象の構造は制約の一部と見なせる、と言う意味においても、構造は副次的なものに過ぎない。むしろ、制約のうちで構造を表現する部分とそうでない部分の間に明確な区別を設ける必要がそもそもないと考えるべきであろう。

また、要請(I)から、(II)で言う処理過程は多様な情報の流れを実現しなければらない、ということも直ちに導かれる。つまり、(II)の処理過程が実現する情報の流れがある方向に偏ったものであれば、制約の記述の中に情報の流れが含まれることになってしまう。たとえば、Prologの処理過程を考えると、ホーン節の本体が常に右から左に処理される等の事情によって情報の流れに偏りが生ずるから、Prologのプログラム中には制約各部の内容とともにそれを処理する際の情報の流れも指定されてしまう。

要請(III)は、対象の担う情報への参照の容易さに関する要請である。まず、情報が部分的にしか得られない場合には、部分的な情報を容易に参照できることが望ましい。即ち、制約の構造は、部分を単に繋ぎ合せることによって全体が構成される、という意味において組合せ的(combinatorial)[2]であるべきである。組合せ的構造とは、最も一般的にはグラフであり、意味ネットワークやフレームなども組合せ的である。このような構造は、各部分に自明な仕方で言及することができるという意味で、部分情報問題の処理に適している。たとえば、$p(X) \wedge q(X)$のような論理式によって表現された制約は、$p(X)$および$q(X)$という部分を自明な方法で繋ぎ合せることによって構成されているという意味において組合せ的であり、これらの部分を簡単に参照できる。ところで、対象の構造もまた上述のように制約の一部と考えられるから、同様に組合せ的となる。即ち、ここで考えるべき対象は、一般にネットワークや木のようなものであって、たとえば実数や有理数

[2]「組合せ的」という言い方は、認知科学において心的表象(mental representation)の性質を論ずる場合[7]の標準的な用語法である。この言い方は、そのような構造がいわゆる組合せ問題(combinatorial problem)と深い関係を持つことにも通ずる。これに対し、合成的(compositional)という言い方を用いなかったのは、関数の合成(composition)などとの連想を避けるためである。

などに限定すべきではない[3]。特に自然言語を扱う場合にこのような意味において組合せ的な構造が必要となることは明らかであろう。しかし、そもそも言語が組合せ的な構造を持つのも、情報の部分性に対処するためと考えられる。

要請(IV)に言うヒューリスティクスは、情報の欠如を補うためのものである。処理の部分性として指摘したように、与えられた制約は完全には処理し切れないから部分的に処理する必要があるが、どの部分をどのような順序で処理すればよいかを決定するための情報は一般には得られない。従って、部分的処理の制御は、必ず正しいとは限らない判断を含む。即ち、処理の優先度に関するヒューリスティクスを用いることになる。さらに、そのようなヒューリスティクスがあれば、部分的処理を制御するのみならず、(暫定的な)結論を導くためにも用いることができる。即ち、2つの競合する結論のうちいずれを選ぶべきかが完全にはわからないけれどもとりあえずどちらかに決めておく、という際に用い得る。知識の各部分にたとえば確信度(certainty factor)のようなものが割り当てられているとすると、知識の中のいくつかの部分が連言(AND)で繋がっているときには確信度の小さな部分から優先的に処理し、選言(OR)で繋がっているときには確信度の大きな部分から優先的に処理すればよい。競合するいくつかの結論から暫定的に1つを選ぶ際にも、確信度の大きなものを選べばよい。(ただし、後に述べるように、処理手順に関するヒューリスティクスを決定するのは確信度のようなものだけではない。)

ヒューリスティクスは、制約の内容と別に与えられるのではなく、制約から導かれるものである。制約は論理によって記号的に分節される側面、つまり組合せ的な構造であるネットワークと、そうでない側面(たとえば、このネットワーク構造上のエネルギーの分布など)とからなり、ヒューリスティクスはこれらの組合せから生ずると考えるべきであろう。また、制約にこのような記号的でない側面を考えざるを得ないのは、制約が完全に記号的だとすると、それは一般には完全に処理し尽くせず、従って、部分的に処理するためのヒューリスティクスが必要になるからである。記号的な制約を処理するためのヒューリスティックな規則をやはり完全に記号的に分節できるような形で与えようとすると、その規則に関しても部分的な処理の必要が生じるから、さらにそのためのヒューリスティクスが要ることになり、無限後退に陥る。これは、そもそもシステムの仕様記述が完結しないという問題であり、処理が完結しないという問題とは異なることに注意されたい。

これに関連して注意すべきは、ヒューリスティクスは制約の一部なのだから、その多くの部分が問題領域に依存する、ということである。従って、ヒューリスティクスを領域知識の一部としてプログラムすることができなければならない。たとえばSimonらの一般問題解決器(GPS; General Problem Solver)は、手段-目的解析(means-ends analysis)のような、領域に依存しない一般的なヒューリスティクスを用いていたが、これでは不十分であり、GPSは非常に限定された簡単な問題にしか適用することができなかった。この失敗を繰り返さないためには、領域知識の一部としてのヒューリスティクスを問題解決過程に反映させられる必要がある。

以上の要請、特に(I)と(II)とは、**制約プログラミング**(constraint programming)に通ずるものである。しかし、これまで行なわれて来た制約プログラミングの研究[5, 6, 8, 19, 23]は、要請(III)、(IV)を満足しようとするものではない。(III)に関して言うと、現在、制約プログラミングの主要なアプローチにおいて要請(I)と(II)を満たすような形で扱っている対象は、数値やブール値、あるいは有限領域内の対象である。また、制約プログラミングに関する従来の研究は、ヒューリスティクスを本格的に論じてはいなかった。これは、組合せ的な対象を扱っていないことにも関係している。即ち、組合せ的な対象に関する制約充足問題の多くは原理的に計算不可能なものであるから、そうした問題を実際的に扱うにはヒューリスティクスが不可欠となる。

