ICOT Technical Memorandom: TM-1117

TM-1117

Cu-Prologによる選言的素性構造

津田　宏

October, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~5
Telex ICOT J32964

Institute for New Generation Computer Technology

cu-Prologによる選言的素性構造
Disjunctive Feature Structure in cu-Prolog

津田 宏 (TSUDA, Hiroshi)
(財) 新世代コンピュータ技術開発機構第3研究室
Institute for New Generation Computer Technology

概要

単一化文法における選言的素性構造 (disjunctive feature structure) は、表現の難しさはもとより、それらの単一化は本質的にNP-completeであることが知られているため、実装においてより実際的な手法が求められている。

本稿では、記号的・組み合わせ的制約論理型プログラミング言語cu-Prologにデータ構造としてPST(partially specified term: 部分項)を取り入れ、選言的素性構造の処理を試みる。応用として、JPSG(Japanese Phrase Structure Grammar)パーザを紹介し、パーザ記述, 言語制約記述, 選言的素性構造が統一した枠組で扱うことができることを示す。

1 はじめに

単一化文法[9]に基づく自然言語処理の実装においては、素性構造の効率良い表現法と単一化が重要な課題である。選言を含まない素性構造は、アークに素性名のラベルを持つ有向グラフと見なせ、それらの単一化はunion/findアルゴリズムにより、almost linearで行え、upper boundも$O(n \log n)$である[4]。

しかし、自然言語の曖昧性を自然に表現し処理するためには、本質的に曖昧性を含んだ選言的素性構造 (disjunctive feature structure: 以下DFSと略す) を扱う必要がある。

我々は、記号的・組み合わせ的制約を扱うことのできる制約論理型言語cu-Prologを開発し、その上でJPSGに基づく日本語パーザを作成してきた[11]。本稿ではcu-PrologにPST(Partially Specified Term)を導入することで、表現力を上げ、制約を拡張することでDFSも同じ枠組で取り扱う。

2 選言的素性構造 (DFS)

DFSは素性構造の中に選言を組み込んだもので、FUG[6]をはじめ多くの単一化文法で用いられる。それは、次の構造からなる[5]。

value disjunction 一つの素性の取り得る値のsetにより表現される。次の例は素性posの値はnまたはvを、scの値は[]または[pos/p]を取ることを示している。

$$\left[\begin{array}{l} pos = \{n, v\} \\ sc = \left\{\begin{array}{l} [] \\ [\ pos = p\ ] \end{array}\right\} \end{array}\right] \qquad (1)$$

general disjunction 複数の素性の取り得るsetにより表現される。次の素性構造の例では$sem = love(X, Y)$は共通で、残りの部分は曖昧性があることを示している。

$$\left[\begin{array}{l} \left\{\begin{array}{l} \left[\begin{array}{l} pos = n \end{array}\right] \\ \left[\begin{array}{l} pos = v \\ vform = vs \\ sc = \left(\ \left[\ pos = p\ \right]\ \right) \end{array}\right] \end{array}\right\} \\ sem = love(X, Y) \end{array}\right] \qquad (2)$$

path equivalence (non-local path) 複数の相異なる素性の値が等しいことを表現する。FUGの記法に基づく次の例では、素性adj.obj(素性adjの値の素性objの値をこのようにパスで表現する)の値sbjは、素性actorの値ともunifyすることを表している。

$$\left[\begin{array}{l} adj = \left[\begin{array}{l} pos = p \\ obj =< actor >= sbj \end{array}\right] \end{array}\right] \qquad (3)$$

FUGでは全ての選言を展開したDNF(Disjunctive Normal Form)によりDFSを格納し、それらの単一化を行うという処理を行ったため、非常に効率が悪かった。DFSの単一化はNP-completeで、最悪の場合には単一化の計算量は指数オーダーになる[5]。しかし、実際の多くの場合では[4, 1]のように、選言をなるべく展開しないことで多項式オーダーのアルゴリズムが得られている。

