自然言語処理 83－12
(1991. 5. 17)

Constraint Projection －制約射影による構文解析－

中野幹生
NTT 基礎研究所
e-mail: nakano@atom.ntt.jp

単一化ベースの構文解析において処理時間を左右するのは、選言の扱いである。単一化の前に選言を展開してしまうと、選言の要素の数の指数関数の時間がかかってしまう。したがって、選言の展開を減らす単一化方法が有効である。本報告では論理的制約で表現された選言的素性構造の新しい単一化方法、制約射影を提案する。制約射影は、変数リストによって規定されたゴールに無関係な情報を捨てる機構を制約単一化に組み込んだものである。 さらに制約単一化と制約射影を実装し、計算時間を比較することによって、その効率性を確かめた。

Parsing with Constraint Projection

Mikio Nakano
NTT Basic Research Laboratories
3-9-11 Midori-cho, Musashino-shi, Tokyo 180 JAPAN
e-mail: nakano@atom.ntt.jp

Unification of disjunctive feature descriptions is important for efficient unification-based parsing. This paper presents constraint projection, a new method for unification of disjunctive feature structures represented by logical constraints. Constraint projection is a generalization of constraint unification, and is more efficient because constraint projection has a mechanism for abandoning information irrelevant to a goal specified by a list of variables.

1 はじめに

単一化は最近の計算言語学研究の中心的な演算である。自然言語解析や文法理論のたくさんの研究が単一化に基づいている。それは、単一化ベースのアプローチが、他の文法理論、計算理論に比べて、たくさんの利点を持っているからである。単一化ベースのフォーマリズムは文法記述を簡単にする。特に、規則や辞書の宣言的な記述を可能にする。また、変形を必要としない。

しかし、いくつかの問題点が残る。大きな問題の一つは、**選言的素性構造** (disjunctive feature structure) の単一化の非効率性である。FUG[8] は、辞書項目の経済的な表現のために、選言的素性構造を用いている。選言的素性構造は辞書項目の数を減らすことができる。しかし、もし選言的素性構造が DCG [9] や Kay のパーザ [8] のように，**選言標準形**[1] に展開されたとすると、単一化は選言の要素の数に関して、指数関数の時間がかかるであろう。効率的な選言的素性構造単一化には、選言の不必要な展開を避けることが重要である。Kasper [7] と Eisele and Dörre [3] はこの問題に取り組み、選言的素性構造単一化の方法を提案している。

これらの研究は項単一化 (term unification) ではなく、グラフ単一化 (graph unification) に基づいている。グラフ単一化は引数の数が自由で、引数がラベルで選択されるために、文法や辞書を記述するのが簡単であるという利点を持つ。しかし、グラフ単一化は二つの欠点を持つ。一つは、特定の素性を検索するのに余分な時間がかかること、もう一つは、コピーが多いことである。このような理由から、本稿では項単一化を採用する。

Eisele and Dörre [3] は彼らのアルゴリズムが、グラフ単一化だけでなく、項単一化にも適用可能であるといっているが、この方法はコピーがあまりいらないという項単一化の利点を生かしていない。逆に、選言的素性構造単一化の一方法である**制約単一化** (constraint unification) [5, 11] は項単一化の利点を生かしている。制約単一化では、選言的素性構造は論理的制約、特に、Horn 節で表現され、単一化は、制約充足問題とみなされる。さらに、制約充足問題を解くことは、制約を等価で充足可能な制約に変換することと等しい。制約単一化は、素性構造への制約を変換することによって、その単一化を行なう。制約単一化の基本的なアイディアは、制約を必要に応じて変換することである。すなわち、充足可能でないかもしれない制約のみを変換するのである。これが、制約単一化が効率的で、余分なコピーを必要としない理由である。

しかし、制約単一化には重大な問題点がある。不必要な情報を捨てる機構を持っていないために、項の引数の数が大きくなって、変換に時間がかかるようになる。従って、一般的な自然言語処理の観点からいえば、制約単一化は小さな構造を持つ制約を処理するのには適しているが、大きな構造を持つ制約を変換するのには適していない。