[3]無論、数値のような対象も組合せ的な構造として表現することにより対象の一種とすることが可能と考えられる。

3 Prologとその拡張

この節では、論理プログラミングにおける組合せ的構造に関する制約の扱いについてのこれまでの主な研究の足跡を辿ってみよう。論理プログラムと言えばまずは何と言ってもPrologである。PrologはHerbrand領域における制約を処理できる、ということに一応なっているようであるが、実はPrologのインタープリタはプログラムを制約としてではなく、むしろ手続きとして扱っている。即ち、その実行順序に対して予め過大な制限を課しているのである。まずその欠点を指摘した後、これを克服しようという研究の流れを振り返って見る。

たとえば、下のようなプログラムは、Prologのプログラムの一部としてしばしば現われる。(以下、DEC-10 Prologの記法を用いることにする。大文字で始まる名前は変数を表わす。)

(1) `:- p(X, Y), q(Y, Z).`

この手のプログラムは、普通はXを入力、Zを出力として用いられる。逆にZを入力、Xを出力としてこれを使おうとすると、やたら効率が悪いことが多く、下手をすると無限ループに陥る。即ち、X(およびY)の値が定まっていない状態でp(X,Y)が起動されると、p(X,Y)の解が多いため、q(Y,Z)と両立する解を見付けるのに手間取ることが多い。つまり、p(X,Y)に後戻り(backtrack)する回数が多くなり、曖昧性を処理する際の効率が悪くなることがしばしばである。

要するに、Prologの実行方式は、特定の方向の情報の流し方に際してしか、効率よく働かないのである。では、どうすればよいか? 上の例から得られる教訓は、次のようなものである。

(2) 具体的な入力が与えられた箇所に対して処理を行なえ。

プログラム(1)に即して言えば、Xの値が所与の場合はp(X,Y)から、Zの値が所与の場合はq(Y,Z)から先に評価せよ、というわけである。この原理は、今更言うまでもなくお馴染みのものであり、データ・フロー計算機や、関数型言語等における遅延評価(lazy evaluation)などに関して、既にしばしば論じられている。

論理型プログラミング言語においても、(2)に基づく試みがいくつかある。Prologのfreeze[4]などがそれである。freezeは特殊な組込述語であり、たとえば、`freeze(X,p(X))`を実行すると、変数Xに制約p(X)が課されることになる。この制約は、Xが具体的な値に束縛された所で、あたかもプログラム中のその箇所に挿入されたかのように振舞い、実行される。

この方法にはいくつかの問題があるが、評価されるべき制約が評価されないことがある、というのが一番まずい。即ち、同一の変数に2つの制約が課された場合、制約は評価されないが、2つの制約が互いに矛盾したり、あるいはそれらの制約からその変数の値が定まるはずであったりするかも知れないから、実は評価すべきなのだ。たとえば、`freeze(X,member(X,[a,b]))`と`freeze(X,member(X,[c,d]))`とが実行された場合、いずれの制約も評価されず、矛盾が看過されてしまう。また、`freeze(X,member(X,[a,b]))`と`freeze(X,member(X,[b,c]))`とを実行しても、制約が評価されないから、Xの値が正しくbに定まらない。

要するに、変数の値が決まったときにだけ制約が評価されるような方式においては、デッドロック(deadlock)の可能性がある、ということである。入出力の流れが巡回路を含まない手続き的プログラムでは、このような問題は生じない。しかし、入出力関係が厳格には定まっていないプログラミング言語における遅延評価は、デッドロックの原因となりがちである。

以上の教訓に基づき、宣言的プログラミングに関して(2)を定式化すると:

(3) ある変数に2つの異なる制約が課せられている場合にその制約を評価せよ。

これは、Horn節論理プログラミングで特に閉世界仮説を用いている場合には非常によい指針を与える。ここで、要素式の引数は全て変数と考える。従って、たとえばp(a)という要素式はp(X)という要素式とX=aという束縛との連言と考える。また、変数に課せられる「制約」とは、あるリテラルのある引数位置に現われるということか、あるいはその変数が具体的な値に束縛されて

いるということである。この意味において変数に2つ以上の異なる制約が課せられているとき、それらの制約の間に依存関係 (dependency) があるということにする。(3) は、たとえば、変数 X に p(X) という制約と q(X) という制約が同時に課された場合とか、p(X) という制約と X=f(X) という制約が同時に課された場合において制約を評価せよ、ということである。

まさしくこのような方略によってデッドロックを回避しているのが、項記述 (term description) [18] である。項記述は、freeze と同じく変数に制約を付加するものだが、同一の変数に2重の制約が課された場合にも制約を評価するので、デッドロックは起こらない。ところが、freeze の場合と同様、実行される制約が、プログラム中のその時に実行中だった部分に手続として固定されてしまうので、次のような別の不都合が生ずる。たとえば、変数 X に制約 member(X,[a,b,c]) が課されているとき、さらに member(X,[b,c,d]) という制約が加わると、これらの制約はそこで評価されて、とりあえず X の値は b ということになるから、その後しばらくして、たとえば member(X,[a,c,e]) を実行すると、失敗して後戻りを起こし、埋め込まれた制約を再び評価しに行くことになる。この時の無駄な処理に要する計算量は、その後戻りの距離の指数関数である。

制約単一化 (constraint unification) [11, 12, 25] は、このような難点を持たない。制約単一化とは2つの制約付き変数を単一化して1つの制約付き変数を得るという操作である。即ち、変数に課された制約を処理する際に上記のような意味においてその制約をプログラムに埋め込むのではなく、制約は常に変数に付随するものとして扱う。処理された後の制約もやはり制約として変数に課されるわけであるから、ここで言う制約の処理とは制約の変換である。この変換は、展開 / 畳込み (unfold/fold) 変換 [24] の一種と見なすことができる。