本稿では最初の二つのdisjunctionについて検討する。non-local path構造は、変数を用いてそれを含まないものに書き換えることで対処する。

3 cu-Prolog

cu-Prolog[11]は、記号的・組み合わせ的制約を対象領域とする制約論理型言語で、AIや自然言語処理への応用に適している。

cu-Prologのプログラム節は次の制約付きホーン節 (Constraint Added Horn Clause:CAHC) からなる。

$$\overbrace{H}^{Head} :- \overbrace{B_1, \ldots, B_n}^{Body}; \overbrace{C_1, \ldots, C_m}^{Constraint}.$$

$$\overbrace{H}^{Head}; \overbrace{C_1, \ldots, C_m}^{Constraint}.$$

H は頭部 (Head) であり、$B_1, \ldots, B_n$ は本体 (Body)。$C_1, \ldots, C_m$ は制約 (Constraint) で、頭部または本体に現れる変数についての制約を表す。

```
_member(X,[X|Y]). % member 述語の定義
_member(X,[Y|Z]):-member(X,Z).

_@ member(X,[ga,wo,ni]),member(X,[wo,ga,to]). % オリジナルな制約

solution = c0(X_0) % 等価に変換された制約
c0(wo). % 変換で作られた新述語の定義
c0(ga).
CPU time = 0.017 sec
0%(program) 0%(pst/const) 1%(string)

_@ member(X,[ga,ni]),member(X,[to,he]).

solution = fail. % 変換失敗
CPU time = 0.017 sec
0%(program) 0%(pst/const) 1%(string)
```

図 1: cu-Prolog の制約変換系の実行例

cu-Prolog の導出規則は以下のように通常の Prolog に制約解消手続きを加えたものである。

$$A, \mathrm{K}; \mathrm{C}.\ ,\ A' :- \mathrm{L}; \mathrm{D}.,$$

$$\frac{\theta = mgu(A, A'), \mathrm{C}' = mf(\mathrm{C}\theta, \mathrm{D}\theta)}{\mathrm{L}\theta, \mathrm{K}\theta; \mathrm{C}'}$$

ここで、A は素式、K、L、C、D、C′ は、素式列。$mgu(A, A')$ は、A と A' の most general unifier である。

mf は制約変換系で、$mf(C_1, \ldots, C_m)$ は $C_1, \ldots, C_m$ と等価で標準形の制約である。標準形に変換できない場合にはこの規則の適用は失敗する。制約の標準形をモジュラーという。

cu-Prolog の制約変換系は unfold/fold 変換 [10] にヒューリスティックスを組み込んだものである (図 1は実行例)。

4 cu-Prolog による DFS

4.1 PST(部分項)

cu-Prolog 第三版では、単一化文法の素性構造の記述枠組として CIL[7] の PST(Partially Specified Term: 部分項) を採用した。cu-Prolog における PST は

$$\{l_1/t_1, l_2/t_2, \ldots, l_n/t_n\}$$

の形の項である。l_i はアトムで $l_i \neq l_j (i \neq j)$、また t_i は PST も含めた任意の項である。

現バージョンでは再帰的な構造は許しておらず、またラベルも全順序 (辞書式など) で整列して格納しているため、単一化はリストのマージにより行われる。[1]

4.2 制約の標準形

以下の議論に必要な用語、モジュラーという制約の標準形を定義する。

[Def] 1 (引数の成分) 述語の各引数位置の成分 *(component)* とは、その引数が具体化し得る *PST* のパスの集合である。通常項 *(アトム, リスト, 複合項)* はパス [] と見なす。

[1]実装上 PST はつねに変数 (ポインタ) を介して、X=PST という形の制約として取り扱われる。X=f/U,g/U と Y=f/1,h/2 の単一化により、X=Y,Y=f/1,g/1,h/2 と単一化子 (1 — U) が得られる。

述語 p の第 n 引数の成分を C(p,n) と書き、C(T) により項 T のパスの集合を表す。また、特殊な述語として X=t なる形がある。変数 X は成分が C(t) の引数位置にあるとみなす。

引数の成分はプログラムを静的に解析することにより得られる。そのアルゴリズムは次の通りである。

1. 全ての述語の引数の成分の初期値を ϕ にする。
2. 定義節の頭部に通常項が現れる引数については、成分に [] を加える。
3. 定義節の頭部に PST f が現れる引数については、f のパスのうち変数を値にとらないものの集合を加える。
4. 以下の操作を引数の成分の変化がなくなるまで繰り返す。
5. p の定義節の頭部で第 n 引数が変数 X の場合、本体で X の出現する引数位置の成分を p の第 n 引数の成分に加える。
6. p の定義節の頭部において第 n 引数が PSTf で f.1=X (X は変数) の場合、本体で X の出現する引数位置の成分の各要素を 1 に連接したものを p の第 n 引数の成分に加える。