本稿は**制約射影** (constraint projection) という、もう一つの選言素性単一化方法を提案する。制約射影の基本的なアイディアはゴールに無関係な情報を捨てることである。例えば、ボトムアップ解析では、もし、文法が、最近の単一化文法のように局所的な制約のみからなっていれば、構文解析の適用の後で、子ノードに関する情報を捨て去ることが可能である。なぜなら、親ノードの素性構造は、子ノードの素性構造と句構造規則のみから計算可能であるからである。無関係な情報を捨て去ることによって、結果の構造を小さくなるので、次の句構造規則の適用が効率的になるであろう。制約射影という言葉は、入力された制約の特定の変数への射影を返すという意味で用いている。

2 論理的制約による選言的素性構造の表現

この節では Horn 節による選言的素性構造の表現について述べる。Horn 節の記述には DEC-10 Prolog の記法を用いる。

最初に、選言を含まない素性構造を論理項で表す。例えば、(1) は (2) で表される。

$$(1)\quad \begin{bmatrix} \text{pos} & \text{v} \\ \text{agr} & \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & \text{3rd} \end{bmatrix} \\ \text{subj} & \begin{bmatrix} \text{agr} & \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & \text{3rd} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

```
(2) cat(n,
        agr(3rd,sing),
        cat(_,agr(3rd,sing),_))
```

ファンクタ cat の引数は *pos* (part of speech)、*agr* (agreement)、*subj* (subject) 素性に対応する。

選言や共有はホーン節の本体を用いて表される。述語が複数の定義節を持つような本体の中の項は選言を表す。例えば、FUG[8] の記法を用いて書かれている選言的素性構造 (3) は (4) で表される。

$$(3)\quad \left\{ \begin{array}{l} \begin{bmatrix} \text{pos} & \text{v} \\ \text{agr} & \boxed{1} \left\{ \begin{array}{l} \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & \left\{ \begin{array}{l} \text{1st} \\ \text{2nd} \end{array} \right\} \end{bmatrix} \\ \begin{bmatrix} \text{num} & \text{plural} \end{bmatrix} \end{array} \right\} \\ \text{subj} & \begin{bmatrix} \text{agr} & \boxed{1} \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \text{pos} & \text{n} \\ \text{agr} & \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & \text{3rd} \end{bmatrix} \end{bmatrix} \end{array} \right\}$$

```
(4) p(cat(v,Agr,cat(_,Agr,_)))
      :- not_3s(Agr).
    p(cat(n,agr(sing,3rd),_)).
    not_3s(agr(sing,Per))
      :-1st_or_2nd(Per).
    not_3s(agr(plural, _)).
    1st_or_2nd(1st).
    1st_or_2nd(2nd).
```

[1] 選言標準形は $\phi_1 \vee \phi_2 \vee \phi_3 \vee \cdots \vee \phi_n$ という形をしている。ここで ϕ_i は選言を含まない。

ここで、述語 p は素性構造の規定に相当する。p(X) は変数 X が p で規定される選言的素性構造の候補の一つであることを意味する。FUG で用いられる *ANY* や規定されていない素性は無名変数'_'で表現される。

原始式はその中の変数への制約と考えることができる。(4) の中の 1st_or_2nd(Per) は Per を 1st か 2nd のどちらかでなくてはいけないと制約している。同様に not_3s(Agr) は Agr が agr(Num,Per) の形の項で Num が sing でかつ Per が制約 1st_or_2nd(Per) を満たすかもしくは、Num が plural であるということを意味する。

ここでは定義節を持たない述語は用いないし、考慮に入れない。制約としては意味を持たないからである。述語が定期節を持つ原始式を制約項と呼び、制約項の並びを制約と呼ぶことにする。(4) のような定義節の集合を制約の構造と呼ぶ。

