ICOT Technical Memorandum: TM-0804

TM-0804

制約充足としての構文解析
－制約論理型言語cu－Prologの応用

津田　宏，橋田浩一，白井英俊（中京大学）

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

制約充足としての構文解析 — 制約論理型言語 cu-Prolog の応用

Parsing as Constraint Satisfaction — an application of cu-Prolog

津田 宏 (TSUDA, Hiroshi)
橋田 浩一 (HASIDA, Kôiti)
(財) 新世代コンピュータ技術開発機構第 2 研究室
Institute for New Generation Computer Technology
白井 英俊 (SIRAI, Hidetosi)
中京大学 文学部
Chukyo University

概要

我々が開発した制約論理型プログラミング言語 cu-Prolog は、従来の多くの制約論理型プログラミング言語が制約に代数方程式を取るのと異なり、ユーザ定義述語による Prolog の素式列を制約とする。このため、自然言語処理、例えば単一化文法の処理に必要な記号的、組合せ的な制約を自然に宣言的に記述できる。

本稿では、制約パラダイムにおける構文解析の 2 つのアプローチを示す。一つは、cu-Prolog の制約付ホーン節 (Constraint Added Horn Clause) により句構造規則や、曖昧語の辞書定義を制約で宣言的に記述する方法である。また、もう一つは、cu-Prolog の unfold/fold 変換による制約解消系を改良して、構文解析プログラム自体を制約とみなし、制約変換により解析を行う方法である。後者では、制約変換時に作られる新述語により、チャートパーザと同等に解析が行われる。

1 はじめに

単一化文法 [6] は、文法理論としてのみならず、計算機による自然言語処理の文法記述としても重要になってきている。しかし、単一化文法の大きな特徴である宣言的な文法記述を自然に実装できるプログラミング言語はなかった。

しばしば使われる Prolog は、宣言的なプログラムの記述こそできるものの、そのプログラムの実行は手続き的（And 方向は左から右、Or 方向は上から下）になるために、実行効率が悪くなることもある。また、PrologII[2] 等に見られる bind-hook 機構は、変数に加えられた制約が充足するか否かが、変数が具体化してその制約を手続きとして実行するまでわからないので、やはり効率が悪くなる。

我々は、上記の Prolog の欠点を解消した制約論理型プログラミング言語 *cu-Prolog* [10] を開発した。cu-Prolog は Prolog のホーン節の一部を制約とした制約付ホーン節 *(CAHC: Constraint Added Horn Clause)* によってプログラムを記述し、従来の単一化の代わりに制約単一化 [12] [1] を採用している。

[1] この論文では、"条件付単一化" とも呼ばれている。また、cu-Prolog は Constraint Unification Prolog の略である。

本稿では、cu-Prolog の概要と、その応用として日本語句構造文法 (Japanese Phrase Structure Grammar) [4] のパーザーを紹介する。また、cu-Prolog の制約解消系を拡張して構文解析そのものを制約充足の過程として行うことの試みについても述べる。

2 cu-Prolog

多くの制約論理型プログラミング言語 (PrologIII [3]、CLP(R) [5] 等) は、代数方程式や不等式による、実数や有理数領域の制約を取り扱い、制約解消にはグレブナー基底等を用いている。

しかし、AI や自然言語処理には数値的な制約よりは、記号的、組み合わせ的な制約が重要になることが多い。cu-Prolog は、ユーザ定義述語の素式列による、エルブラン空間を領域とする制約を記述できるため、この手の問題に適していると言える。cu-Prolog の制約変換には unfold/fold プログラム変換 [8] を用いている。

2.1 CAHC

cu-Prolog のプログラム節は以下の制約付ホーン節 *(Constraint Added Horn Clause : CAHC)* からなる。

[Def] 1 (CAHC) 制約付ホーン節 は、以下の 2 種類の節からなる。

$$\overbrace{H}^{Head} :- \overbrace{B_1, B_2, \ldots, B_n}^{Body}; \overbrace{C_1, C_2, \ldots, C_m}^{Constraint}.$$

$$\overbrace{H}^{Head}; \overbrace{C_1, C_2, \ldots, C_m}^{Constraint}.$$

H は頭部 *(Head)* であり、$B_1, B_2, \ldots, B_n$ は本体 *(Body)*。$C_1, C_2, \ldots, C_m$ 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。