次例では、C(c0,1)={f,g,h}, C(c0,2)={[]},C(c0,3)=ϕ, C(c1,1)=ϕ,C(c1,2)={[]}, C(c2,1)={f,h} となる。

```
c0({f/b},X,Y):-c1(Y,X).
c0(X,b,_):-X={g/c},c2(X).
c1(X,X).
c1(X,[X|_]).
c2({h/a}).
c2({f/c}).
```

[Def] 2 (依存関係) 制約 *(素式列)* において、

1. 同一変数が共通の成分を持つ複数の引数位置に出現している場合、または
2. 同一変数が成分 {[]} の引数と、成分が *PST* のパスのみからなる引数位置に出現している場合、または
3. 成分が ϕ でない引数位置が具体化されている場合

は依存関係があると言う。

[Def] 3 (モジュラー (制約の標準形)) 依存関係の存在しない制約をモジュラーという。

4.3 value disjunction の記述

一素性の取り得る値の集合を記述する value disjunction は、素性値に変数を導入し、それに制約を課すことにより記述できる。この方法で、次のように general disjunction の一部である複数の素性の取り得る値の集合も記述できる。

```
T={f/X,g/U},c0(X,U)
c0(a,b).
c0(c,d).
```

4.4 general disjunction の記述

DFS X は次のように CAHC の制約部分にモジュラーに記述できる。

$$X = PST, c_1(X), c_2(X), \ldots, c_n(X)$$

X=PST は Kasper[4] の unconditional conjunct に対応し、選言を含まない PST を表す。$c_1(X), c_2(X), \ldots, c_n(X)$ は conditional conjunct に対応する。

例えば DFS(2) は T={sem/love(X,Y)},c1(T) と記述される。ただし述語 c1 の定義は次の通り。

```
c1({pos/n}).
c1({pos/v,vform/vs,sc/[{pos/p}]}).
```

4.5 複数の素性構造の間の選言の記述

従来の DFS の研究では、一つの素性構造の中の選言しか対象としていなかった。しかし、後述の単一化文法の実装に見られるように、複数の素性構造の間の関係を記述したい場合が多い。cu-Prolog では二つの素性の間の選言は次のようにモジュラーに記述される。

$$X = PST_1, Y = PST_2, C1(X,Y), \ldots, Cn(X,Y)$$

例えば以下のような述語 C1 により、X.f と Y.g の素性の値の組み合わせが記述される。

```
C1({f/a},{g/1}).
C1({f/b},{g/2}).
```

4.6 DFS の制約変換

制約の標準形の拡張に伴い、制約解消系の拡張も必要となる。ここでは unfold 変換を拡張する。DFS の制約変換は、PST 引数に関する依存関係を解消することである。これは

$$U = \{f/t\}, c(U) \text{ ただし、 } f \in C(c,1)$$

なる形の制約解消に帰着される。[2]この制約は次の抽出 (extract) 操作により $U = \{f/t\}, c1(U, V_0, \ldots, V_n)$ と依存構造を含まない形に変換される。
ただし、$C(c1,1) = C(c,1) - \{f\}$ であり、$V_0, \ldots, V_n$ は t に含まれる変数である。

[Def] 4 (抽出 (t が変数 V の場合)) $U = \{f/V\}, c(U)$ は $U = \{f/V\}, c1(U,V)$ に変換される。述語 c1 は c の定義から次のように作られる。[3]

1. c(gt):−Body. に対応する定義は作らない。
2. c({f/X} ∪ Rest):−Body. に対しては
   c1(Rest,X):-Body.
3. c(p):-Body., f ∉ C(p),p は PST に対しては
   c1(p,_):−Body.
4. c(Y):-Body.,Y は変数 に対しては
   c1(Y,V):−Y = {f/V},Body. として本体を再帰的に抽出した後、Y = {f/V} を消去する。

t が functor の場合も同様の処理が考えられる。t が ground の場合は新述語の引数は増えないだけで、ほぼ同様に行われる。

例として、次の T1 と T2 の単一化を示す [1]。

```
T1={a/X},s(X)
T2={a/{b/X},d/X}
s({b/+,c/-}).
s({b/-,c/+}).
```

