Vol. 36 No. 1 情報処理学会論文誌 Jan. 1995

論理的制約の射影演算を用いた単一化に基づく構文解析

中野幹生† 島津 明†

本論文では，単一化ベースの効率的な構文解析を行うための論理的制約充足法である制約射影を提案し，それを日本語のチャート解析へ応用して行った実験の結果を報告する．DCG 流の項単一化の方法では，選言的素性記述を論理的制約で表し，論理的制約の充足によって単一化を行うことができる．効率的な論理的制約の充足方法として制約単一化が提案されている．制約単一化は論理的制約を変換することで充足と同等の処理を行う．制約単一化は，Prolog を用いるより，結果がコンパクトになり，構文解析のように制約充足の結果がまた別の制約充足で用いられる場合に効率がよい．しかし，制約単一化は，解析が進むにしたがって制約中のリテラルの引数の数が増え，効率が悪くなるという欠点をもつ．制約射影は制約単一化を拡張したもので，論理的制約と，ゴールとよばれる変数リストを与えた時に，ゴール中の変数に関して，入力の制約と同等で，かつ，ゴール中の変数に関する情報しか持たないような制約を出力する．制約射影を用いることによって，引数の増加を抑えることができる．制約単一化および制約射影をチャート解析に適用し，その計算時間およびメモリ使用量を比較した結果，制約射影が，より少ない時間，少ないメモリで解析することができた．また，入力文の複雑さが増すにしたがって，差が広がることが判明した．

Unification-Based Parsing Using Projection of Logical Constraints

MIKIO NAKANO† and AKIRA SHIMAZU†

This paper describes constraint projection, a logical constraint satisfaction method developed for efficient unification-based parsing. In the context of term unification, disjunctive feature descriptions are represented by logical constraints, and the unification of the descriptions can be considered as a logical constraint satisfaction. Constraint unification is a constraint satisfaction method that transforms logical constraints into satisfiable ones and is more efficient than using Prolog. However, the constraint-unification-based parser has a defect in that the number of arguments in the literals in the transformed constraints increases as parsing proceeds. Constraint projection is a modification of constraint unification. It takes a logical constraint and a list of variables called the goal, and outputs a constraint equivalent to the input constraint concerning variables in the goal and it only has information about those variables. Constraint projection enables increase in the number of arguments to be avoided. It is proved that the parser employing constraint projection can parse all the sample sentences more efficiently than the parser employing constraint unification.

1. はじめに

近年，単一化に基づく自然言語解析や文法理論の研究が盛んになってきている[2),7),8),23),25)]．単一化ベースのアプローチは，規則や辞書の宣言的な記述を可能にし，従って，文法の記述，改変，追加を容易にするという工学的な利点を持っている．単一化文法では，辞書項目に多くの情報を持たせるため，句構造規則の数が少なくてすむ．したがって，辞書の情報量が多くなり過ぎる傾向がある．この問題は，選言を用いて，共通部分を共有する形で記述することによって解決できる[12),15),16)]．

選言的素性構造を用いることによって，辞書の冗長性を減らすことはできるが，もし，選言的素性構造がKayの構文解析法[15)]のように，解析の前に選言標準形に展開されたとすると，単一化は非常に効率が悪くなってしまう．効率的な選言的素性構造単一化のために，選言の余分な展開を避ける単一化アルゴリズムが提案されている[4),5),13)]．

これらの研究はいわゆるグラフ単一化をベースにしている．グラフ単一化ではラベルつき有向グラフで素性構造を表現するのに対し，DCG[22)]のように一階の項で素性構造を表す方法を項単一化という．グラフ単一化と項単一化は用いるデータ構造が異なるため，単

† NTT 基礎研究所
NTT Basic Research Laboratories

純に比較はできないが，グラフを項に変換した方が単一化の効率が良いこと[24]や，複数の素性値の選言（組合せ的な選言）が自然に表現できるという点などから我々は項単一化のアプローチをとる．

選言的項単一化法として制約単一化[10),27)]がある．制約単一化では，選言的素性構造は論理的制約（本論文では単に制約と呼ぶこともある）で表され，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性の保証された形の制約に変換することと等しい[9)]．制約単一化は，素性構造を表す制約を変換することによって，その単一化を行う．制約単一化は，不必要な変換を避けること，および，変換後の制約が選言を含むことから，効率的な構文解析を可能にする．

しかし，制約単一化を用いた構文解析には問題点がある．制約単一化では，制約変換が行われるたびに，変換後の制約に含まれるリテラルの引数の数がふえていくため，構文解析が進むと，変換に時間がかかるようになる．この問題点は，制約単一化が返す制約が，以後の解析では用いられない情報を蓄えていることに起因する．

この問題を解決するために，制約射影（constraint projection）という制約変換法を提案する．制約射影は，制約単一化を拡張したもので，制約と，ゴールと呼ばれる変数の集合を与えた時に，ゴール中の変数に関して入力の制約と同等で，かつ，ゴール中の変数に関する情報しか持たないような制約を出力する．これによって，以後の解析には用いられない情報を捨てることができ，効率よく構文解析を行うことができる．さらに，制約射影をチャート解析に応用する方法，および，その有効性を示す．