宣言的意味論からみると、上の 2 種類の節は次の Prolog のプログラム節と等価である。

1. $H :- B_1, B_2, \ldots, B_n, C_1, C_2, \ldots, C_m.$
2. $H :- C_1, C_2, \ldots, C_m.$

Prolog のホーン節は制約付ホーン節の特殊な場合になっている。

2.2 制約の標準形

CAHC の制約部分は、ユーザが定義する通常の Prolog の述語から成る。現バージョンでは、述語は satisfiable なものでなければならず、制約は次に示すモジュラーなものでなければならない。

[Def] 2 (モジュラー) 素式列 $C_1, C_2, \ldots, C_m$ は、以下の条件を満たす時モジュラーという。

1. 各 C_i の全ての引数は変数。
2. どの変数も 2 個所には現れない。

例えば、

member(X,Y),member(U,V) はモジュラー。
member(X,Y),member(Y,Z) はモジュラーでない。
append(X,Y,[a,b,c,d]) はモジュラーでない。

2.3 推論規則

cu-Prolog の導出規則は、通常の Prolog の規則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。

$mf(C_1, \ldots, C_m)$ は、$C_1, \ldots, C_m$ と等価でモジュラーな制約を（存在する場合に）返す制約解消系。

2.4 制約解消系

cu-Prolog の制約解消系は、モジュラーでない制約を、新たな述語をいくつか定義しながらモジュラーな制約に変換する（詳しくは [10] 参照）。制約変換には unfold/fold プログラム変換を用いているので、変換後の制約は元の制約と等価になる [11]。

例えば、

member(X,[a,b,c]),member(X,[b,c,d])

は、新制約 c0(X) に変換される。この変換では新述語 c0 が次のように定義される。

```
c0(b).
c0(c).
```

また、

member(X,[a,b,c]),member(X,[k,l,m])

は、充足不可能なので変換されない。

cu-Prolog の制約解消系は、選択関数にはヒューリスティックスを使っているが、深さ優先探索を行っているため、Prolog と同じ意味で不完全である。後で、この解消系を完全なものに改良する試みについて述べる。

```
_member(X,[X|Y]).
_member(X,[Y|Z]):-member(X,Z).
_append([],X,X).
_append([A|X],Y,[A|Z]):-append(X,Y,Z).

_@ member(X,[ga,no,wo,ni,kara,made,sae]),
   member(X,[to,he,ni,kara,sura,ga]).

solution = c0(X)
c3(ni).
c3(kara).
c0(ga).
c0(X0):-c3(X0).
CPU time = 0.017 sec

_@ member(A,Z),append(X,Y,Z).

solution = c14(A, Z, X, Y)
c15(X2, X2, X0, Y1, Y3):-append(X0, Y1, Y3).
c15(X2, Y3, X0, Y1, Z4):-c14(X2, Z4, X0, Y1).
c14(A0, X1, [], X1):-member(A0, X1).
c14(A0, [A1|Z4], [A1|X2], Y3):-c15(A0, A1, X2, Y3, Z4).
CPU time = 0.000 sec
```

最初の 4 行は、*member* と *append* の定義。"@" で始まる行は、ユーザーの入力素式列（制約）。cu-Prolog は、入力と等価でモジュラーな制約 ($c0(X)$) と、新述語の定義を返す。(CPU タイムは 1/60 秒単位で計測している)

図 1: 制約変換ルーチンの実行例

2.5 cu-Prolog の実装

cu-Prolog は C 言語で記述しており、現在 (Ver 2.3) 約 5,000 行である。各種 UNIX 及び MS-DOS 上で動いている。図 1 は、cu-Prolog の制約変換ルーチンの実行例である。

3 制約付加による構文解析

単一化文法に基づく自然言語のパーザを、CAHC で記述することを考えよう。例えば、日本語句構造文法（JPSG）における制約とは、局所的な枝分かれにおけるカテゴリの各素性の関係として宣言的に記述される。cu-Prolog では辞書や句構造規則を表すプログラム節に制約を直接付加できるので、宣言的な文法の自然な記述が可能である。構文解析プログラムを書くときには、構文木を作成する部分はプログラムの本体で手続き的に記述し、曖昧性に関わる部分のみ制約として記述するのが、処理効率の点からは望ましい。

ここでは JPSG に基づく簡単な日本語パーザーを紹介する。CAHC は、2 箇所で使われる。

3.1 制約による曖昧性の処理

一つは、同音異義語等の曖昧性の処理である。例えば、名詞 "はし" には、橋、箸、端などの曖昧性がある。しかるに、CAHC で辞書を記述すると、次のように一つのエントリで表すことができる。

lexicon([hasi|X],X,[...sem(SEM)]);hasi_sem(SEM).