T1 と T2 の PST 単一化により

```
T1=T2
T2={a/U,d/X},U={b/X},s(U)
```

を得る。C(s,1)={b,c} なので素性 b に関して依存関係が生じる。抽出によって次のように述語 s1 を定義する

```
s1({c/-},+).
s1({c/+},-).
```

これにより、T2={a/U,d/X},U={b/X},s1(U,X) と依存関係を含まない形に変換される (C(s1,1)={c} である)。

本方法は制約変換により新述語を作るため、[4] に比べてメモリの消費量は大きい。しかし fold 変換で部分解を再利用するため時間効率は良くなる。複数の依存関係がからみあっている問題に対しては、どの依存関係から解消するかというヒューリスティックスが重要になる。本枠組の大きな特徴はこの扱いにあると言ってもよい [11, 3]。

5 応用例:JPSG パーザ

cu-Prolog の応用として JPSG パーザを説明する。

5.1 lexical ambiguity

同音異義語を別々の辞書エントリに実装しては、バックトラックの度に辞書引きを行い、非常に効率が悪くなる。制約により辞書エントリをまとめるのが制約ベースの自然言語処理の常套手段である。以下は、助動詞「れる」の辞書の一部である。五段、サ変動詞の未然形に接続し、動詞の subcat が二つ (主格、目的格) ならば通常の受身になり、一つ (主格) ならば被害の受身になる。

```
                % 「れる」の辞書記述
lex(reru,{sc/SC,sem/Sem,
   adjacent/{core/{pos/v,infl/I},sc/VSC,sem/Sem}});
     reru_form(I),        % 直前の動詞の活用型
     reru_sem(VSC,Vsem,SC,Sem).
                    % subcat,sem の組み合わせ

reru_form(vs).  % 五段動詞
refu_form(vs1). % サ変動詞
      % 被害の受身と通常の受身
reru_sem([{form/ga,sem/Sbj}],
  [{form/ga,sem/A},{form/ni,sem/Sbj}],affect(A,Sem)).
reru_sem([{form/ga,sem/Sbj},{form/wo,sem/Obj}],Sem,
  [{form/ga,sem/Obj},{form/ni,sem/Sbj}],Sem)
```

同様にして、決まった組み合わせしかとらない語句、文脈により素性の値が変わる語句の辞書記述はこの方法で行うことができる。

---

[2] c1(U),c2(U),C(c1,1)∩C(c2,1) ≠ φ なる制約は、新述語 c3 を c3(U):-c1(U),c2(U). と導入し、unfolding を行うことで、U=PST,c(U) の形になる。これは、∀f ∈ C(c,1)∩C(PST), U = PST′ ∪ {f/X} に対して、U={f/X},c(U) の依存関係を解消することに帰着される。

[3] 直感的には、述語のアリティを増やすことで特定の引数の成分を減らす操作である。

5.2 構造原理の実装

単一化文法の中でもHPSG,JPSGはconstraint-based grammarとも範疇化される[4]。これは文法が素性構造に対する言語制約として記述されるからである。一般にJPSGの制約は一つの枝分かれの親娘の三つのカテゴリの間の関係として記述される。例えば、親の素性の値が、両娘の素性のappendと等しという制約は次のようになる。

```
psr(Hcar,RDcat,LDcat);ffp(Hcat,RDcat,LDcat).
        % 句構造に制約を記述した例
ffp({slash/MS},{slash/RS},{slash/LS})
    :-append(RS,LS,MS).     % 制約ffpの定義
```

このように文法記述やパーザ記述と同じ枠組で統一的にDFSも扱うことができるのがcu-Prologの大きな利点である。

5.3 実行例

「健が愛する」という文の解析例を示す。

```
_:-p([ken,ga,ai,suru]).   % ユーザの入力

% パーザの返値:
category= {sem/[love,Sbj_678,Obj_680],    % トップカテゴリ
     core/{form/Form_1550, pos/v}, sc/Hsc_1622,
     ref1/Href_1624, slash/Hsl_1626, psl/[], sjn/[], sjc/[]}
constraint= c12(Cref_1598, Href_1624, Csl_1600, Hsl_1626, Csc_1596,
               Hsc_1622, Obj_680, Sbj_678, HC_284, Hsl_288),
           syu_ren(Form_1550)    % トップカテゴリに対する制約
true.
CPU time = 0.433 sec
18%(program) 3%(pat/const) 12%(string)
```