2. 論理的制約による選言的素性記述

2.1 論理的制約

まず，論理的制約を定義する．論理的制約（以後制約と呼ぶ）は，確定節論理の正リテラルの集合である．制約の要素である正リテラルを制約要素（または要素）と呼ぶ．本論文では，制約を **B**, **C** などのボールド体の英大文字で表し，制約要素を T，H などのローマン体の英大文字で表す．(1)は制約の一例である．制約要素は DEC-10 Prolog[3)] の記法を用いて書く（X，Y など大文字で始まるものは変数である．）．

(1)　{p(X), q(X, f(Y))}

述語の定義節は，その述語が頭部の述語に等しいようなホーン節[17)]のことである．例えば，(2)は p の定義節である．

(2)　p(f(X, Y))← {r(X), s(Y)}

定義節の本体は，制約と同等であると考えることができる．本体は，頭部にある変数を制約していると考えることができる．たとえば，定義節(2)は X と Y のあるインスタンスに関し，それらのインスタンスが，{r(X), s(Y)} を満たせば，p(f(X, Y)) が真であることを意味する．本体が空の場合は←の右辺には何もかかない．システムに登録されている定義節の集合をデータベースと呼ぶ．

定義節と制約の意味を規定するため，最小エルブランモデル[17)]を採用する．そうすると，ある基礎原始式 (ground atomic formula)[17)] が真であるということは，その基礎原始式がデータベースに存在する節から推論できるということに等しい．例えば，述語 p の定義節が(3)と(4)だけであったとする．

(3)　p(a)←

(4)　p(b)←

この場合，p(a) と p(b) は真であるが，p(c) は真ではない．したがって，制約 {p(X)} は，変数 X が a か b であると規定していることになる．もし，制約の中に定義節を持たない述語が存在する場合には，その制約を満たす代入はないと考える．

次に制約の構造を定義する．制約の構造とは，その制約に現れる各述語の定義節と，それらの定義節の本体に現れる各述語の定義節といったように，その制約を定義するすべての定義節の集合である．次節に例を示す．

本論文では，再帰的に定義された述語，すなわち，述語の定義節の本体に同じ述語が現れたり，2つ以上の述語の定義が相互に依存しているような述語は考えない．

2.2 選言的素性記述

選言のはいっていない素性構造[25)]は確定節論理の項を用いて表現することが可能である．例えば，(5)は(6)のように記述することができる．

(5)　$$\mathbf{sign}\begin{bmatrix}\text{pos } \mathbf{v}\\ \text{agr } \mathbf{agr}\begin{bmatrix}\text{num } \mathbf{sing}\\ \text{per } \mathbf{3rd}\end{bmatrix}\\ \text{subj } \mathbf{sign}\begin{bmatrix}\text{agr } \mathbf{agr}\begin{bmatrix}\text{num } \mathbf{sing}\\ \text{per } \mathbf{3rd}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

(6)　sign(v, agr(sing, 3rd), sign(_, agr(sing, 3rd), _))

(5)は，タイプつき単一化文法[6]で用いられる，タイプつき素性構造である．**sign**, **v** などのボールド体の記号はタイプを表す．タイプに応じて，取り得る素性の種類が決まっていると仮定する．タイプ階層は特に仮定しない．(6)中の "_" は無名変数である．関数記号 sign の引数はそれぞれ *pos* (*p*art *o*f *s*peech) 素性値，*agr* (agreement) 素性値，*subj* (subject) 素性値に対応し，それぞれの値は，v, agr(sing, 3rd), sign(-, agr(sing, 3rd), -) である．

選言を含むような素性構造は，定義節の集合を用いて表現することができる．例えば，選言的素性記述(7)は，本論文の枠組では，(8)で表現する．

(7)
$$\left\{\begin{array}{l}\textbf{sign}\left[\begin{array}{ll}\text{pos} & \textbf{v}\\ \text{agr} & \boxed{1}\left\{\begin{array}{l}\textbf{agr}\left[\begin{array}{ll}\text{num} & \textbf{sing}\\ \text{per} & \left\{\begin{array}{l}\textbf{1st}\\ \textbf{2nd}\end{array}\right.\end{array}\right]\\ \textbf{agr}\ [\text{num}\ \textbf{plural}]\end{array}\right.\\ \text{subj} & \textbf{sign}\ [\text{agr}\ \boxed{1}]\end{array}\right]\\ \textbf{sign}\left[\begin{array}{ll}\text{pos} & \textbf{n}\\ \text{agr} & \textbf{agr}\left[\begin{array}{ll}\text{num} & \textbf{sing}\\ \text{per} & \textbf{3rd}\end{array}\right]\end{array}\right]\end{array}\right\}$$

(8)
```
p(sign(v, Agr, sign(-, Agr, -)))
                          ← {not_3s(Agr)}
p(sign(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)←
```

値の指定されていない素性の値は，無名変数 '_' で表す．p(X) は X が p で規定されるような選言的素性記述の一つであることを意味している．(8)の制約要素 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規定している．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．この例からわかるように，本体の要素は選言に対応し，その要素の述語の各定義節は選言要素 (disjunct) に対応する．また，制約 {p(X)} の構造は(8)であるので，制約の構造を用いて選言的素性記述が行えることがわかる．

上記の方法により，HPSG[23], JPSG[8], PATR-II[25], LFG[2] のような単一化文法を，論理的制約を用いて記述することができる．SUBCAT 素性や SLASH 素性のように，集合やリストの値をとる素性の記述のためには，append や member などの再帰的に定義された述語が必要となる．しかし，実際の自然言語文では，SUBCAT 素性や SLASH 素性の値の要素の数は無限に大きくなることはない．したがって，SUBCAT 素性や SLASH 素性の値の要素の数の上限を定めることによって，再帰的に定義された述語の使用を避けることができる．以上のことにより，論理的制約は，単一化文法を表現するのに十分な記述力を持っていると言える．