ここで、hasi_sem は、次のように定義されている。

```
hasi_sem(bridge).
hasi_sem(chopsticks).
```

```
hasi_sem(edge).
```

意味素性の値は変数(SEM)で、制約hasi_sem(SEM)が課せられている。辞書の段階では曖昧でも、処理が進むうちに変数に別の制約が加わり、制約を変換することで曖昧性が減少していく。同音語の辞書を分けると、処理が失敗する度にバックトラックを行うことになり、能率が悪い。同様にして、動詞の活用語尾や、後置詞の記述も扱うことができる。図2は、制約による曖昧性の処理の例である。

3.2 JPSGの制約記述

JPSGの句構造規則もCAHCで自然に記述される。前述のように、局所的な枝分かれにおけるカテゴリの各素性の関係を記述できればよい。

JPSGでは[4]、FFP（束縛素性原理）は次のようになっている。

> 局所的な枝分かれにおいて、親の束縛素性の値は、子の束縛素性の値の和と単一化する。

CAHCを使うと、この原理は次のように句構造規則に埋め込むことができる。

```
psr([slash(MS)],[slash(LDS)],[slash(RDS)]);

union(LDS,RDS,MS).
```

しかし、Prologではこのように宣言的には表現できず、手続き的（psrかunionのどちらかが先に実行される）になってしまう。

3.3 評価

JPSGパーザは、現段階では、素性の数や辞書の語彙数は少ないものの、20語ほどの文を高々1秒以内で解析できる（VAX8600にて測定）。JPSGは、文法記述の枠組なので、パーザの解析アルゴリズムは独立に選ぶことができる。現バージョンでは、単純なレフトコーナーアルゴリズム[1]を用いているので効率は悪い。

4 制約充足としての構文解析

cu-Prologの制約解消系を改良して、構文解析のプログラム自体を制約と見て、制約変換により構文解析を行うことを考えよう。この方法では、前述のパーザでは取り扱えなかった木構造の曖昧性が処理できる。例えば、

> I see a man with a telescope.

のような構文的な曖昧性を持つ以下のような簡単なCFGの構文解析プログラムを考える。

```
P1 = parse([see|X],X,tvp).
P2 = parse([a,man|X],X,np).
P3 = parse([with,a,telescope|X],X,pp).
P4 = parse(X,Y,Cat):-parse(X,Z,C1),parse(Z,Y,C2),
     psr(C1,C2,Cat).
P5 = psr(tvp,np,vp).
P6 = psr(vp,pp,vp).
P7 = psr(np,pp,np).
```

現在のcu-Prologの制約変換では、探索木の再帰構造を検出する機構がないため

```
parse([see,a,man,with,a,telescope],X,Cat)
```

の制約変換を行うと無限ループに陥ってしまう。そこで、OLDT導出[9]と同様の反駁方略を用いて、再帰性を検出する。つまり、再帰性のある節は変換を一時中断し、別のOr分岐を先に実行し、そこで得られた解を用いて、再び中断中の節を変換する。

また、上のプログラムを見ると、述語parseの第2引数はつねに変数なのでそれ自体で何かに具体化することはない。これを空虚な引数位置という。そして、制約の標準形の定義も次のように拡張する。

[Def] 3（モジュラー（改訂版））素式列 $C_1, C_2, \ldots C_m$ は、以下の条件を満たす時モジュラーという。

1. 各 C_i の全ての引数は変数。
2. どの変数も、空虚でない引数位置のうち2個所には現れない。

以上の点に注意して、制約を変換すると（変換ではpsrのように単位節から成る述語は優先的に展開して無駄な述語の定義を訪ぐ）、

```
D0 = c0(X,Cat)
       <-parse([see,a,man,with,a,telescope],X,Cat).
T1 = c0([a,man,with,a,telescope],tvp).
T3' = c0(X,vp):-c1(X).
D1 = c1(X)<-parse([a,man,with,a,telescope],X,np).
T4 = c1([with,a,telescope]).
T6' = c1(X):-c2(X).
D2 = c2(X)<-parse([with,a,telescope],X,pp).
T7 = c2([]).
T10' = c0(X,vp):-c2(X).
```

