ICOT Technical Report: TR-215

TR-215

自然言語処理のための
単一化の拡張と遅延的実行制御

向井国昭

November, 1986

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

自然言語処理のための単一化の拡張と遅延的実行制御

向井国昭

(財)新世代コンピュータ技術開発機構

内容

1. はじめに
2. オブジェクトと制約
3. ロジック・プログラミング
4. 部分的に指定された項
5. 単一化
6. 遅延実行制御
7. 諸記法

付録1. CILのプログラム例題
付録2. 部分的に指定された項の意味論
付録3. 遅延型論理演算に基づく制約型プログラミング

要旨

単一化ベースの文法理論，状況意味論(Situation Semantics)の基礎である状況理論(Situation Theory)にあらわれる共通の単一化を，ロジック・プログラミング図式の枠組みの中でPrologに組み込む．この拡張されたPrologにおいて，構文論，意味論，そして語用論を，より，統一的に扱うことができる．

1．はじめに

従来から指摘されている，ロジック・プログラミングと自然言語処理との親和性の高さを，単一化ベースの文法理論，状況意味論(Situation Semantics)の基礎である状況理論(Situation Theory)，そしてロジック・プログラミング図式の三つの観点から分析する．特に，これら三つの領域のそれぞれに，互いに同型な単一化の構造が存在することを示す．そして，親和性の根拠としての，この共通の単一化(マージ)を組み込んだロジック・プログラミング図式の具体例として，Prologの拡張言語CILを紹介する．CILの意味領域は木の領域である．CILは，変数を含んだ木表現形式(部分的に指定された項，部分項)という拡張された項形式をもつ．CILは，宣言的な記述力を高めるために，算術式を含む制約等式言語とそのインタプリタを組み込んでいる．付録1にCILで書かれた例題プログラムを示す．付録2に部分項の形式意味論を詳述する．付録3で，遅延的論理演算に基づいた，遅延制御の構成法を述べる．

2．オブジェクトと制約

計算言語学は，形式的に見れば，オブジェクトとそれら間の制約(constraint)の対の形で理論が構成されている．例として，構文論からGazdarらのGPSG(generalized phrase structure grammar)とBresnanらのLFG(lexical functional grammar)を挙げる．一方，意味論の側でも，状況(situation)というオブジェクトとその間の制約(constraint)という枠組みを意味論の基本にすえる状況意味論(Situation Semantics: J. Barwise & J. Perry)が現われ，注目されている．状況意味論よれば，世界は状況から成り，それらは構造的な制約関係—たとえば因果関係—にある．

そして，興味ぶかいことに，これら構文論と意味論に現われる制約の形式は，いずれも次の形式をもつ：

$A \Rightarrow B$ (A ならば B である)

ロジック・プログラミングについても同様である．なぜなら，プログラム(ホーン節の集合)は論理素式の(ground atomic formula)の間の論理的帰結(logical consequence)関係であるから，これも上の図式の具現化と見ることができる．制約は限定節で書き表される範囲のものを想定して一般性を失わないことが分かる．

構文論の重要オブジェクトは素性(feature)であり，その単一化は素性のマージのことである．意味論の重要オブジェクトは事態(state of affairs)であり，その単一化は事態のマージのことである．ロジック・プログラミングの重要オブジェクトはエルブラン項であり，単一化は，一階述語論理の項の単一化のことである．したがって問題は，素性と事態と項を表現する共通形式と素性のマージと事態のマージと項の単一化(実は，恒等関係=に過ぎない)を含む共通拡張単一化を見出せということである．

本稿の部分項およびその単一化理論は，その問題の一つの解である．すなわち，部分項は構文論，意味論，およびロジック・プログラミングのオブジェクトを統一的に表現する．これは，構文論と意味論を横断的にまたがる制約を記述するときにも有利である．また，状況意味論のように，語用論も意味論であつかう立場ではなおさら威力を発揮すると期待される．

3．ロジック・プログラミング

ロジック・プログラミングは単一化(unification)(=等式理論(equality theory))と制御により特徴付けることができる[J.Jaffar 他 85]．話しを簡単にするた

めに，制御は下降型の深さ優先(SLD)と固定することにすると，各ロジック・プログラミング言語は単一化理論と同一視できることになる．通常の Prologは等式理論が x=x という公理たったひとつからなる場合である．無限木と Freezeで良く知られている Prolog-IIもこの図式のひとつの例である．それはX=f(X)を許すような等式理論を持っている．そして CILもこの図式の具体例であり，そのパラメータは部分項の間の単一化規則である．