2.3 制約の変換による選言的素性記述の単一化

前節で説明したように，論理的制約で選言的素性記述が行えるが，選言的素性構造同士の単一化[13),5]は，論理的制約の充足に相当する[10]．例えば，p(X) で表される選言的素性記述と q(Y) で表される選言的素性記述の単一化は，{p(X), q(X)} を満たす X のインスタンスの集合を求めることに等しい．

単一化の結果を新たな単一化に用いることを考えると，結果を制約の形で出力することが望まれる．そのような充足方法を制約変換[9]という．制約変換は，入力の制約が充足可能な場合は，それと同等の制約を返し，充足不可能な場合は失敗するような演算である．どのような形の制約を返すかによって，それが新たな単一化に用いられる時の効率を左右する．

もっとも単純な制約変換の方法として Prolog を用いる方法がある．例えば，{p(X), q(X)} の変換は，Prolog の呼び出し

```
?-p(X), q(X), assert(c0(X)), fail.
```

を行い，もし，c0 の定義が1つでもできれば，{c0(X)} を返し，そうでなければ失敗することによって行うことができる．ここで，c0 は制約変換器が自動的に生成する述語であり，それまでに使われていないとする．c0 の定義節には p(X) と q(X) を満たすすべての X の値が書き込まれるため，{p(X), q(X)} と {c0(X)} は同値である．再帰的に定義された述語を扱わないことから，この呼び出しが無限ループに陥ることはない．

3. 制約充足を用いたチャート解析

本章では，前章で説明した，論理的制約による選言的素性記述を用いて上昇型のチャート解析[14]を行う方法について説明する．

文脈自由文法の場合，チャート解析の基本的なデータ構造である弧 (edge) は次の形で表現される．

$[A \rightarrow \beta \cdot \gamma, i, j]$

ここで，A は非終端記号，β, γ は 0 個以上の非終端記号もしくは終端記号の列である．この弧は $i+1$ 番目の語から j 番目の語までの句が，β で表されるカテゴリの列として解析されたことを示している．この弧 (edge) を我々のシステムでは次のように表現する．

[X→・ξ, i, j, **C**]

Xは変数で，ξ は変数の列である．**C** はXと ξ 中の変数に対する制約で，この制約によって，実際にどのような論理項，すなわち素性構造がXと ξ に来ることができるかを規定する．ドットの左側には記号は書かない．これは，単一化文法においては，素性構造に意味表現を含むすべての情報が書いてあるため，解析後に解析木をとり出す必要がないからである．

[X→・Y, 1, 4, {c34(X, Y)}]

は，もし，XとYが制約 {c34(X, Y)} を満たすならば，5番目以降のある単語（仮に n 番目の単語とする）までが素性構造Yを持つとすると，2番目の単語から n 番目の単語までが素性構造Xを持つ句になることを示す．

次に句構造規則のデータ構造を示す．句構造規則のデータ構造は，次の形をしている．

〈X→ξ, **C**〉

Xは親の素性構造に相当する変数，ξ は子の素性構造に相当する変数の列，**C** はXと ξ の各変数に対する制約である．例えば，

〈X→Y Z, {c3(X, Y, Z)}〉

は，あるX，Y，Zが {c3(X, Y, Z)} を満たせば，素性構造Yを持つ句と素性構造Zを持つ句の並びが，素性構造Xを持つ句を構成することを示す．

これらのデータ構造を用いて，上昇型チャート解析の基本手続きを示す．上昇型チャート解析は，(a)辞書を調べて，入力文の各単語の素性構造を記述した語彙弧 (lexical edge) を作る手続き（辞書引き），(b)不活性弧と句構造規則から，新たな弧を作る手続き (手続き1)，(c)隣合った句の活性弧と不活性弧から，新たな弧を作る手続き（手続き2），の3つの手続きからなる．

まず辞書引きであるが，j 番目の単語が w だとすると，

[X→・, $j-1$, j, {p$_w$(X)}]

をアジェンダ（要素の並び替えが可能なリスト）に入れる．ここで，p$_w$ は単語 w の選言的素性記述を規定する述語で，p$_w$ と w の対応は辞書で示され，p$_w$ の定義節はあらかじめデータベースに存在しているとする．

手続き1は不活性弧(9)と規則(10)から(11)をつくる操作である．

(9) [X→・, i, j, **C**$_1$]

(10) 〈X0→X1…Xn, **C**$_2$〉

(11) [X0→・X2…Xn, i, j, **C**]

ここで，**C** は，(11)内に現れる変数 X0, X2, …, Xn に対する制約で，**C**$_1$∪**C**$_2$∪{eq(X, X1)} を変換したものである．ただし，eq は同等性を表す述語で，

eq(X, X)←

を定義節として持つ．もし，**C**$_1$∪**C**$_2$∪{eq(X, X1)} が充足不可能なら，弧(11)は作らない．

手続き2は活性弧(12)と不活性弧(13)から(14)を作る操作である．

(12) [X0→・X1…Xn, i, j, **C**$_1$]

(13) [X→・, j, k, **C**$_2$]