句構造規則も論理的制約で表される。例えば、もし、句構造規則がバイナリ規則のみで L、R、M が左の子、右の子、親をそれぞれ表すとすると、これらは、我々が psr(L,R,M) と書き表す 3 項関係を満たす。psr のそれぞれの定義節が句構造規則に対応する。(5) は一例である。

```
(5) psr(Subj,
        cat(v,Agr,Subj),
        cat(s,Agr,_)).
```

psr の定義節には本体があっても良い。

もし、二つの選言的素性構造が制約項 p(X) と q(Y) で規定されているとすると、X と Y の単一化は (6) を満たす X を見つけることと等しい。

```
(6) [p(X),q(X)]
```

このように選言的素性構造の単一化は制約充足問題と等価である。句構造規則の適用も、制約充足問題と考えることができる。例えば、もし、左子カテゴリと右子カテゴリが c1(L) と c2(R) で規定されているとすると、親カテゴリを計算することは (7) を満たす M を見つけることと等価である。

```
(7) [c1(L),c2(R),psr(L,R,M)]
```

(8) のような Prolog の呼びだしによってこの制約充足を行なうことができる。

```
(8) :-c1(L),c2(R),psr(L,R,M),
      assert(c3(M)),fail.
```

しかし、この方法は効率が悪い。二つ以上の定義節が適用可能な時に、Prolog は一つの定義節を選んで処理を進めるので、同じ処理を何回も繰り返さなければならない。この方法は、単一化の前に選言を選言標準形に展開するのと同じである。

3 制約単一化とその問題点

この章では選言的単一化の一方法、制約単一化[2] [5, 4, 11] を説明し、その問題点を指摘する。

3.1 制約単一化の基本的なアイディア

1 節で述べたように、制約充足問題は、制約の変換によって、解くことができる。我々が求めているのは、結果の制約の充足可能性が保証されておりかつ選言の要素の数が少ないような、変換のアルゴリズムである。

制約単一化は選言の余分な展開を避ける制約変換システムである。制約単一化の目標は入力された制約をモジュラーな制約に変換することである。モジュラーな制約とは以下のように定義される。

(9) 制約がモジュラーであるとは、
  1. すべての制約項のすべての引数が変数
  2. どの変数も 2 箇所には現れない。
  3. すべての述語がモジュラー定義である。

述語がモジュラー定義であるとは、すべての定義節の本体がモジュラーであるか、*NIL* であることである。

例えば、もし、すべての述語がモジュラー定義であるとすると、(10) はモジュラーな制約であるが、(11), (12), (13) はモジュラーではない。

```
(10) [p(X,Y),q(Z,W)]
(11) [p(X,X)]
(12) [p(X,Y),q(Y,Z)]
(13) [p(f(a),g(Z))]
```

(10) は述語が定義節を持っているので充足可能である。証明は省略するが、モジュラーな制約は必ず充足可能である。制約をモジュラーに変換することは、制約を満たすインスタンスの集合を見つけることに他ならない。逆に、モジュラーでない制約は充足可能でないかも知れない。制約がモジュラーでない場合、依存関係があるという。例えば、(12) には Y に関する依存関係がある。

制約単一化のアイディアは、(a) 入力された制約の制約項を、変数を共有しないようないくつかのグループに分類してそれらを別々に処理する (b) モジュラーな制約は処理しない、の二つである。

簡単にいうと制約単一化は依存関係をもつ制約のみを処理する。これは選言の不必要な展開を避けることに相当する。制約単一化では処理の順序が依存関係に応じて決められる。この柔軟性によって制約単一化は処理量を減らすことができる。

制約単一化のこれらのアイディアを例を通して簡単に説明しよう。制約単一化は二つの関数 *modularize* (制約) と *integrate* (制約) からなる。制約単一化は *modularize* を呼ぶことによって実行できる。*modularize* は入力された制約をいくつかの制約にわけ、それらの *integrate* を返す。もし、ひとつで

[2]制約単一化は以前は条件付単一化と呼ばれていた。

も *integrate* が失敗したら、*modularize* も失敗する。*integrate* は入力された制約と等価な新しい制約項を作り、そのモジュラーな定義節を発見し、そして、その制約項を返す。*modularize* と *integrate* はお互いに呼びあって処理が進む。

(14) の実行を考えよう。

(14) *modularize*(
     [p(X,Y),q(Y,Z),p(A,B),r(A),r(C)])

述語は次のように定義されている。

(15) p(f(A),C):-r(A),r(C).
(16) p(a,b).
(17) q(a,b).
(18) q(b,a).
(19) r(a).
(20) r(b).