ここで，ロジック・プログラミング(LP)の基礎を復習しよう．ついでに，制約ロジック・プログラミングの基本的な考え方も導入する(Jaffar 86)．なお付録2で，単一化ベースの文法理論に対する，ロジック・プログラミングからの解釈が与える．

限定節の集りをプログラムといい，原始論式の集りをゴールという．等式の集りを環境といい，ゴールと環境の対を導出状態と呼ぶ．次に，導出状態の間の遷移関係を定義する．変数を含まないことを基礎的(ground)という．たとえば，変数を含まない項のことを基礎項という．他も同様である．

[融合(resolution)ステップ]

(G,E)を与えられた導出状態とする．Gは空でないゴールとする．Gに含まれる任意の原始論理式をA，与えられたプログラムPに含まれる任意の限定節をδとする．δに含まれる変数をすべて新しい名前に変更し，それをδ'とする (renaming)．δ'が単位節ならば，βを空集合とおき，δが B<-C1, ..., Cnの形の限定節ならば，β {C1,...,Cn} とおく．そのとき，(G,E)を新しい計算状態((G-{A})∪β , E∪{A=P})に書き換える：

(G, E) -> ((G-{A})∪β , E∪{A=P})

つぎに新しい環境 E∪{A=P}を解かなければならない．Prologの場合は，つぎのような，等式系の書き換え規則に従う．それを，制約解決(constraint solver)機構の一種と見ることが要点である．

(Prologの制約解決)

x,yを変数，t,s,ai,bjを項，そしてf,gを関数記号とする．定数は，次数が0の関数記号とみなす．

i) x=xなる等式を取り除く．
ii) x=y (x≠y)ならば，すべてのyをxで書き換える．
iii) t=xをx=tに書き換える．
iv) x=t,x=sなる二つの等式を x=t,t=sで置き換える．但し，tはsよりサイズが大きくないとする．
iii) f(a1,...,an)=f(b1,...,bn)は a1=b1, ..., an=bnと置き換える．

また，等式系が次のタイプの等式を含めば，書き換えは失敗であると定義する．

i) xが非変数項tに出現しているような等式x=t．
ii) f(x1,...,xn)=g(y1,...,ym) (f≠g)．

(制約)等式の書き換えは書き換え規則が適用できなくなるまで続ける．書き換え規則が適用できない等式系を正規という．(失敗せずに)最終的に得られたこの正規等式系が解である．LPの手続き解釈とは，この，融合ステップと，制約解決の列である．最終導出状態が空ゴールと正規等式系の対であるとき，この計算は，成功(success)という．また，({A}, φ)が成功するような基礎論理式Aの全体集合をプログラムの手続き的意味という．

[プログラムPの宣言的意味]

Sを基礎論理式の集合で，つぎの性質を満たす最小のものとする．Sは，Pの単位節Cの総ての基礎例(ground instance，すべての変数を基礎項で一斉に書き換えたもの)を含む．任意の限定節の任意の基礎例 A<-B1,...,Bnに対して，{B1,...,Bn}⊂Sならば SはAを含む．すなわち，Sは ,imply(<-) に関して閉じている最小のモデルである．どんなプログラムPに対しても，その意味すなわちPの最小モデルが存在することが保証されている．この最小モデルのことをプログラムPの宣言的意味という．

つぎに，このように定義された，手続き意味と宣言的意味が一致することが証明できる．これは，限定節の世界では，①証明可能であること，②正しいということの二つが同値であることを示している．

Prologにおいて，基礎論理素式Aの証明が，有限のステップで失敗する場合，その意味はなんであろうか．それは，論理式 not Aが，プログラムPを完備化(completion)したものに，等号理論を加えたものの論理的帰結であることと同値である，というのが，有名な，negation as failure 規則の重要な性質である．ここで，Aのモデルが，常にBのモデルであることを「B は Aの論理的帰結(logical consequence)である」という．また完備化とは，おおざっぱに言って，implyを if and only if と読みなおすことである．詳細は省く．たとえばプログラム

```
integer(0).
integer(s(X))<-integer(X)
```