(14) [X0→・X2…Xn, i, k, **C**]

ここで **C** は **C**$_1$∪**C**$_2$∪{eq(X, X1)} を制約変換したものである．手続き1と同様に，制約変換が失敗したら弧は作らない．

4. 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である，制約単一化[10),27)]を説明し，その問題点を指摘する．

4.1 制約単一化の概略

2章で述べたように，もっとも単純な制約変換の方法として Prolog を用いる方法があるが，以下の理由から効率が悪い．2つ以上の定義節が適用可能な時に，Prolog は1つの定義節を選んで処理を進め，バックトラックするため，同じ処理を何回も繰り返さなければならない．しかも結果として得られる制約の述語の定義節は全く選言を含まない．したがって，この変換でできた弧が次回の句構造規則の適用で用いられる時に，効率が悪くなってしまう．

制約単一化は，この問題点を解決することができる．制約単一化を用いることによって，選言の余分な展開を避け，変換後の制約をコンパクトな形にすることができる．

制約単一化は入力された制約をモジュラー[10)]と呼ばれる形に変換する．制約がモジュラーであるとは，次の3つを満たすことである．

(1) すべての制約要素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) すべての制約要素の述語のすべての定義節の

本体がモジュラーであるか，空である（この定義を満たす述語をモジュラー定義であるという）．

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，(15)はモジュラーな制約であるが，(16)，(17)，(18)はモジュラーではない．

(15) {p(X, Y), q(Z, W)}

(16) {p(X, X)}

(17) {p(X, Y), q(Y, Z)}

(18) {p(f(a), g(Z))}

モジュラーな制約は必ず充足可能である[10]．制約をモジュラーな形に変換することは，制約を満たすインスタンスの集合を見つけることと同等である[10]．再帰的定義をもつ述語が用いられないという条件の下では，充足可能な制約は必ず同等でモジュラーな制約に変換することができ，充足不可能な制約の変換は失敗する[10]．

制約単一化の方法の概略は，(a)入力された制約の制約要素を，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，それらを独立に処理する，(b)すでにモジュラーであるような制約は処理しない，の2つである．(a)は不必要な繰り返しを行わないこと，(b)は遅言の不必要な展開を避けることに相当する．これらのアイディアによって，Prolog を用いるのに比べ，効率良く制約変換を行うことができる．制約単一化の詳しいアルゴリズムは文献[10]などに述べられている．

4.2 制約単一化の問題点

制約単一化を用いたチャート解析の欠点☆は，解析が進むに従って，制約の中に現れる変数の数が増えることである．例えば，

(19) [X1→•, 1, 4, {c34(X1)}]

と

(20) 〈X2→Y2, {c3(X2, Y2)}〉

から次の弧ができる場合を考える．

(21) [X2→•, 1, 4, {c35(X1, X2, Y2)}]

ここで {c35(X1, X2, Y2)} は {c34(X1), c3(X2, Y2), eq(X1, Y2)} を変換してできた制約である．このとき {c35(X1, X2, Y2)} は，それより左側には存在しない変数 Y2, X1 を持つ．次に(21)と(22)から(23)ができるとする．

(22) 〈X3→Y3 Z3, {c4(X3, Y3, Z3)}〉

(23) [X3→• Z3, 1, 4, {c36(X3, Y3, Z3, X2, Y2, X1)}]

ここで，{c36(X3, Y3, Z3, X2, Y2, X1)} は {c4(X3, Y3, Z3), c35(X1, X2, Y2), eq(X2, Y3)} を制約単一化で変換してできた制約である．このようにして，手続き1を行うたびに，引数の数が増えていく．手続き2が用いられる場合も同様である．

引数の増加は次の2つの理由から計算時間の増加を引き起こす．(a)引数の増加が，新しいリテラルの作成や単一化等の計算時間を増加させること．(b)結果として得られる制約の構造が大きくなり，その制約が後の制約変換で用いられる時に効率が悪くなること．

5. 制約射影

5.1 制約射影の概略

上記の問題を解決するため，制約を，特定の変数に関する情報のみを持つ制約に変換する方法として，制約射影と呼ぶ演算を提案する．制約射影の定義を示す前に，いくつかの補助概念を定義する．まず基礎代入(ground substitution)[17]の制限を定義する．

定義 1 [基礎代入の制限] 変数の集合 V への，ある基礎代入 σ の制限 σ_V とは，σ のうち，V中の変数への束縛のみを取り出したものである．

例えば，σ={X/a, Y/b}，V={X} とすると，σ_V={X/a} である．次に制約の制限を定義する．

定義 2 [制約の制限] 制約 **C1** が，制約 **C2** の，変数の集合Vへの制限であるとは，(a) **C1** がVの変数のみを持つ，(b) $\{\sigma_V | \mathbf{C2}\sigma$ が真$\} = \{\theta | \mathbf{C1}\theta$ が真$\}$ の2つを満たすことである．ここで，σ, θ は基礎代入であるとする．