制約は (21), (22), (23) に分けられ別々に処理される。

(21) [p(X,Y),q(Y,Z)]
(22) [p(A,B),r(A)]
(23) [r(C)]

もし、制約が分割されないとすると、(21) が複数の解をもつ時には (22) の処理が何回も繰り返されるであろう。これが制約単一化が効率が良い理由の一つである。(23) はすでにモジュラーなので処理されない。Prolog だったら、r の定義節を見にいって、不必要な処理を行なうだろう。これが制約単一化が効率が良いもう一つの理由である。

(21) と (22) をモジュラーな制約項に変換するために (24) と (25) が呼ばれる。

(24) *integrate*([p(X,Y),q(Y,Z)])
(25) *integrate*([p(A,B),r(A)])

(24) と (25) は成功して c0(X,Y,Z)[3] と c1(A,B) をそれぞれ返すので、(14) は (26) を返す。

(26) [c0(X,Y,Z),c1(A,B),r(C)]

もし (24) か (25) が失敗すれば、この *modularize* は失敗する。

次に、*integrate* を (24) の実行を通して説明する。最初に新しい述語 c0 が作られ (27) のようにおかれる。

(27) c0(X,Y,Z)$\Longleftrightarrow$p(X,Y),q(Y,Z)

(27) は、もし [p(X,Y),q(Y,Z)] が充足可能であるならば、すなわち、c0(X,Y,Z) がモジュラーに定義され、c0(X,Y,Z) と p(X,Y),q(Y,Z) が X, Y, Z を同じように制約するならば、(24) が c0(X,Y,Z) を返すことを意味する。

次に、標的項が選ばれる。この選択には何らかのヒューリスティックスが適用可能であるが、ここでは単純に先頭の要素 p(X,Y) 選ぶ。すると、p の定義節が参照される。これは選言の展開に相当する。

最初に (15) が参照される。(15) の頭部が (27) の中の p(X,Y) と単一化されると、(27) は (28) になる。

(28) c0(f(A),C,Z)$\Longleftrightarrow$r(A),r(C),q(C,Z)

(28) の右辺では p(f(A),C) はその本体 r(A),r(C) におきかえられている。(28) はもし、各変数が (28) の右辺を充足すれば c0(f(A),C,Z) が真となることを意味する。

(28) の右辺はモジュラーではないので、(29) が呼ばれる。そして (30) を返す。

(29) *modularize*([r(A),r(C),q(C,Z)])
(30) [r(A),c2(C,Z)]

そうすると (31) が c0 の定義節として用いられる。

(31) c0(f(A),C,Z):-r(A),c2(C,Z).

次に (16) が参照される。すると、(28) は (32) になる。そして (33) が呼ばれ、これは (34) を返し、(35) が作られる。

(32) c0(a,b,Z)$\Longleftrightarrow$q(b,Z)
(33) *modularize*([q(b,Z)])
(34) [c3(Z)]
(35) c0(a,b,Z):-c3(Z).

結局 c0 の定義節ができたので (24) は c0(X,Y,Z) を返す。

(14) によって引き起こされた制約単一化によってできたすべての節は (36) のとおりである。

(36) c0(f(A),C,Z):-r(A),c2(C,Z).
     c0(a,b,Z):-c3(Z).
     c2(a,b).
     c2(b,a).
     c3(a).
     c1(a,b).

新しい節が作られた時、もし、その本体の中の項の述語が、定義節を一つしか持たなければ、その項はその定義節の頭部と単一化され、その本体と置き換えられる。この操作を帰着 (reduction) と呼ぶ。たとえば、(36) の 2 番めの節は (37) に帰着される。なぜなら、c3 は定義節を一つしか持たないからである。

(37) c0(a,b,a).