の完備化は

```
integer(U)<-> (U=0) or ∃X(U=s(X) and integer(X))
```

である．

今まで，限定節の中に，否定記号が出てこなかった．否定情報の扱いは，きわめて，重要であるが，それを許すと話が難しくなることも事実である．たとえば次のような拡張された，プログラムを考える．

```
p<- not q
q<- not p
```

これは，二つの極小モデル{q},{p}を持つことがモデルの定義からすぐ分かる．(基礎論理式の集合Mは 基礎論理式Aを含んでいないとき，およびそのときに限り 論理式not A のモデルであると定義されている．)

最近のロジック・プログラミングの基礎論では，このようなnotをボディに許す限定節に拡張したクラスが議論されている．上の例ではpという概念が，「まわりまわって」not pという，自身の否定に帰着している．これは，vicious circleの一種を考えられる．このような，circleを含まないプログラムのクラスすなわち，stratified ロジック・プログラムのクラスが注目され， その最小モデルとnegation as failure間の関係が議論されている．詳細は省かざるを得ない．

4． 部分的に指定された項(Partially Specified Term)

通常の項(term)は，fをファンクタそしてa1，...,anを項としてつぎのような再帰的な構造として定義されるものである：

```
f(a1, ..., an)
```

a1,a2等を引数と呼ぶ．引数は，一般には，論理変数を含んでよい．この記法では引数の位置が暗黙的に指定されており，それが(速い)単一化の基礎になっている．しかし引数順序は本来便宜上のものである．またいつもすべての引数を書かなければならず複雑なデータを表現する場合に誤りを起こしやすいという実際的な欠点もある．我々は ，GPSGの素性 ，LFG の機能構造，状況意味論の関係や割り当てを計算機上での表現と操作の観点から検討した結果，部分項の概念にたどりついた．部分項という名前は項の必要な部分のみを指定する記述法という面を強調したものである．

部分項は，非常にありふれたデータ構造の一つである．例えば上に挙げた例の他にも，連想リスト，セマンティック・ネットワーク，フレーム，レコード，値-属性対リストがある．ロジック・プログラミングに部分項とその単一化を組み込むことの意義がそこにある．

部分項は

{a1/b1, ..., an/bn}

の形で書かれる.ここでaiはエルブラン項であり，biはエルブラン項—ただし引数に部分項を含んでもよい—または部分項である.各 ai/biの出現順序には意味がない.
部分項に関する呼び名はまだ確立されていないが，ここでは,通常の項との対応を考えて，各 ai を引数位置名と呼び，biをその位置の引数と呼ぶ．例として,貸すという事態を部分項で表現してみよう:

例1．{関係　　/ 貸す，
　　貸す人　　/ 太郎，
　　借りる人　/ 花子，
　　借りられた物 / 《花子の車》，
　　貸し出代金　/ 一万円，
　　貸し出し期間 / 一ケ月，
　　等々，....}

通常の項表現では,例えば，

貸す(太郎，花子，《花子の車》，一万円，一ケ月，等々，....)

となる．逆に通常の項を部分項であらわせることも明きらかであろう．

例2．文Kissing means touchingで表される制約は，たとえば，次のようにかくことができる．

```
{type/soa,rel/touching, agent/A, object/B}(-
            {type/soa, rel/kissing, agent/A, object/B}
```

例3．appendプログラムを部分項を用いて表現する．

```
append({},X,X).
append({X|Y},Z,{X|U})(-append(Y,Z,U).

{rel/append,arg1/{},arg2/X,arg3/X}.
{rel/append,arg1/{car/X, cdr/Y},
            arg2/Z,
            arg3/{car/X, cdr/U}}(-
        {rel/append, arg1/Y, arg2/Z, arg3/U}.
```

簡単のために必要な引数位置名全体を自然数の,ある有限集合{1,2,...,n}と同一視し，そして $ を次数nの特別なファンクタとする.すると部分項を通常の項の表現

$(A1,A2,...,An)

の形($-項と呼ぶ)へ自然に翻訳ができることが分かる．この対応をΦとする．例えば引数位置の最大値を4として

Φ({1/a, 3/{2/b, 4/c}}) = $(a,_,$(_,b,_,c),_)

である.ここで "_" の各出現はそれぞれ異なる新変数を表す.

すなわち(本稿では)部分項をこの$-項の略記と定義する.この略記法で注意しなければならないことがある.それは,一般に,異なる$-項が同一の部分項に応ずる事である.