例えば，定義節が

c1(a) ←

c1(c) ←

c2(a, b) ←

c2(c, d) ←

で与えられている時，{c1(X)} は {c2(X, Y)} の {X} への制限である．また，制約が正規であることを次のように定義する．

定義 3 [制約の正規性] 制約が変数の集合 V に関して正規であるとは，以下の4つの条件を満たすことである．

(a) すべての制約要素のすべての引数が変数．

(b) どの変数も2箇所以上に現れない．

(c) Vの要素以外の変数をもたない．

☆ ここで述べる欠点は，チャート解析特有の問題点ではなく，CYK 法[1]などを用いた場合にも生ずる．

(d) 各制約要素の述語のすべての定義節の本体は，頭部に現れる変数の集合に関して正規（この条件を満たす定義節を正規定義節と呼ぶ）．

条件(c)，(d)は，制約がV中の変数に関する情報のみを持つということを意味する．例えば，各述語の定義節が(24)で与えられる時，(25)，(26)，(27)のうち，{X, Y} に関して正規な制約は(25)のみである．

(24) p(a) ←
q(b) ←
r(c, d) ←
s(X) ←{r(X, Y)}

(25) {p(X), q(Y)}

(26) {r(X, Y), q(Z)}

(27) {s(X)}

以下では，あらかじめデータベースに存在する定義節はすべて正規定義節であると仮定する．以上の概念を用い，制約射影を次のように定義する．

定義 4 [制約射影] 制約射影とは，ある制約 **C** とゴール（goal）と呼ばれる変数の集合Vとが与えられた時に，**C** が充足可能な場合には，**C** のVへの制限であり，かつ，Vに関して正規であるような制約（**C** のV上への射影と呼ぶ）を返し，そうでなければ，"fail" を返すような演算である．

言い替えると，制約射影は，制約 **C** と変数の集合Vに関して同等で，Vに関する情報のみを持ち，充足可能であるような制約を返す．

5.2 制約射影のアルゴリズム

制約射影は主関数 project，副関数 normalize，その他の補助関数からなる．project と normalize が相互に呼びあって処理が進む．

主関数 project は引数として制約とゴール（変数の集合）をとる．そして，制約を互いに変数を共有しないいくつかの制約に分割する．それらの制約は2つのグループに分類される．1つはゴール中の変数を持っている制約のグループで，もう1つはその他の制約のグループである．前者のグループの制約をゴール関連制約と呼び，後者のグループの制約をゴール無関連制約と呼ぶ．ゴール関連制約のみを normalize を用いて，ゴールに関して正規な制約に変換し，ゴール無関連制約は充足可能性のみをチェックする．そして，normalize によって得られた正規な制約の和集合を返す．

副関数 normalize は制約とゴールを引数としてとり，ゴールに関して正規であり，かつ，入力の制約のゴールへの制限であるような制約を返す．結果の制約は要素を1つしか持たない．この要素の述語は正規定義節を持つ．

以下，project と normalize のアルゴリズムを示す．

まず，project のアルゴリズムの概略を示す．アルゴリズム中に現れる関数 vars は，制約または制約要素を引数としてとり，その中に現れる変数の集合を返す関数である．

Function project (**C**: 制約，V: 変数の集合) は，**C** が充足可能な場合は，Vに関して正規であり，かつ **C** のVへの制限であるような制約を返し，そうでなければ "fail" を返す．

P1. **C**={}なら{}を返す．

P2. V={} の時は，**C** が充足可能なら {} を返し，さもなくば，"fail" を返す．

P3. divided_cons := $\{\mathbf{C}_1, \cdots, \mathbf{C}_n\}$．ただし，$\cup_i \mathbf{C}_i = \mathbf{C}$ で，$\mathbf{C}_i$ が互いに素でかつ互いに変数を共有しない．

P4. relevant_cons := divided_cons の要素のうちゴール中の変数を持つものの集合．

P5. irrelevant_cons := divided_cons\relevant_cons.

P6. irrelevant_cons の各要素 **I** に対して，**I** が充足不可能なら，"fail" を返す．

P7. **S** := {}.

P8. relevant_cons の各要素 **R** に対して，以下を行う．

P8.1. V′ := vars(**R**)∩V.

P8.2. **B** := normalize(**R**, V′).

P8.3. **B**="fail" なら "fail" を返す．さもなくば，**S** := **S**∪**B**.

P9. **S** を返す． ■

次に normalize のアルゴリズムの概略を示す．

Function normalize (**C**: 制約，V: 変数の集合) は，**C** が充足可能な場合は，Vに関して正規であり，かつ **C** の V への制限であるような制約を返し，そうでなければ "fail" を返す．

N1. もし **C** が要素を1つしか持たず，Vに関して正規ならば，**C** を返す．

N2. N := f($X_1, \cdots, X_n$) の形の新しいリテラル．ただし，f は今まで使われていない述語で，$X_1, \cdots, X_n$ はVのすべての要素．

N3. T := V 中の変数をもつ **C** の要素の1つ．

N4. **R** := **C**\{T}.

N 5. flag :=nil.

N 6. T の述語の各定義節 $H_i \leftarrow \mathbf{B}_i$ に対し以下を行う.

N 6.1. T と H_i が単一化可能で θ_i をその最汎単一化子 (most general unifier[17]) とするとき,

N 6.1.1. $\mathbf{Q}_i$:=project($(\mathbf{B}_i\theta_i \cup \mathbf{R}\theta_i)$, vars($N\theta_i$)).

N 6.1.2. もし $\mathbf{Q}_i \neq$ "fail" なら $N\theta_i \leftarrow \mathbf{Q}_i$ をデータベースに入れ, flag :=t とする.