制約単一化は畳み込み (folding) と呼ばれるもう一つの演算を持っている。それは同じ種類の *integrate* の繰り返しを避けて、変換を効率的にする。畳み込みはまた member や append のように再帰的に定義された述語を扱うことを可能にする。

3.2 制約単一化を用いた構文解析

どんな構文解析アルゴリズムでもいいのだが、ここでは簡単のために CYK アルゴリズム [1] を用いる。

[3]新しくできる述語の名前として $cn(n = 0, 1, 2, \cdots)$ を使う。

制約項 cat_n_m(X) が X 入力文の $(n+1)$ 番目の単語から m 番目の単語までの句のカテゴリであることを意味するとする。すると、句構造規則の適用は (38) のようなホーン節を作ることに相当する。

(38) cat_n_m(M) :-
    modularize([cat_n_k(L),
        cat_k_m(R),
        psr(L,R,M)]).
    ($2 \leq m \leq l$, $0 \leq n \leq m-2$, $n+1 \leq k \leq m-1$,
    l は入力文の長さ)

新しく作られた節の本体は右辺の *modularize* が返す制約である。もし、*modularize* が失敗すれば、節は作られない。

3.3 制約単一化の問題点

制約単一化ベースのパーザの問題点は構文解析が進むにつれて制約項の引数の数が増えることである。例えば、cat_0_2(M) は (39) によって計算される。

(39) *modularize*([cat_0_1(L),
        cat_1_2(R),
        psr(L,R,M)])

これは [c0(L,R,M)] のような制約を返す。すると、(40) が作られる。

(40) cat_0_2(M):-c0(L,R,M).

次に、(40) が次のような規則の適用の中で使われることを考えよう。

(41) *modularize*([cat_0_2(M),
        cat_2_3(R1),
        psr(M,R1,M1)])

すると (42) が呼ばれる。

(42) *modularize*([c0(L,R,M),
        cat_2_3(R1),
        psr(M,R1,M1)])

これは c1(L,R,M,R1,M1) のような制約を返す。このようにして制約項の引数の数が増える。

これはつぎの 2 つの理由から計算時間の爆発を引き起こす。(a) 引数の増加が、新しい項、環境をつくったり、分割したり、単一化等の計算 時間を増加させること。(b) 結果の構造が、親カテゴリに無関係な素性の曖昧性のせいで、余分な選言を 含む可能性があること。

4 制約射影

この節では制約射影を説明する。制約射影は制約単一化の拡張で、前節で説明した欠点を克服している。

4.1 制約射影の基本的なアイディア

制約単一化に基づく構文解析の非効率性は子ノードの情報を持ち続けることにある。そのような情報は、もし親ノードの情報のみが必要であるという仮定の元では、捨て去ることが可能である。すなわち、(43) のような変換が、(44) よりも有用である。

(44) [c1(L),c2(R),psr(L,R,M)] $\Longrightarrow$ [c3(M)]
(43) [c1(L),c2(R),psr(L,R,M)]
    $\Longrightarrow$ [c3(L,R,M)]

(44) の c3(M) は充足可能で、かつ、左辺と M に関して等価でなくてはならない。また、c3(M) は M に関する情報のみを含むので、正規な制約でなくてはいけない。ここで、正規な制約を (45) のように定義する。

(45) 制約が正規とは
    (a) モジュラーであり、かつ
    (b) すべての定義節が正規定義節 (normal difinition clause) である。すなわち、本体には頭部に現れない変数は現れないような定義節である。

例えば (46) は正規定義節であるが (47) は違う。

(46) p(a,X):-r(X).
(47) q(X):-s(X,Y).