この変換は、上の (3) に従って行なわれる。たとえば、制約 member(X, Y) と制約 Y=[a, b, c] とが同時に存在した場合には、前者は単一の制約に変換されて、新たな制約 abc(X) となる。述語 member と ab の定義はそれぞれ (4) および (5) である。

(4)
```
memb(A, [A | _]).
memb(A, [_ | S]) :- memb(A, S).
```
(5)
```
abc(a).
abc(b).
abc(c).
```

ここで変数 Y に関する依存関係が解消されていることに注意されたい。さらに、制約 bcd(X) が生じたとすると、これは abc(X) と合わさって新たな制約 bc(X) に変換される。bcd と bc の定義はそれぞれ (6) および (7) である。

(6)
```
bcd(b).
bcd(c).
bcd(d).
```
(7)
```
bc(b).
bc(c).
```

無論、変換によって新たに作られる述語が再帰的に定義されることもある。

これらの例のように、ホーン節論理で閉世界仮説を用いた場合、依存関係があるということは、制約が充足不能かも知れないということであり、逆に、制約の中に現われる全ての述語が本体に依存関係のない定義節を持つならば、先頭節の否定は (無限木を許せば) 充足可能である。従って、変数に関する依存関係を解消するというのは、非常によい処理のヒューリスティックである。

たとえば以下のような文の解析は、制約単一化によってうまく扱うことができる。

(8) the idea that I believe that you hate $\left\{ \begin{array}{l} \epsilon \\ \text{me} \end{array} \right.$

この場合、thatが関係代名詞であるか接続詞であるかはもっと右の方を見ないとわからないから、thatを続んだ時点でいずれであるかを決定しないことが望ましい。制約単一化を用いれば、この決定の遅延を自動的に行なうことができる。しかし、このthatが関係代名詞か接続詞だという制約と依存関係を持つ制約(たとえば、「関係代名詞に導かれる節の中にはその関係代名詞に対応する空所(gap)がある」という制約)がそれ以前にも現われるから、freezeや項記述のような方法だとうまく遅延ができない。

制約単一化は先に述べた要請(I) ~ (IV)をある程度満たしているが、完全に満たすわけではない。特に、閉世界仮説の下でのHorn節論理に限定したことは情報の流れを予め制限することになっており、(I)を十分に満たしているとは言えない。統語的制約の処理はそれでもかなりの程度まで可能かも知れないが、意味的制約の処理においてはより一般的な否定の扱い等が必要となり、この限定は不適切であろう。また、意味的制約の処理とか、構文解析そのものを制約の変換として行なうとかいうことを考えた場合、単に依存関係を解消するというヒューリスティクスよりもキメの細かいヒューリスティクスが必要となろう。

4 制約の個別的処理

上の節で述べた方法は、制約の種類にさほど依存しないものであったが、ここでは、制約の各部分の内容に依存した処理方法について紹介する。

文を解釈する際、まず統語的制約の処理を行なっていくつかの構造を得た後に、各々の構造を意味的に検査することによって妥当な解釈を絞り込む、という方法は、統語的制約を処理した段階で得られる構造が莫大な個数に上ることが多く、効率が悪い。従って、意味的な処理をもっと早い段階で適用する必要がある。この考え方に沿って、文全体に関する統語的処理が終了する前に意味的な処理を行なう方法を論じた研究がいくつかある。

WinogradのSHRDLU [29]は、名詞句を読み終えたら直ちにその指示対象を同定しようとする。たとえば、

(9) Put the blue pyramid on the block in the box.

という文は統語的に曖昧であり、

(10) Put the blue pyramid on [the block in the box].
(11) Put [the blue pyramid on the block] in the box.

という2通りの統語構造を持ち得る。ここで、(10)が正しいのはブロックが箱の中にあるときであり、(11)が正しいのは青いピラミッドがブロックの上にあるときであるから、多くの場合、the block in the boxとかthe blue pyramid on the blockとかいう名詞句の指示対象を同定する処理を行なうことによって、この曖昧性は解消される。そして、この指示対象の同定は、SHRDLUが扱っている例題のような場合には、各名詞句を読み終えた直後に行なうことができる。

ところが一般には、指示対象の同定等に関する処理をもっと早く行なうべきこともあるし、逆に遅らせるべきこともある。たとえば、

(12) the beautiful girl sitting beside the bald guy

のように名詞がいくつかの修飾句を受けている場合、最初の方の修飾句の情報を使わなくても主名詞の指示対象を同定できることがあり、そのような場合には指示対象を早期に決定してしまうことが望ましい。(12)の場合には、the beautiful girlの指示対象を決定することが、the bald guyの指示対象を決定する上で役立つだろう。一方、

(13) The teacher is silly in my class.

の the teacher のように、指示対象を決定するために名詞句の外の情報を参照しなければならない場合もある。

SHRDLU の意味処理部は手続き型言語である PLANNER を用いている。PLANNER の実行方式は、トップダウンに処理を行ない、行き詰まったらバックトラックするというものであり、少くとも Prolog と同程度に実行の制御に関する柔軟性が低い。即ち、名詞句の指示対象を処理するための PLANNER の手続きをその名詞句を読み終えた瞬間に起動すると、指示対象の決定は名詞句を読んだ直後に限られることになる。従って、上記のような多様な文脈に合うように指示対象の決定を早めたり遅らせたりすることができない。

Mellish [17] は、このような問題を解決するため、指示対象の同定を制約充足問題として扱い、いつ同定するかを柔軟に制御している。定名詞句と代名詞の指示対象の同定のために Mellish が用いた方法は、簡単に言えば、ある名詞句の指示対象の候補の集合と指示対象が満たすべき制約の集合とを常に記憶しておくというものである。the girl のような単数名詞や the two guys のような数詞付きの複数名詞のように、指示対象の個数が指定されている名詞句の場合には、指示対象の候補の集合の濃度がこの指示対象の個数に等しくなれば、指示対象が決定されたことになる。たとえば、