N 7. もし flag=nil なら "fail" を返し, さもなくば {N} を返す. ■

具体例として,

(28) {p(X, Y), q(Y, Z), p(A, B), r(A), r(C)}

を変換することを考える. ただし, ゴールとして {X, C} を与える. 各述語の定義節は制約単一化の例と同様に次のとおりとする.

p(f(A), C)← {r(A), r(C)}
p(a, b)←
q(a, b)←
q(b, a)←
r(a)←
r(b)←

この射影は, 次の project の呼び出しにより始まる.

(29) project({p(X, Y), q(Y, Z), p(A, B), r(A), r(C)}, {X, C})

この実行は以下のように進む (**図 1**).

- (P 3) divided_cons={{p(X, Y), q(Y, Z)}, {p(A, B), r(A)}, {r(C)}}.
- (P 4) relevant_cons={{p(X, Y), q(Y, Z)}, {r(C)}}.
- (P 5) irrelevant_cons={{p(A, B), r(A)}}.
- (P 6) {p(A, B), r(A)} が充足可能であることを確認.
- (P 8) (a)normalize({p(X, Y), q(Y, Z)}, {X}) を呼ぶ. 後述するように {c0(X)} という形の制約がかえる. (b)normalize({r(C)}, {C}) を呼ぶ. {r(C)} がかえる.
- (P 9) **S**= {c0(X), r(C)} を返す.

次に, normalize({p(X, Y), q(Y, Z)}, {X}) の実行を通して normalize を説明する. この実行は以下のように進む (**図 2**).

- (N 2) N=c0(X), ただし, c0 は今までに使われていない述語.
- (N 3) T=p(X, Y).
- (N 6) p(f(A), C)← {r(A), r(C)} を用いる. H_1= p(f(A), C), $\mathbf{B}_1$= {r(A), r(C)}.
- (N 6.1) p(f(A), C) と p(X, Y) を単一化すると, θ_1= {X/f(A), Y/C}.
- (N 6.1.1) project({r(A), r(C), q(C, Z)}, {A}) を呼ぶ. $\mathbf{Q}_1$= {r(A)} となる.
- (N 6.1.2) c0(f(A))← {r(A)} をデータベースに追加. flag=t.
- (N 6) p(a, b)←を用いる. H_2=p(a, b), $\mathbf{B}_2$= {}.
- (N 6.1) p(a, b) と p(X, Y) を単一化すると, θ_2= {X/a, Y/b}.

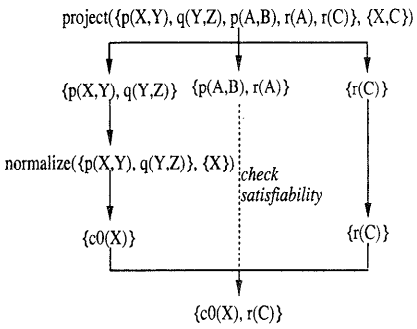


図 1 project の実行例
Fig. 1 Example execution of project.

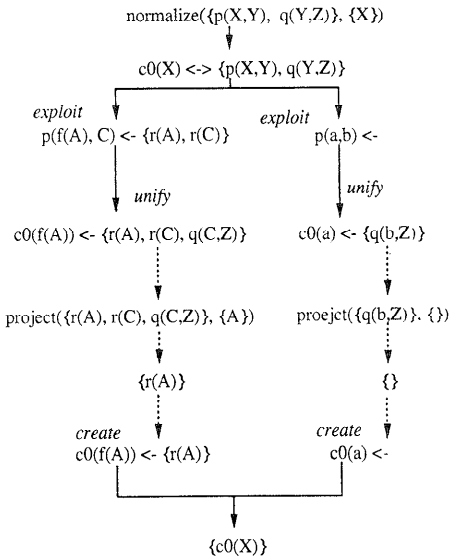


図 2 normalize の実行例
Fig. 2 Example execution of normalize.

- (N 6.1.1) project({q(b, Z)}, {}) を呼ぶ．$Q_2 = \{\}$ となる．
- (N 6.1.2) c0(a)←をデータベースに追加．
- (N 7) {N} = {c0(X)} を返す．

この実行中に新しく作られる定義節は次のとおりである．

c0(f(A))← {r(A)}

c0(a)←

これを制約単一化の例と比べると，制約射影の返す制約の構造がコンパクトであることが分かる．

5.3 制約射影を用いたチャート解析

制約射影をチャート解析に用いるには，3章で説明した手続き1と手続き2を次のように変更する．まず，手続き1では，3章の弧(11)の $\mathbf{C}$ を $\mathbf{C}_1 \cup \mathbf{C}_2 \cup \{eq(X, X1)\}$ の $\{X0, X2, \cdots, Xn\}$ 上への制約射影とする．次に手続き2では，弧(14)の $\mathbf{C}$ を $\mathbf{C}_1 \cup \mathbf{C}_2 \cup \{eq(X, X1)\}$ の $\{X0, X2, \cdots, Xn\}$ 上への制約射影とする．このように，制約射影を用いて，弧内の変数のみの情報をもつ制約を求める．

6. 実験および考察

制約単一化と制約射影とを Sun SPARCstation 10 モデル41上の Lucid Common Lisp 4.0.0 で実装した．これらは，選言を含まない項の単一化は同じものを用いている．さらに，これらの制約変換を用いた上昇型チャート解析器を製作した．この解析器を用いて日本語の解析実験を行った．実験で用いた文法は，JPSG[8] をもとにしている．この文法と解析器を用いることにより，日本語の基本的な文型の文を解析し，論理形式を出力することができる．実験に用いた文の中から代表的なものを選んで**表1**に示した．現在の文法では扱っていない現象として，浮遊数量子や再帰代名詞「自分」などがある．

制約単一化と制約射影との間で，計算時間およびメモリの使用量の差を調べたものを**表2**に示す．純粋に構文解析の時間のみを測定するため，わかち書きしたものを入力とした．Prolog を用いた解析も行ったが，メモリの不足のため，どの文も解析できなかった．こ

表1 主な解析可能な文
Table 1 Examples of parsable sentences.