パーザは入力文に対応するカテゴリと、カテゴリに含まれる変数に課されている制約を返す。カテゴリの選言の中身を見るためには、次のように制約を解いてやれば良い。

```
_:-c12(_,Refl,_,Slash,_,Sc,Obj,Sbj,_,_). % 制約を解く
  Refl=[] Slash=[]  Sc=[{sem/VO_36, core/{form/so, pos/p}}]
      Obj=VO_36  Sbj=ken;     % 解1
  Refl=[] Slash=[{sem/V3_58}] Sc=[] Obj=V3_58 Sbj=Ken; % 解2
no.
CPU time = 0.067 sec
16%(program) 0%(pat/const) 12%(string)
```

最初の解はsubcat素性値が残っている。「奈緒美を健が愛する。」のような文の解析の中間結果に対応する。次の解はsubcatの値がslash素性に値が移動しており「健が(誰かを)愛する。」もしくは「健が愛する(誰か)」のような関係節の読みに対応している。

トップカテゴリに含まれていない変数も構文解析のプロセスの中で制約に入るので制約は多少冗長になりがちである。これはconstraint projection[8]により消去することが可能である。

6 考察

本論文ではcu-PrologにPSTを組み込むことで、選言的素性構造を効率良く取り扱うことができることを示した。既存の方法はDFSのみを扱う狭い枠組であるのに対し、本方法は文法記述やパーザ記述と同じ枠組で扱っている点でより一般的である。cu-Prolog,DP[2]など我々が行っている制約ベースの自然言語処理の有効性をまた一つ示したと言えよう。

[4]これに対してLFG,GPSG等はrule-based grammarと言われる

またこれはヒューリスティックスを容易に取り込める枠組でもあることにも言及しておきたい。ヒューリスティックスについては、[11, 3]等で得られたものが応用できる。例えば、unfoldingで依存関係の多いものから展開する、とかOR分岐の少ないものから展開する等はDFSに対してもやはり有効である。

謝辞

cu-Prologの実装はICOTの橋田浩一氏、中京大学の白井英俊先生と共同で行っています。また、Mark Johnson教授をはじめ様々な方に助言をいただきました。ここに感謝致します。

参考文献

[1] A. Eisele and J. Dörre. Unification of Disjunctive Feature Descriptions. In *Proc. of 26th ACL Annual Meeting*, pages 286–294, June 1988.

[2] K. HASIDA. Common Heuristics for Parsing, Generation,and Whatever. In *Workshop on Reversible Grammar in Natural Language Processing*, Berkeley, 1991.

[3] K. HASIDA and H. TSUDA. Parsing without Parser. In *Proc. of IWPT91*, pages 1–10. Sigparse ACL, February 1991.

[4] R. T. Kasper. A Unification Method for Disjunctive Feature Descriptions. In *25th ACL Annual Meeting*, pages 235–242, July 1987.

[5] R. T. Kasper and W. C. Rounds. A Logical Semantics for Feature Structure. In *Proc. of 24th ACL Annual Meeting*, pages 257–266, 1986.

[6] M. Kay. Parsing in functional unification grammar. In D. R. Dowty, L. Karttunen, and A. M. Zwicky, editors, *Natural language parsing*, pages 251–278. Cambridge university press, 1985.

[7] K. MUKAI and H. YASUKAWA. Complex Indeterminates in Prolog and its Application to Discourse Models. *New Generation Computing*, 3(4):441–466, 1985.

[8] M. NAKANO. Constraint Projection:An Efficient Treatment of Disjunctive Feature Descriptions. In *Proc. of 29th ACL Annual Meeting*, 1991.

[9] S. M. Shieber. *An Introduction to Unification-Based Approach to Grammar*. CSLI Lecture Notes Series No.4. Stanford:CSLI, 1986.

[10] H. TAMAKI and T. SATO. UNFOLD/FOLD Transformation of Logic Programs. In *Proc. of Second International Conference on Logic Programming*, pages 127–137, 1983.

[11] H. TSUDA, K. HASIDA, and H. SIRAI. JPSG Parser on Constraint Logic Programming. In *Proc. of 4th ACL European Chapter*, pages 95–102, 1989.