実は,部分項の概念は木(tree)として数学的にきちんと定義することができる.ここで木とは有限列の集合で先頭部分列をとる操作に関して閉じているものである.そうし

て部分項そしてその間の単一化の概念はそれ独自の存在意義を持つことがわかる.詳しくは付録2にゆずり,ここでは通常の項概念に帰着させて説明している.

5. 単一化

二つの項を等しくするような代入(substitution)を求めることを単一化と言った.例えば項f(X,X,Y)とf(A,1,2)の単一化の場合,解は X=A=2,Y=2であった.上述のように部分項は通常の項として解釈されたからその間の単一化もそれに帰着して定義される.例えば二つの部分項{a/X, b/X}と{b/1, c/2}の単一化の結果は部分項{a/1, b/1,c/2}であり変数Xの値は1である.もう一つの直観的な解釈としてこの単一化は,部分項を,上で与えた対応Φの意味で,対応する $-項のすべてからなる集合と同一視するときの共通部分をとる集合演算になっている.次に構文論と意味論における単一化を見る.

文法は素性の束に関する制約を句構造規則の形で与えるものであり,それぞれ部分項およびホーン節を対応させることができる.また,素性共起制約などを遅延実行制御に対応させる.例えばGPSGの素性共起条件は後述の簡単な制約言語の範囲で容易に記述できる.

状況(意味)論によれば,自然言語の意味は以下のオブジェクトから構成される: 個体,ロケーション,関係,極,不定項(パラメータ),割り当て(部分解釈),条件, 事態(state of affair), 状況(事態の集合).事態は関係と割り当ての対である.(状況理論は,関係,引数位置, までもオブジェクトと考えているがこれはロジック・プログラミングの中のメタ・プログラミングと対応させて考えることができるだろう.) 事態は適当な代数的構造が入っている.そして有用な演算のひとつとして事態のマージがある.そして事態と部分項の間にはそれぞれマージと単一化に関して同型であるよう自然な対応を与えることができる(前掲の貸すの例).他のオブジェクトは事態の表現が決まればほぼ一意的に表現がきまってしまうことが分かる.

すなわち,構文論,意味論,そして記述言語(CIL)のそれぞれの領域の同型な単一化構造の存在を示した.これは,構文論と意味論の区別が基本的には,存在しないのだということ強く示唆しているように見える.

6. 遅延実行制御

CILの遅延実行制御について説明する.構文論や意味論の制約を効率良く解釈するための一般的な制御手段は難しい.われわれの出発点は,簡単な制約言語を設定し,それに対する遅延実行解釈を与えることである.その制約言語は ① 等式・不等式および基本算術述語(=:=, =\=, <) から論理結合詞(and, or, not, imply他)で結合した論理式と②制約の定義のための機構を含んでいる.この遅延解釈の手続きではバックトラックしない.また,このインタープリタに制御モード(① チェックするだけ, ② 正しい代入を遅延的に探せ,③否定を正しくするような代入を遅延的に探せ) を指示することができ,そして結果のモード(① 未定 ② 正しい ③ 正しくない)を返させる事ができる.このような簡単な制約言語でも応用は十分広いと期待している.より詳しい説明は付録3を参照のこと.

基本遅延制御としてA.Colmerauerのfreezeを用いている.PSIマシンでは bind_hookの名で提供されている.今,Xを変数としGをゴールとする.そのとき freeze(X,G)は「Xが値を持たなければXが値を持ったときにただちにGが実行できるようにして次の実行にうつれ.Xが値を持っている場合は次のゴールをGとして実行せよ.」と手続き解釈されるものである.

さて,ロジック・プログラミングの導出状態は,ゴールと環境の対であった.そして,り,環境の要素を等式とは限らずに,たとえば,不等式などに拡張することにより,一般の制約プログラミングが定義できる.そして,実数上の線形不等式の範囲では,かなり程度実現されている(Jaffar他86).本稿の方法は,そのような,一般の制約プグラミング言語を目指したものではない.むしろ,freeze機能の上に,遅延的な制御機能を実現するための,具体的手法を開発することが目的である.