文番号	文(わかち書きしたもの)	特徴
1.	鈴木さん は 藤原さん を 愛し ている	単文
2	誰 が 京都 に 行っ た の	疑問詞疑問文
3.	鈴木さん は 岡山 で 生まれ 東京 で 育っ た	並列構造
4.	太郎 は 花子 に 数学 を 喫茶店 で 教え た	単文
5.	花子 が 太郎 に 数学 を 喫茶店 で 教え られ た	受身
6.	藤原さん は 佐藤さん に 仕事 を 手伝っ てくれる ように 頼ん だ	接続助詞を含む文
7.	何度も 電話し た のにもかかわらず 福田君 に 連絡する こと はできなかっ た	否定文
8.	山田君 に マージャン の ルール を 教え た の は 課長 だ	A は B だの形の文
9.	京都 で 学会 が あっ た 時 に 僕 も 行き まし た	形式名詞を含む文
10.	太郎 が 花子 に 教え た 道 を 健 が 奈緒美 に 教え た	連体修飾節を含む文
11.	鈴木さん は 大学 で 物理 を 勉強し た が 今では 言語 の 研究 を し ている	接続助詞を含む長文

表2 計算時間と使用メモリの比較
Table 2 Comparison of computation time and memory used.

文番号	単語数	弧の数	語の曖昧度 (*1)	計算時間(秒)		使用メモリ(キロバイト)	
				制約単一化	制約射影	制約単一化	制約射影
1	6	32	3.67	2.64	0.91	1424	539
2	7	41	4.57	2.44	1.66	1261	741
3	9	486	4.56	89.99	17.93	44478	9983
4	10	93	3.80	10.96	3.49	7274	1641
5	11	160	5.64	15.61	6.42	8753	3026
6	11	112	4.45	15.31	4.86	9683	2260
7	12	336	4.58	60.02	13.14	38861	7141
8	12	134	4.75	19.01	7.35	9838	3624
9	13	280	4.23	49.45	13.02	25293	6084
10	14	116	3.86	7.71	4.53	4749	2104
11	16	378	4.38	113.23	15.11	58776	8002

(*1) 文中の各単語の辞書項目を定義するためのホーン節の数の平均．

の表からわかるとおり，すべての文の解析において，制約射影が最も少ない時間，メモリで解析できる．また，文が複雑になるにしたがって，計算時間，メモリ使用量の差は広がる．

以上の実験結果により，制約射影による構文解析の効率性を確かめることができたが，次になぜ制約射影が効率的であるかを調べるために，表1中の第11文の解析において，次の点を調査した．まず，新しくリテラルが作られる時の引数の数の平均は，制約単一化の場合が19.0であったのに対し，制約射影の場合は2.3であった．次に，制約射影において，制約がゴール関連制約の数とゴール無関連制約に分割される時に，ゴール無関連制約がどのくらいの割合であるのかを調べたところ，2042個中273個（約13.4%）であった．以上のことから，引数の数の増加を抑えること，および，ゴール無関連制約を発見することの双方が効率に寄与していることが分かった．

7. おわりに

論理的制約の一変換方法，制約射影を提案し，そのアルゴリズムを提示するとともに，その構文解析への応用について述べた．制約射影は，すでに提案されている変換法である制約単一化に比べて，コンパクトな結果を得るという利点を持つ．制約射影を用いることにより効率的な構文解析が可能になることを，実装することによって確認した．自然言語の文法記述法として注目されている単一化文法は，論理的制約を用いて記述することができるため，それに基づく構文解析法の効率化の研究は重要であると考えられる．

制約を用いたその他の言語処理の研究として，丸山[19]，長尾[21]らの研究がある．彼らは係り受け解析を有限領域の制約の充足[18]の枠組で行うことを提案しているが，この方法は単一化文法には適用できない．また，丸山[20]は，選言的素性記述を丸山[19]の枠組の制約充足で扱うことを提案しているが，その枠組での制約の記述力は，本論文で扱う制約に比べると弱いため，選言的な素性記述に困難が生じる可能性がある．

津田[26]は，Partially Specified Term (PST) というグラフ型のデータ構造を扱えるように，制約単一化を拡張している．本論文で提案した手法にも，同様にしてPSTを導入することが可能である．

また，橋田[11]は，ポテンシャルエネルギーという概念を用いて制約処理を制御することによって，単一化，構文解析，その他の処理を統一的に行うことを目指している．本論文の手法にも確率などの数値を用いた制御を導入することにより，より柔軟で高速な処理を行う方法が考えられる．

本論文で提案した制約射影は一般的な論理的制約変換法であるため，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できる．