(14) the string hanging over the pully

という表現を考える。ここで、the string の指示対象の候補の集合が S={s1, s2, s3} であったとする。これは、string(X) を満足するような X の集合として求められる。また、同様にして the pully の指示対象の候補の集合が P={p1, p2} になったとする。このとき、hang_over(X, Y) となる X∈S と Y∈P の組が s2 と p1 のみであるとすれば、この制約 hang_over(X, Y) を加えることによって 2 つの名詞句の指示対象を決定することができる。また、hang_over(s3, p2) も満たされるとすれば、新たに S は {s2, s3} となり、指示対象の決定は後の処理に委ねられることになる。

ここで注意すべきは、SHRDLU の場合と異なり、各名詞句の指示対象をいつ決定するかが前以て定められてはいないという点である。指示対象の同定は、名詞句にまつわる情報が累進的 (incremental) に処理されて行くにつれて動的に制御される。

しかし、特定の問題に特定の処理手順を用いていることが、やはりこの方法の難点となっている。Mellish 自身も指摘しているように、彼のプログラムは、特に量化 (quantification) などの処理について一般性を欠く。たとえば、

(15) the man in each car

のような表現は、特定の 1 人の男が何台かの車に乗っていると解釈されてしまう。あるいは、

(16) John wants to marry a Swedish girl.

という文に関して、特定のスウェーデン娘がいないという解釈を得ることができない。

Mellish の議論の難点は、制約の内容に関する議論と制約の処理に関する議論が峻別されていない、即ち、先に述べた要請 (I) が認識されていないということである。制約の内容に関する研究と制約の処理法に関する研究は、一応分離すべきである。これらを分離すれば、(15) や (16) のような場合が扱えないというのは、処理手続きの問題と言うよりは、制約の内容が明らかになっていないという問題であることがわかる。制約の内容の多く (大部分ではないが) は言語学の研究成果として得られているから、その制約を記述する形式的体系 (たとえば 1 階述語論理) に関する一般的な処理手続きを与えればよいということになる。無論、そのような一般的な手続きを求めるのは簡単なことではないから、こう考えたからと言って直ちに問題が解けるわけではない。しかし、定名詞句、代名詞、不定名詞句 (indefinite noun phrase)、量化などについて、そ

れぞれを処理するためのデータ構造や処理手続きを個別的に求めようとするアプローチよりは見通しがよい。一般的な処理手続きに基づく方法については、次節で述べる。

統語的制約に関して特定の制約充足手続きを適用した研究もいくつかある。杉村ら [22] は、構文解析に用いる表の各々のスロットに入るべき項目を命題と考えてその表の構造に関する制約を命題論理によって表現し、その処理を制約論理プログラミング言語によって行なうという方法を提案している。また、丸山 [16] もやはり統語的制約の処理を論じており、その方法は、統語構造に関する制約をグラフとして表現し、構文解析をこのグラフに関する整合ラベリング問題 (consistent labeling problem) [26, 15] として扱うというものである。

5 一般的な方法

ここでは、制約の内容によらない一般的な方法を論ずる。以下で述べる方法は、いずれも1階述語論理における導出 (resolution) に基づく一般的な定理証明法を用いるものである。1階論理を用いるのは、それが組合せ的な対象の領域における制約を一般に記述するための最も簡単な体系だからであり、これを何らかの形で補うことによって、自然言語における統語的制約や意味的制約を統一的に記述することができる。主な課題のひとつは、情報の部分性に由来する不確実性などにうまく対処するような処理手続きを求めることである。また、一般的な定理証明法は効率が悪いと考えれらているが、これは、適切な実行制御を欠いていたためである。従って、多様な情報の流れを保証しつつ組合せ的爆発を起こさないような柔軟な処理のヒューリスティクスを与える必要もある。

Hobbs ら [14, 20] は、文章の理解を、**アブダクション** (abduction)[4] によって「最良の説明」を与える作業として捉え、その処理過程を1階論理の導出法を用いて定式化している。説明されるべき命題は、Prolog のゴールの形、即ち、存在量化されたリテラルの連言であり、推論は、その中の各リテラルに対して節形式 (clausal form) の公理を適用して逆向き推論 (アブダクション) を行なったり、2つのリテラルを因子化 (factoring) したりすることによってなされる。説明とは、こうして得られたリテラルの連言である。各リテラルには**仮説コスト** (assumption cost) と呼ばれるスカラー量が割当てられており、さらに、説明の仮説コストはその中のリテラルの仮説コストの和として定義される。仮説コストが小さいほどその説明は良い説明だとされる。仮説コストの大きさは、制約 (説明) の充足可能性の低さに対応している。

たとえば、以下のような談話における代名詞の照応を考えよう。

(17) I saw my doctor last week.
He told me to get more exercise.