7. CILの諸記法

現在のCILをPrologと比較するとき,以下の諸記法の導入が主要な特徴点である.な

お，付録1にCILのプログラム例を示す．

① 引数指示記法　部分項B の引数位置名Pに置かれている値をB!Pと書く．
例：　{a/1,b/X}!b=3 =〉 X=3

② 条件付き項　「条件Cをみたす X」を項 X:C と書く．条件付項X:Cの単一化，T=(X:C) は「XとTを単一化し，ただちにゴールCを解け．」と手続き解釈される．

③ 遅延制御記法　ゴール中のリテラルにあらわれる変数に"?"が付いているときはその変数の値が束縛されるまで実行が遅らされる．
例．print(X?) は freeze(X, print(X)) と同値である．

④ 同格記法 X#Y　は X:(X=Y)の略記である．この記法もたいへん便利である．特殊な例であるが，表現 X#{a/1,b/X}　はループ(-)b-〉...-)b-〉...)を持つグラフを表している（CILはサイクルを含む有向グラフあるいは自己参照型の状況の項表現の必要性を考慮して無限木を許す立場を採っている）．

⑤ 遅延実行型条件付項 @pとX@pはそれぞれ M:p(M?)と X:p(X?)と等価な項表現である．

8．関連研究等

DCKR〔Tanaka 86〕は通常の項表現の範囲で知識表現の可能性を追及している．DLOG〔Goebel 85〕，項記述〔Nakashima 85〕は単一化を記述(description)に対して拡張している．PATR-II〔S. Shieber 他86〕は単一化を明確に意識した文法フォーマリズムである．CIL〔Mukai他85〕は単一化を拡張し，遅延的実行制御を強化することにより，単一化ベースの文法フォーマリズムをロジック・プログラミングに埋め込んだものであり，その意味では，DCG(F.Pereira他)の拡張である〔付録1〕．

9．　おわりに

ロジック・プログラミングが，自然言語の構文・意味の計算モデルの記述に対して親和性があることを，拡張された単一化の概念のもとに示した．そして親和性を具体的な実現を示すものとして，Prologの項の概念の拡張と遅延実行制御の一部の機能を含むPrologの拡張言語 CILを示した．本稿で述べた部分は現在 DEC-10 PrologとPSI-ESPにて稼働している．

今後の課題として，つぎのようなものがある：

① 部分項に否定を導入すること，disjunctionを導入することなど，プロパティの継承の導入など．
② 制約条件のインタープリタの強化．
③ 抽象データ型記述による状況の型の記述．
④ CILコンパイラの強化，デバッグ環境の開発等．

謝辞

本研究は，談話理解システムDUALSの記述言語CILの研究開発の一環として横井第2研究室室長の下に行われた．日頃，御討論頂く，横井室長はじめ，DUALSプロジェクトの同僚：杉村，瀧塚，木村，および奥西の各氏に感謝する．

参考文献

〔Shieber 86〕 S.M. Shieber: Introduction to Unification-Based Approaches to Grammar, Center for the Study of Language and Information, Leland Stanford Junior University, 1986.

〔Pereira 他80〕 F.C.N. Pereira and D.H.D. Warren: Definite Clause Grammar for Language for Analysis - A Survey of the Formalism and Comparison with Augumented Transition Networks, Artificial Intelligence 13:231-278, 1980.

〔Tanaka 86〕 H. Tanaka: DCKR—Knowledge Representation in Prolog and its

Application to Natural Language Processing, In the Proceedings of International Symposium on Lnaguage and Artificial Intelligence, at Kyoto March 16-21 1986.

[Goebel 85] R. Goebel: The Design and Implementation of DLOG, a Prolog-based Knowledge Representation System, New Generation Computing, Vol.3 No.4, 1985.

[Jaffar et al 85] J. Jaffer, J-L.Lassez and M.J. Maher: A Logic Programming Lnaguage Scheme, in D. Degroot and G. Lindstrom(Eds.), Prentice-Hall, 1985.

[Jaffar 他 86] J. Jaffar and J.L. Lassez: Constraint Logic Programming August, IBM Thomas J. Watson Research Center, 1986.

[Nakashima 85] H. Nakashima: Term Description: A Simple Powerful Extension to Prolog Data Structure, in the Preceedings of the ninth IJCAI, 1985.

[Mukai et al 85] K. Mukai and H. Yasukawa: Complex Indeterminates in Prolog and its Application to Discourse Models, New Generation Computing, Vol.3. No.4, 1985.