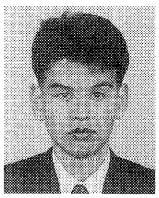
謝辞 貴重なコメントを頂いたNTT基礎研究所小暮潔主幹研究員，山崎憲一主任研究員，および対話理解研究グループの各氏に感謝いたします．また制約処理について御教示を賜わった電子技術総合研究所の橋田浩一氏と(財)新世代コンピュータ技術開発機構の津田宏氏に感謝いたします．

参考文献

1) Aho, A. V. and Ullman, J. D.: The *Theory of Parsing, Translation, and Compiling, Volume 1: Parsing*, Prentice-Hall (1972).
2) Bresnan, J. W. (ed.): *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Mass (1982).
3) Clocksin, W. F. and Melish, C. S.: *Programming in Prolog*, Springer-Verlag (1984).
4) Dörre, J. and Eisele, A.: Feature Logic with Disjunctive Unification, *Proc. 13th COLING*, Vol. 2, pp. 100-105 (1990).
5) Eisele, A. and Dörre, J.: Unification of Disjunctive Feature Descriptions, *Proc. 26th ACL*, pp. 286-294 (1988).
6) Emele, M. C. and Zajac, R.: Typed Unification Grammars, *Proc. 13th COLING*, Vol. 3, pp. 293-298 (1990).
7) Gazdar, G. and Mellish, C.: *Natural Language Processing in Lisp: An Introduction to Computational Linguistics*, Addison-Wesley (1989).
8) Gunji, T.: *Japanese Phrase Structure Grammar*, Reidel, Dordrecht (1987).
9) 橋田：情報の部分性と制約プログラミング，古川，溝口，Lassez, J.-L. (編)，制約論理プログラミング，pp. 123-142, 共立出版 (1989).
10) 橋田，白井：条件付単一化，コンピュータソフトウェア，Vol. 3, No. 4, pp. 28-38 (1986).
11) Hasida, K.: Dynamics of Symbol Systems: An Integrated Architecture of Cognition, *Proc. FGCS '92*, pp. 1141-1148 (1992).
12) Karttunen, L.: Features and Values, *Proc. 10th COLING*, pp. 28-33 (1984).
13) Kasper, R. T.: A Unification Method for Disjunctive Feature Descriptions, *Proc. 25th ACL*, pp. 235-242 (1987).

14) Kay, M.: Algorithm Schemata and Data Structures in Syntactic Processing, Technical Report CSL-80-12, Xerox PARC (1980).
15) Kay, M.: Parsing in Functional Unification Grammar, Dowty, D.R., Karttunen, L. and Zwicky, A.M. (ed.), *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, pp. 251-278, Cambrige University Press (1985).
16) 小暮：増進的複製による型付き素性構造汎化手法，情報処理学会論文誌，Vol. 34, No. 9, pp. 1919-1930 (1993).
17) Lloyd, J.W.: *Foundations of Logic Programming*, Springer-Verlag (1984).
18) Mackworth, A.K.: Constraint Satisfaction, Shapiro, S.C. (ed.), *Encyclopedia of Artificial Intelligence* (*second ed.*), pp. 285-293, John Wiley (1992).
19) Maruyama, H.: Structural Disambiguation with Constraint Propagation, *Proc. 28th ACL*, pp. 31-38 (1990).
20) Maruyama, H.: JUANT: A Constraint Solver for Disjunctive Feature Structures, *Proc. 14th COLING*, pp. 1162-1166 (1992).
21) Nagao, K.: A Preferential Constraint Satisfaction Technique for Natural Language Analysis, *Proc. 10th ECAI*, pp. 523-527 (1992).
22) Pereira, F.C.N. and Warren, D.H.D.: Definite Clause Grammar for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artif. Intel.*, Vol. 13, pp. 231-278 (1980).
23) Pollard, C.J. and Sag, I.A.: *Information-Based Syntax and Semantics, Vol. 1 Fundamentals*, CSLI Lecture Notes Series No. 13, CSLI, Stanford (1987).
24) Schöter, A.: Compiling Feature Structures into Terms: an Empirical Study in Prolog, Technical Report EUCCS/RP-55, Centre for Cognitive Science, University of Edinburgh (1993).
25) Shieber, S.M.: *An Introduction to Unification-Based Approaches to Grammar*, CSLI Lecture Notes Series No. 4, CSLI, Stanford (1986).
26) Tsuda, H.: cu-Prolog for Constraint-Based Grammar, *Proc. FGCS '92*, pp. 347-356 (1992).
27) Tuda, H., Hasida, K. and Sirai, H.: JPSG Parser on Constraint Logic Programming, *Proc. 4th European Chapter of ACL*, pp. 95-102 (1989).

(平成6年3月4日受付)
(平成6年10月13日採録)



中野 幹生（正会員）

1965年生まれ．1988年東京大学教養学部基礎科学科第一卒業．1990年同大学大学院理学系研究科相関理化学専攻修士課程修了．同年日本電信電話(株)入社．現在，同社基礎研究所情報科学研究部対話理解研究グループに所属．自然言語処理の研究に従事．構文解析，対話理解等に興味を持つ．日本ソフトウェア科学会，人工知能学会，言語処理学会，ACL 各会員．

島津 明（正会員）

1948年生．1971年九州大学理学部数学科卒業．1973年同大学院修士課程修了．同年日本電信電話公社入社．現在，NTT 基礎研究所情報科学研究部対話理解研究グループリーダ．言語コミュニケーションの研究に従事．工学博士．言語処理学会，計量国語学会，電子情報通信学会，人工知能学会，ACL, ACM 各会員．