これらの新定義節が作られ（D0,D1,D2は、c0,c1,c2の元の定義）、

```
parse([see,a,man,with,a,telescope],X,Cat)
```

はc0(X,Cat)に変換されたことになる。c0([],Cat)を解けば、2つの木構造に対応する2つの解が求まる。また、変換時に作られた新述語の元の定義を調べると、チャートパーザのチャートの働きをしており、無駄な処理を省いている（入力文字列の長さをNとして、時間空間ともに$O(N^3)$）ことがわかる。

5 今後の展開

今後の研究には種々の方向がある。まずは、上に述べた新しい制約変換手続きの定式化と実装が手近な目標である。

自然言語処理としては、パーザだけではなく、状況意味論の制約ベースの記述にも応用できる。こういった問題は、部分的な情報から処理を進めていくメカニズムがなければならない。またパーザにしても、構文的な制約と意味的な制約の強さを調節して、非文でも意味がとれるといったような現象のモデルにすることもできる。

```
_:-p([ken,ga,ai,suru]).

v[Form_675, AJN{Adj_677}, SC{SubCat_679}]:SEM_681---[suff_p]
  |
  |--v[vs2, SC{p[wo]}]:[love,ken,Obj0_415]---[subcat_p]
  |   |
  |   |--p[ga]:ken---[adjacent_p]
  |   |   |
  |   |   |--n[n]:ken---[ken]
  |   |   |
  |   |   |__p[ga, AJA{n[n]}]:ken---[ga]
  |   |
  |   |__v[vs2, SC{p[ga], p[wo]}]:[love,ken,Obj0_415]---[ai]
  |
  |__v[Form_675, AJA{v[vs2,SC{p[wo]}]}, AJN{Adj_677}, SC{SubCat_679}]:SEM_681---[suru]

cat     cat(v, Form_675, □, Adj_677, SubCat_679, SEM_681)
cond     c7(Form_675, SubCat_679, Obj0_415, Adj_677, SEM_681)
True.
CPU time = 0.050 sec

_:-c7(F,SC,_,A,SEM).
  F = syusi  SC = [cat(p, wo, □, □, □, Obj00_30)]  A = □  SEM = [love,ken,Obj00_30]
  F = rentai  SC = □  A = [cat(n, n, □, □, □, inst(Obj00_38, Type3_36))]
  SEM = inst(Obj00_38, [and,Type3_36,[love,ken,Obj00_38]])
no.
CPU time = 0.017 sec
```

「健が愛する」の実行例。パーザは、解析木及び最上ノードのカテゴリと制約を返す。「健が（誰かを）愛する」（終止形）または「健が愛する（誰か）」（連体形）の統語的な曖昧性は、制約(c7)の2つの解として表されている。

図 2: 曖昧な文の解析例

そのためには、制約の記述能力をさらに上げなければならない。例えば、否定も含め一階述語に拡張するとか[7]、制約の強さを陽に（数値などで）定義してデフォルト推論を行う等である。

謝辞

この研究はICOTのJPSGワーキンググループの郡司隆男教授をはじめとする委員諸氏、および東京大学の山田尚勇教授、小野芳彦氏の助言に負うところが大きい。ここに感謝致します。また、cu-Prologの実装は、一部ICOTの委託研究として行いました。

参考文献

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

[2] A. Colmerauer. Prolog II Reference Manual and Theoretical Model. Technical report, ERACRANS 363, Groupe d'-Intelligence Artifielle, Universite Aix-Marseille II, October 1982.

[3] A. Colmerauer. Prolog III. *BYTE*, August 1987.

[4] T. GUNJI. *Japanese Phrase Structure Grammar.* Reidel, Dordrecht, 1986.

[5] J. Jaffar and J. L. Lassez. Constraint Logic Programming. In *Proceedings of the 14th ACM POPL Conference*, pages 111–119, Munich, 1987.

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

[7] M. E. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler. In *Journal of Automated Reasoning*, pages 353–380, 1988.

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

[9] H. TAMAKI and T. SATO. OLD Resolution with Tabulation. In *Proc. of Third International Conference on Logic Programming*, pages 84–98, 1986.

[10] H. TSUDA, K. HASIDA, and H. SIRAI. cu-Prolog and its application to a JPSG parser. In *Proc. of Logic Programming Conference*, pages 155–164, Tokyo, 1989.

[11] 古川康一, 溝口文雄 共編. プログラム変換. 知識情報処理シリーズ7. 共立出版, 東京, 1987.

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