(44) のような演算は (48) で定義される新しい演算、制約射影と一般化される。

(48) ある制約 C とゴール (goal) と呼ばれる変数リストが与えられた時に、制約射影は、ゴールの中の変数に関して C と等価で、かつゴールの中の変数のみしか持たないような正規な制約を返す。

制約射影も制約単一化と同じように入力された制約を、依存関係に応じて、いくつかの制約に分割し、別々に処理する。分割された制約は二つのグループに分類される。一つはゴールの中の変数を持っている制約で、もう一つはその他の制約である。前者をゴール関連制約 (goal-relevant constraints) と呼び、後者をゴール無関連制約 (goal-irrelevant constraints) と呼ぶ。ゴール関連制約のみが正規な制約に変換される。ゴール無関連制約は充足可能性のみがチェックされる。ゴール無関連制約はもう使われることはなく、充足可能性のチェックは変換するより簡単だからである。これが制約射影が効率が良い理由である。

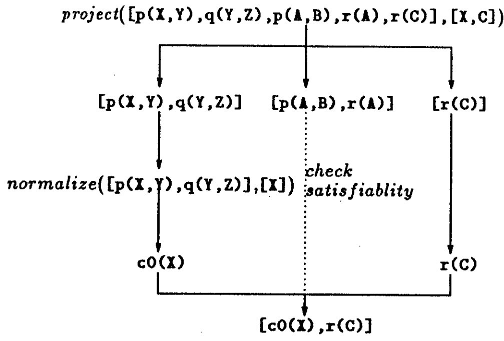

図 1: *project* の実行例

4.2 制約射影のアルゴリズム

制約射影は *project*(制約, ゴール) と *normalize*(制約, ゴール) の二つの関数からなる。これらはそれぞれ、制約単一化の *modularize* と *integrate* に相当する。*project* を呼び出すことによって、制約射影を実行できる。

(49) の実行を通して制約射影の処理方式を説明する。

(49) *project*(
  [p(X,Y),q(Y,Z),p(A,B),r(A),r(C)],
  [X,C])

述語は (15) から (20) と同じように定義されているとする。この実行を図 1に示す。最初に入力された制約は依存関係に応じて (50)、(51)、(52) に分割される。

(50) [p(X,Y),q(Y,Z)]
(51) [p(A,B),r(A)]
(52) [r(C)]

(50) と (52) はそれぞれ X と C を含むのでゴール関連制約である。(51) はゴール無関連制約なので充足可能性のみが調査される。もし、ゴール無関連制約の一つが充足不可能とわかったら、この射影は失敗する。制約 (52) は既に正規なので処理されない。そして (53) が (50) の変換のために呼ばれる。

(53) *normalize*([p(X,Y),q(Y,Z)], [X])

ここで第二引数すなわちゴールは (50) と (49) のゴールの両方に現れる変数のリストである。この *normalize* は [c0(X)] という形の制約を返すので、(49) は (54) を返す。

(54) [c0(X),r(C)]

これはゴールの中の変数のみを含んでいる。この制約の構造は (26) の構造よりも小さい。

次に、(53) の実行を通して *normalize* を説明する。この実行は図 2に示した。最初に新しい項 c0(X) が作られ (55) を意図する。

(55) c0(X)⟺p(X,Y),q(Y,Z)

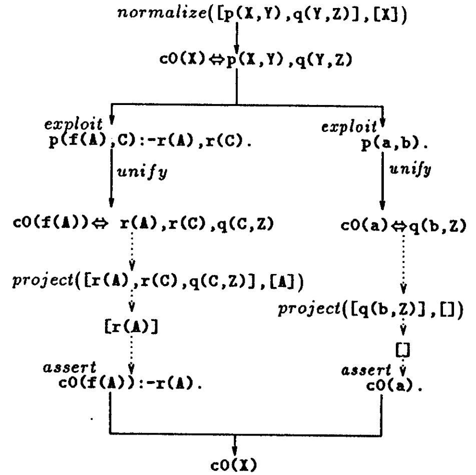

図 2: *normalize* の実行例

この新しい項はゴールの中のすべての引数である。*normalize* の仕事は c0 の正規な定義を見つけることである。標的項はゴールの中の変数を含んでいなければならないので p(X,Y) が選ばれる。p の定義節は (15) と (16) である。

(15) p(f(A),C):-r(A),r(C).
(16) p(a,b).