この談話において説明すべき命題は、次のようなリテラルを含む。

(18) `doctor(D)`
`he(H)`

代名詞 he から得られたリテラル `he(H)` の仮説コストが高いとする。代名詞が自分自身では新たな指示対象を導入せず、既に導入されている対象を指示する、という現象を、これによって以下のように処理することができる。ここでは、次のような公理を用いる。

(19) $\forall$X `doctor(X)` $\rightarrow$ `person(X)`

(20) $\forall$X `person(X)` $\wedge$ `male(X)` $\rightarrow$ `he(X)`

仮説コストの高いリテラル `he(H)` に対してこれらの公理を用いてアブダクションを行なうと、`he(H)` から下のリテラルの連言が得られる。

[4] 事実 p を説明するために、公理 q→p を用いて仮説 q を導くような逆向き推論 (backward chaining)。

(21) doctor(H)
male(H)

このとき、he(H) の持っていた仮説コストがこれらのリテラルにほぼ均等に分配されたとすると、これらのリテラルは通常よりも高い仮説コストを持つことになる。そこで、doctor(H) と doctor(D) を因子化すると、それによって得られたリテラルの仮説コストはもともと仮説コストの小さかった doctor(D) のそれに一致し、こうして説明全体として仮説コストが下がる。このとき、male(D) が新たな仮説として立てられたことになる。

アブダクションに伴う仮説コストの分配の仕方は、次のようにして指定される。一般に、公理は下のような形を持つ。(全称量化子は省略する。)

(22) $P_1^{w_1} \wedge P_2^{w_2} \wedge \cdots \wedge P_n^{w_n} \rightarrow Q$

この公理を用いて Q から $P_1 \cdots P_n$ を導いたとき、Q の仮説コストが c ならば、P_i の仮説コストは $w_i c$ となる。(変数の単一化は省略する。)従って、もしも $\sum_{i=1}^{n} w_i < 1$ ならば、この公理を適用することによって必ず仮説コストが下がるから、より詳しい説明が優先される。逆に $\sum_{i=1}^{n} w_i > 1$ ならば、より抽象的な説明が優先される。ただし、この場合でもアブダクションによって導かれたリテラルが他のリテラルと因子化して仮説コストが下がることもあるから、この公理を用いた説明が全く採用されないというわけではない。

上の例では語用論的な処理を扱っているが、統語論等の制約も1階論理と仮設コストを用いて恐らく同程度にうまく表現可能であり、基本的にはこの方法によって自然言語に関するさまざまな種類の制約を統合的に扱うことができると考えられる。さらには、文の理解のみならず、生成にも同様に適用可能であろう。

仮説コスト(一般には説明の間の順序付け)は、情報の部分性から生ずる曖昧性に対処するために必要であり、やはり情報の部分性によって要請される制約プログラミングとは密接な関係にある。しかし、仮説コストは、いつくかの説明が陽に与えられたときにそのうちのどれがどれほど望ましいかを示すものであって、説明の集合が明示的に求まる以前に処理をどのような順序に従って行なえばよいかは、仮説コスト自身から直ちにわかることではない。1つの説明の中では仮説コストの大きなリテラルから処理すればよいとか、複数個の説明の内では仮説コストの小さなものから処理すればよいとかいうことも考えられるが、処理の優先度は仮説コストによって完全に決まるわけではない。先に述べたように、仮説コストは制約の充足不能性に対応するものだが、一般に、充足可能性の程度と処理の優先度とは一応は別のことである。たとえば、仮説コストが極めて高く、従って充足可能性が非常に低いにもかかわらず、充足されようがされまいがどうでもよく、従って処理しなくてもよいような制約の部分もあり有る。

直観的に言えば、処理はより重要な部分から行なわれるべきであり、この意味において、処理の優先度は、関連性(relevance)、重要度(importance)、さらには注意(attention)や連想(association)等の概念と関係している。関連性や重要度を計算的に求める手法として、知識表現のネットワーク上の活性拡散(spreading activation) [27, 28][5]がしばしば用いられる。活性拡散により、ネットワークの中で活性が高い節点に近い節点や、他の多くの節点と繋がっている節点の活性が高くなる。活性を重要性と見なせば、これは、重要なものに関係しているものや、多くのものに関係しているものは重要だという直観に対応する [9]。

制約の表現をネットワークと見なすことにより、制約プログラミングにおいても同様の手法が適用可能である。制約のネットワークにおける節点とは変数や述語や節であり、リンクとはそれらの間の参照関係である。このネットワーク上で活性拡散を行なった結果、活性の高くなった部分から優先的に処理を行なえばよい。たとえば仮説コストがリテラル活性を高め、節の活性を弱めると考えることにより、制約の充足不能性を活性度には反映させることが考えられる。制約

[5]その記号的近似としてマーカ伝達(marker passing) [3] があるが、以下ではそれも含めて活性拡散と言う。

の充足不能性の高い部分を処理することが重要なのは、それによって制約の中に含まれている選択肢のいくつかが除去され、より特定的な制約の表現が得られる可能性があるからである。

情報の部分性に対処する上での活性拡散の利点は、ネットワークの構造に沿った多様な方向の情報の流れを簡単に実現できる点にある。たとえば、上の例で (19) のような公理が適用できる場面で適用可能な公理は非常に多いから、そのような公理を全て調べる[6]ことは実際には不適当であるが、この公理を適用すべきだということを適用の以前に仮説コストから予測することはできない。しかし、この公理は述語 doctor を介して既存の仮説とリンクされているから、活性拡散を用いればこの公理が活性化され、この予測を行なうことができる。

依存伝播 (dependency propagation) [13] は、一階論理を用いて仮説コストを用いてアブダクションを行なうという意味においては Hobbs らの方法と同様であるが、活性拡散による実行制御をも行なっている。また、依存伝播は前述の制約単一化の拡張 [10] に相当するものであり、計算は制約の変換であるという意味でも Hobbs らの方法とは異なる。以下では、依存伝播による談話処理に関する長めの例を紹介する。これは、ICOT で開発中の DUALS という談話理解実験システムの第3版である DUALS-III [21] による質問応答の例である。DUALS-III は、小学校6年生の国語の教科書の文章を読んで「理解」し、その内容に関する質問に答えるシステムであり、その問題解決モジュールにおいて依存伝播を用いている。

DUALS-III は初め、下のような要素制約を持っている。(以下、特に明示しない限り、これらの要素制約は、全て説明すべきリテラルの連言の中に現われる。)