(15) が最初に利用される。(15) の頭部は (55) の中の p(X,Y) と単一化される。そうすると (55) は (56) となる。(もしこの単一化が失敗すれば次の定義節が参照される。)

(56) c0(f(A))⟺r(A),r(C),q(C,Z)

右辺は依存関係を持ち、また左辺に現れない変数を持っているので、正規ではない。したがって (57) が呼ばれる。これは r(A) を返し (58) が作られる。

(57) *project*([r(A),r(C),q(C,Z)], [A])

これは r(A) を返し、(58) が作られる。

(58) c0(f(A)):-r(A).

次に (16) が利用され同様にして (59) が作られる。

(59) c0(a).

c0 の定義節ができたので、結果として (55) は c0(X) を返す。

この制約射影によってできたすべての節を (60) に示す。

(60) c0(f(A)):-r(A).
  c0(a).

これを (36) と比べると、制約射影が制約単一化よりも効率的であるのみならず、記憶領域も必要としないことがわかる。

4.3 制約単一化による構文解析

制約射影に基づくCYKパーザを構築するには(61)のようにすれば良い。

```
(61) cat_n_m(M) :-
       project([cat_n_k(L),
                cat_k_m(R),
                psr(L,R,M)],
               [M]).
```

$(2 \leq m \leq l,\ 0 \leq n \leq m-2,\ n+1 \leq k \leq m-1,$

ここで l は入力文の単語数)

一例として、"Japanese work." という文の解析を次のような射影によって行なうことを考えよう。

```
(62) project([cat_of_japanese(L),
              cat_of_work(R),
              psr(L,R,M)],
             [M])
```

ここで、規則と辞書は次のように定義されているとする。

```
(63) psr(n(Num,Per),
         v(Num,Per,Tense),
         s(Tense)).
(64) cat_of_japanese(n(Num,third)).
(65) cat_of_work(v(Num,Per,present))
       :-not_3s(Num,Per).
(66) not_3s(plural,_).
(67) not_3s(singular,Per)
       :-first_or_second(Per).
(68) first_or_second(first).
(69) first_or_second(second).
```

制約は分割することができないので、(70)が呼ばれる。

```
(70) normalize([cat_of_japanese(L),
                cat_of_work(R),
                psr(L,R,M)],
               [M])
```

新しい項 c0(M) が作られ、(63)が利用される。すると、右辺が失敗しないならば、(71)が作られる。

```
(71) c0(s(Tense)):-
       project([cat_of_japanese(n(Num,Per)),
                cat_of_work(v(Num,Per,Tense))],
               [Tense]).
```

この射影は(72)を呼ぶ。

```
(72) normalize([cat_of_japanese(n(Num,Per)),
                cat_of_work(v(Num,Per,Tense))],
               [Tense]).
```

新しい項 c1(Tense) が作られ、(65)が利用される。すると、もし右辺が失敗しないならば、(73)が作られる。

```
(73) c1(present):-
       project([cat_of_japanese(n(Num,Per)),
                not_3s(Num,Per)],
               []).
```

この射影の第一引数は充足可能なので、*NIL* が返される。従って(74)が作られ、(71)が c1(Tense) を返すので、(75)が作られる。

```
(74) c1(present).
(75) c0(s(Tense)):-c1(Tense).
```

(75)はデータベースに加えられる時、(76)に帰着される。

```
(76) c0(s(present)).
```

結局 [c0(M)] が返される。

このように制約射影はCYKパージングに応用可能であるがいうまでもなく制約射影はCYK以外のアルゴリズム、例えば、アクティブチャートパーザなどに用いることができる。

5 インプリメンテーション

制約単一化と制約射影を Sun 4 sparc station 1 上の Sun Common Lisp 3.0 でインプリメントした。これらは、簡単な Prolog インタープリタ[12]に基づいており、普通の単一化のメカニズムは同じものを用いている。また Prolog, 制約単一化, 制約射影を選言的単一化方法として用いる3つのCYKパーザを構築した。文法と辞書は HPSG[10]に基づいている。それぞれの辞書項目は平均すると選言の要素を約3つ含む。