(23) `Sit=ground`

(24) `hold(Sit,earth)`

(25) `listen(Objs0)`

こうした意味構造の表現は、状況理論 [2] および状況意味論 [1] に基づいている。Sit は現実に対応する状況 (situation) であり、この状況に、earth (「地球」の指示対象) などの個物 (individual) が存在し、事実 (facts) が成立する。Sit が ground に束縛してあるのは、他の対象と無闇に単一化させないためである。対象 ϕ が状況 ζ に存在することも、事実 ϕ が状況 ζ で成立することも、共に要素制約 hold(ζ,ϕ) で表現する (このように「もの」と「こと」を統一的に扱っている点は元祖状況理論と異なる)。このように、状況の内部構造はその状況に対する制約として表現される。また、この段階において Sit で成立している事実は、DUALS-III の持つ事前の知識を構成するものであるが、ここでは省略した。

DUALS-III では、会話の進行も制約の変換と見なされる。要素制約 (25) は、DUALS-III が、入力文を受け取ることによって外界の情報を取り込もうとしている、ということを表現している。ここで Objs0 は、その入力文中に含まれる照応表現の指示対象となり得る対象や事実を要素とする集合である。ただしこの Objs0 は、会話の開始時のものなので空とする。一般に、listen(ξ) の形の要素制約を処理することによって、入力文 s が読み込まれて解析されると、この要素制約はその文の内容に相当する制約やその他の制約に変換される。その結果の制約の中には、listen(η) または speak(ψ,η) の形の要素制約が含まれており、η は ξ の要素 (の一部) の他に、文 s によって導入された対象や事実を要素とする。ここで、speak(ψ,η) は、DUALS-III が ψ という内容にあたる発話を行なう、という制約である。

まず DUALS-III は、「人間が、この地球の上で生き続けていくためには、どうしても、自然の恵みに頼らなければならない。」という文を読み込む。その処理は、上の要素制約 listen(Objs0) の活性が、他の要素制約のそれに比べて十分高いことによって生ずる。述語 listen は手続きとして定義されており、これを処理するということは、文解析モジュールを呼び出すということに

[6] Hobbs が用いている Stickel のシステムでは、探索の深さは制限しているが、幅は制約していないので、そのような全解探索を行なっている。

相当する。この処理によって、listen(Objs0) が次のような要素制約に変換される。(他にも要素制約が生成されるが、全部は示さない。以下同様に省略する。)

(26) hold(Sit,Purpose)
(27) hold(Sit1,Continue)
(28) presuppose(Sit1,Sit3)
(29) hold(Sit3,Loc1)
(30) hold(Sit3,Live)
(31) Purpose=infon(purpose,[Sit1,Sit2],yes)
(32) Continue=infon(continue,[Sit3],yes)
(33) Loc1=infon(loc,[earth,Live],yes)
(34) Live=infon(live,[human],yes)
(35) Sit1=sit(Sit,Purpose,1)
(36) Sit2=sit(Sit,Purpose,2)
(37) Sit3=sit(Sit1,Continue,1)
(38) listen(Objs1)

Sit1、Sit2 および Sit3 は現実の状況よりも下位の状況である。Purpose、Live などは事実であり、何らかの状況において成立 (hold) している。これらの要素制約の間の関係を図 1 に示す。

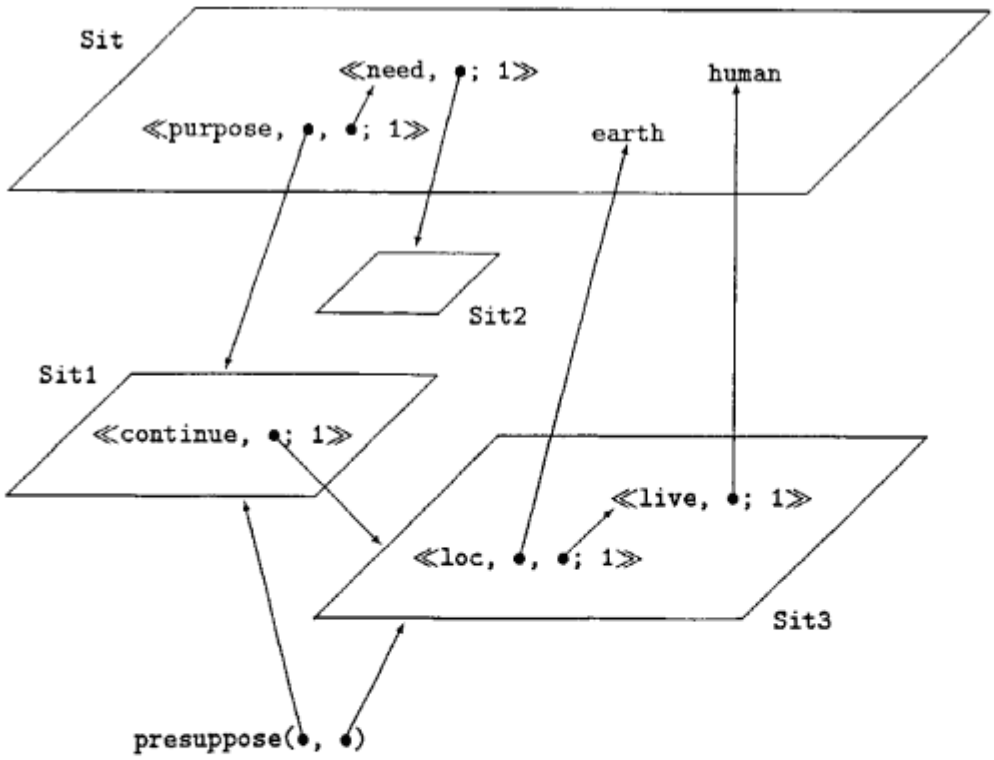

図 1: 「人間がこの地球の上で生き続けてゆくためには、…」に対応する意味構造

このとき、DUALS-III に対して「人間はどこで生きていますか。」という質問をすると、同様に (38) が処理され、下のような要素制約に変換される。

(39) hold(Sit,Loc2)

(40) hold(Sit,Live)

(41) Loc2=infon(loc,[Where,Live],yes)

述語 presuppose は、下のように定義される。

```
(42) presuppose(S1,S2) :- sub_sit(S2,S),project(S1,S).
     project(sit(S1,Infon,ArgPlace),S) :- project(S1,S).
     project(S,S).
```

ここで sub_sit(S2,S) は、S2 が S の部分状況であること、即ち、hold(S2,σ) なる全ての σ に関して、σ のコピー τ が hold(S,τ) を満たすことを意味する。ただし、σ が構造を持つ場合、その構造中に hold(S2,α) なる α が現われるならば、τ の構造の中の対応する位置には α のコピーが現われるものとする。(28) は、Sit3 が Sit1 または Sit の部分状況であるという意味である。この要素制約は、動詞「続ける」に付随する前提 (presupposition) であり、「続ける」の語彙項目から得られる。

一般に、平叙文の意味構造に関する制約は、仮説コストが低く、質問文の意味構造に関する制約は仮説コストが高い。たとえば、(26)~(37) の仮説コストは低く、(39)~(41) の仮説コストは高い。前述のように、仮説コストの高い要素制約を仮説コストの低い要素制約と因子化することによって、全体として仮説コストを下げることができる。

この例の場合には、(33) と (41) を因子化することができる。しかし、(39) と (40) を直ちに他の要素制約と因子化することはできない。たとえば、Sit と Sit3 とは因子化不可能なので、Loc1=Loc2 だからと言って (29) と (39) を因子化することはできない。ここで、(39) および (40) からのエネルギーの伝播が起こり、これらに繋がった様々な要素制約が活性化されていると考える。こうして、(28) の活性度が高くなっているとすると、(28) が展開され、sub_sit(Sit1,Sit3) が実行されて、以下のような要素制約ができる。

(43) hold(S,Loc1)

(44) hold(S,Live)

上述の通り、(28) は Sit3 が Sit または Sit1 の部分状況だということであるが、どちらであるかはそれだけではわからない。しかし、もしも Sit3 が Sit の部分状況であるとすれば、上の 2 つの要素制約をそれぞれ (39) および (40) と因子化することにより、仮説コストを下げることができる。一方、もし Sit3 が Sit1 の部分状況だとすれば、(39) と (40) の仮説コストを下げることはできない。従って、前者の方が全体として仮説コストが低くなるので、より望ましいことになり、前者が選択される。こうして、DUALS-III は「地球に生きている」という答を返すことができる。

この推論過程は、部分情報問題を扱う上で重要な意義を持ついくつかの側面を実現している。第 1 に、これは一種のデフォルト推論になっている。即ち、仮説コストをより低くする方の選択肢をデフォルトとして扱っていることになる。DUALS の現在の版が上の例題を処理する場合には、推論速度を上げるため、選択しなかった方の可能性は単に捨てているが、それを捨てずに記憶しておくこともできる。それによって、いわゆる非単調推論などを行なうことも可能である。デフォルト推論は不完全な情報によって判断を下す際に、非単調推論はそうして下した判断を棄却して信念を改変するのに必須である。

第 2 に、活性拡散によって、当面の問題に無関係な知識の部分が正しく無視されている点に注意されたい。上記の説明は無関係な知識には触れていないが、たとえテキストの文を大量に読

み込んで上記の質問には無関係な知識が大量に蓄えられた後でも、DUALSは、以前とほぼ同じ応答時間で同じ質問に答える。即ち、大量の無関係な知識を単に無視することができる。

第3に、上の例において**会話の含意** (conversational implicature) が処理されている。つまり、質問応答において、DUALS-IIIが質問者に情報を提供するのみならず、質問文が暗黙の内に含意している内容がDUALS-IIIの知識に逆に流れ込んでいる。その含意とは、DUALS-IIIが質問の答を知っている、というものであり、質問の内容に関する要素制約(39)~(41)がDUALSの信念に直接付加され、しかも仮説コストが高いという状態に対応する。即ち、この状態は、それらの要素制約を仮説コストの低い他の要素制約と因子化することによって仮説コストを下げるべきだということを意味し、これは、その質問の答えを導出し得るべきだという会話の含意と見なすことができる。

ここで重要なのは、情報の流れる方向が可能性としては限定されていないということである。この性質により、部分情報問題において生ずる多様な情報の流れを実現することができる。たとえば、質問文から情報を得ることを禁じてはいないということによって、上の会話の含意の処理が自然に生じている。このような処理は、従来の手続き型の方法論においては明示的なプログラミングを必要とする。これは、制約プログラミングが談話理解のような部分情報問題を扱う上で見通しのよい方法を与えることを例証している。

6 おわりに

部分情報問題の処理において実現すべき最も重要な処理の側面は、情報の流れの多様性である。自然言語処理のような部分情報問題においては、莫大な多様性を持つ文脈に応じて、実現されるべき情報の流れもまた莫大であり、予測不能であるから、そのような問題を扱う上で手続きプログラミングの方法は不適当であり、何らかの制約プログラミングの方法が必要となる。特に自然言語処理のような場合には、適当な文脈を与えればほとんどあらゆる情報の流れが生じ得るようにするため、一般の導出原理を含むようなるべく一般的な推論の方式と仮説コストの割当てや活性拡散のような方法を用いたヒューリスティックな実行制約が必要と考えられる。

情報の流れの多様性にいかに対処するかに関連するソフトウェア工学上の重要な問題で、本稿では扱うことができなかったものがいくつかある。中でもとりわけ重要な問題は、デバッグの方法に関するものであろう。情報の流れが予測不能であるような問題領域においては、たとえばプログラムの実行過程のトレースを見ても人間には理解し難いなどの事情により、従来のデバッグの方法がそのままでは使えない。これは制約プログラミングの問題と言うよりは、談話処理のような問題を扱う際の情報処理過程そのものの性質による一般的な問題である。

参考文献

[1] Barwise, J. and Perry, J. (1983) *Situations and Attitudes*, MIT Press.