表1は3つのパーザの計算時間を比較したものである。入力文が長いと、制約単一化は制約射影よりも効率が悪い。

6 関連研究

グラフ単一化の世界では Carter[2]が親ノードに関係のない情報を捨てるボトムアップ解析法を提案している。しかし、Carterの方法は、捨てられる情報の無矛盾性の検出に失敗する。さらに Carter の方法では、項文規則の適用がすべて終った後に情報を捨てるのに対して、制約射影はゴール無関連制約をダイナミックにその処理の中で捨てていく。これが制約射影の方がいいとするもう一つの理由である。

入力文	実行時間 (秒)		
	Prolog	制約単一化	制約射影
He wanted to be a doctor.	3.88	6.88	5.64
You were a doctor when you were young.	29.84	19.54	12.49
I saw a man with a telescope on the hill.	(out of memory)	245.34	17.32
He wanted to be a doctor when he was a student.	65.27	19.34	14.66

表 1: 計算時間の比較

制約射影のもう一つの利点はあまりコピーを必要としないことである。制約射影は利用されるホーン節のみがコピーされる。これが制約射影が他の選言的素性構造単一化方法よりも効率的で、記憶領域を使わないと期待できる理由である。

Hasida[6] は 3.3 節で説明した問題点を克服するもう一つの方法として依存伝播 (dependency propagation) を提案している。これは大域変数 (transclausal variables) を依存関係の効率的な発見のために用いる方法である。しかし、子カテゴリの情報を捨ててもよいという仮定の下では、制約射影はその単純さの故に、より効率的であろう。

7 おわりに

制約射影という選言的素性構造単一化の新しい演算を提案した。制約射影の特徴は特定の変数のみへの制約を返すことである。制約射影は選言的素性構造単一化の一方法にとどまらず、論理的推論システムとも考えることができる。従って、言語処理と非言語処理との統合に重要な役割を果たすと期待される。

謝辞

日頃より御指導いただく小暮潔氏と島津明氏に感謝致します。また制約単一化について御教示を賜わった橋田浩一氏と津田宏氏に感謝致します。

参考文献

[1] A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation, and Compiling, Vol. 1: Parsing.* Prentice-Hall, 1972.

[2] D. Carter. Efficient Disjunctive Unification for Bottom-Up Parsing. In *Proc. of COLING-90*, vol. 3, pp. 70–75, 1990.

[3] A. Eisele and J. Dörre. Unification of Disjunctive Feature Descriptions. In *Proc. of ACL-88*, 1988.

[4] 橋田浩一, 白井英俊. 条件付単一化. コンピュータソフトウェア, 3(4), pp. 28–38, 1986.

[5] K. Hasida. Conditioned Unification for Natural Language Processing. In *Proc. of COLING-86*, pp. 85–87, 1986.

[6] K. Hasida. Sentence Processing as Constraint Transformation. In *Proc. of ECAI-90*, pp. 339–344, 1990.

[7] R. T. Kasper. A Unification Method for Disjunctive Feature Descriptions. In *Proc. of ACL-87*, pp. 235–242, 1987.

[8] M. Kay. Parsing in Functional Unification Grammar. In *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, pp. 251–278. Cambridge University Press, 1985.

[9] F. C. N. Pereira and D. H. D. Warren. Definite Clause Grammar for Language Analysis—A Survay of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence*, Vol. 13, pp. 231–278, 1980.

[10] C. J. Pollard and I. A. Sag. *Information-Based Syntax and Semantics, Vol.1 Fundamentals.* CSLI, 1987.

[11] H. Tuda, K. Hasida, and H. Sirai. JPSG Parser on Constraint Logic Programming. In *Proc. of ACL European chapter '89*, pp. 95–102, 1989.

[12] 梅村恭司. Small Prolog インタープリタ移植のすすめ. *bit*, 15(6), pp. 59–66, 1983.