[2] Barwise, J. and Etchemendy, J. (1987) *The Liar: An Essay on Truth and Circular Propositions*, MIT Press.

[3] Charniak, E. (1986) 'A Neat Theory of Marker Passing,' *Proceedings of AAAI'86.*

[4] Colmerauer, A. (1982) *Prolog II Reference Manual and Theoretical Model*, ERA CRNS 363, Groupe d'Intelligence Artificielle, Université de Marseille.

[5] Colmerauer, A. (1987) *An Introduction to Prolog III*, unpublished manuscript.

[6] Dincbas, M., Simonis, H. and Van Hentenryck, P. (1988) Solving a Cutting-Stock Problem in Constraint Logic Programming, *Proceedings of the 5th International Conference of Logic Programming*, pp. 42-58.

[7] Fodor, J., and Pylyshyn, Z. (1988) 'Connectionism and Cognitive Architecture: A Critical Analysis,' *Cognition*, Vol. 28, pp. 3-71.

[8] Jaffer, J., and Lassez, J (1987) 'From Unification to Constraints,' Furukawa, K., Tanaka, H., and Fujisaki, T. (*eds.*) *Logic Programming '87*, Lecture Notes in Computer Science 315, pp. 1-18.

[9] Hasida, K., Ishizaki, S., and Isahara, H. (1987) 'A Connectionist Approach to the Generation of Abstracts,' in Kempen, Gerard (ed.), *Natural Language Generation: New Results in Artificial Intelligence, Psychology, and Linguistics*, Dordrecht/Boston: Martinus Nijhoff Publishers (Kluwer Academic Publishers), pp. 149-156.

[10] Hasida, K. and Ishizaki, S. (1987) 'Dependency Propagation: A Unified Theory of Sentence Comprehension and Generation,' *Proceedings of the 10th IJCAI*, pp. 664-670.

[11] Hasida, K (1986) 'Conditioned Unification for Natural Language Analysis,' *Proceedings of the 11th COLING.*

[12] 橋田 浩一, 白井 英俊 (1986) 「条件付単一化」, コンピュータ・ソフトウェア, Vol. 3, pp. 28-38.

[13] 橋田 浩一 (1988) 「依存伝播」, 第29回プログラミング・シンポジウム論文集, pp.147-158.

[14] Hobbs, J., Stickel, M., Martin, P., and Edwards, D. (1988) 'Interpretation as Abduction,' *Proceedings of the 26th Annual Meeting of ACL*, pp. 95-103.

[15] Mackworth, A. (1977) 'Consistency in Networks of RElations,' *Artificial Intelligence*, Vol. 8, pp. 99-118.

[16] 丸山 宏 (1989) 「語間の係り受け関係の制約に基づく形式文法」, *Proceedings of the Logic Programming Conference '89*, ICOT.

[17] Mellish, C. (1985) *Computer Interpretation of Natural Language Descriptions*, Ellis Horwood. (邦訳: 田中 穂積 (1988) コンピュータのための自然言語意味理解の基礎, サイエンス社.)

[18] Nakashima, H. (1985) 'Term Description: A Simple Powerful Extension to Prolog Data Structures,' *Proceedings of the 9th IJCAI*, pp. 708-710.

[19] Sakai, K. and Sato, Y.: *Boolean Gröbner Bases*, ICOT Technical Memo No. 488, 1988.

[20] Stickel, M. (1988) 'A Prolog-like Inference System for Computing Minimum-Cost Abductive Explanations in Natural Lanuguage Interpretation,' *Proceedings of Internaltional Computer Conference '88 — Artificial Intelligence: Theory and Applications*, pp. 343-350.

[21] Sugimura, R., Hasida, K., Akasaska, K., Hatano, K., Kubo, Y., Okunishi, T., and Takizuka, T. (1988) 'A Software Environment for Research into Discourse Understanding Systems,' *Proceedings of the FGCS'88*, pp. 285-295.

[22] Sugimura, R. Miyoshi, H., and Mukai, K. (1988) 'Constraint Analysis on Japanese Modification,' in Dahl, V. and Saint-Disier, P. (*eds.*) *Natural Languge Understanding and Logic Programming, II*, Elsevier.

[23] Sussman, G. and Steele, G., Jr.: Constraints – A Language for Expressing Almost Hierarchical Descriptions, *Artificial Intelligence* (1980), Vol. 14.

[24] Tamaki, H. and Sato, T. (1984) 'Unfold/Fold Transformation of Logic Programs,' *Proceedings of the Second International Conference on Logic Programming*, pp. 127-138.

[25] Tuda, H., Hasida, K., and Sirai, H. (1989) 'JPSG Parser on Constraint Logic Programming,' *Proceedings of the European Chapter of ACL'89.*

[26] Waltz, D. (1975) 'Understanding Line Drawings of Scenes with Shadows,' in Winston,

P. (*ed.*) *The Psychology of Computer Vision*, McGraw-Hill. (邦訳: 白井 良明, 杉原 厚吉 (1979) コンピュータビジョンの心理, 産業図書)

[27] Waltz, D., and Pollack, J. (1985) 'Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation,' *Cognitive Science*, Vol. 9, pp. 51-74.

[28] Ward, N. (1988) 'Issues in Word Choice,' *Proceedings of the 12th COLING*, pp. 726-731.

[29] Winograd, T. (1972) *Understanding Natural Language*, Academic Press. (邦訳: 淵 一博, 田村 浩一郎, 白井 良明 (1976) 言語理解の構造, 産業図書.)