付録１．

〔例題1 対話プログラム〕

つぎの例題は，簡単な対話プログラムである．構文および意味情報を統一的に取り扱うというアイデアを具体的に示すことが目的である．その主な処理内容は，格助詞，受け身と使役の助動詞および質問の助詞など，初歩的な部分である．構文情報も一種の状況（文法的事態と呼ぶ）として扱う．格助詞も助動詞も，文法的な状況の間のコンストレイントとして記述している．たとえば，《本》-《を》-《買う》における《を》のはたらきの分析は，《本》と《買う》の間の"スロット・フィリング"関係として分析する．

文の解析はプッシュダウン・スタック方式である．まず文節をまとめて，つぎに文節内の係り受けを処理する．こたえを探すためのパターンマッチ処理は，部分項を用いて記述した．部分項の柔軟な表現力を示しものである．

プログラム中の略記号

```
gsoa : 文法的事態(grammatical state of affair)
soa  : 事態(state of affair)
ass  : 割り当て(assignment)
cond : 条件(condition)
agt  : 動作主体(agent)
recip: 受け手(recipient)
obj  : 対象物(object)
aux  : 助動詞(auxiliary)
xobj : 修飾名詞句
rel  : 関係(relation)
var  : 変数(variable)
```

```
%メイン
dialog:-dialog([]).

dialog(X):- read(Y), interpret(Y,A), dialog(A,X).

dialog([type/soa, rel/question, ass/[obj/A]],X):-!,
        find_answer(A,X,B),
        nl,print(B),
        dialog(X).
dialog([type/soa,rel/exit],_):-!, nl, print('SAYOUNARA').
dialog(X,Y):- nl, print(X), dialog([X|Y]).

%解の探索
find_answer(X,[Y|_],A):-p_match(X,Y,A).
find_answer(X,[_|Y],A):-find_answer(X,Y,A).

%照合処理:変数に対応する部分を求める．
p_match([type/soa,rel/R,ass/[agt/[var/yes],obj/[name/O]]],
        [type/soa,rel/R,ass/[agt/K,        obj/[name/O]]],
        K):-!,
p_match([type/soa,rel/R,ass/[agt/[name/N],obj/[var/yes]]],
        [type/soa,rel/R,ass/[agt/[name/N],obj/O]],
        O):-!,
p_match([type/soa,rel/R,ass/[agt/[name/N],obj/[name/O]]],
        [type/soa,rel/R,ass/[agt/[name/N],obj/[name/O]]],
        yes):-!,
p_match(_,_,no). %探索失敗

%単語列の解釈
interpret(Words, X):-interpret(Words, [], [X]).
```

```
interpret({}, Xs, Xs):-!.
interpret(Words, Xs, NewXs):-
        phrase(Words, X, RemWords),
        mother(X, Xs, MotherX,  RemXs),
        interpret(RemWords, [MotherX|RemXs], NewXs).

%文節の解釈
phrase([Word|TailWords], X, RemWords):-
        lexical_item(Word, D),
        objective(D),
        phrase(TailWords, D, X, RemWords).

phrase([],X,X,[]):-!.
phrase([Word|TailWords], X, NewX, RemWords):-
        lexical_item(Word, XX),
        mother(X, XX, Mother),
        phrase(TailWords, Mother, NewX, RemWords).
phrase(Words, X, X, Words).

%自立語の定義
objective([type/soa]).
objective([type/obj]).
objective([type/xobj]).

%スタックの先頭部分との係り受け処理
mother(X, [], X, []):-!.
mother(X, [Y|Z], U, V):-
        mother(Y, X, W),
        mother(W, Z, U, V).
mother(X, Y, X, Y).

% 主要部(head)と補部(complement)の干渉処理
%
% mother(〈Left〉, 〈Right〉, 〈Mother〉) ⇔
% 句構造規則 "〈Mother〉 --> 〈Left〉, 〈Rigth〉".
%
% (くるま　を)+(かう)
mother(
        X#[type/gsoa,rel/R,head/Y,comp/C],
        Y#[type/soa, mark/[R/C]],
        Y):-!, locate(M,R,C).
        % Y):-!, locate(M,R,C).
        % locate(M,R,C): 組み込み述語. M,R,Cをそれぞれ連想リスト,
        % キィ-,および結果として,Mの中をサーチする. Mがキィを持た
        % ないときは失敗する.この点は下のgetRoleと異なる.

% (くるま) + (を)
mother(
        X,
        Y=[type/gsoa, comp/X],
        Y):-!.

% (かう) + (られる) (受け身形)
mother(
        X#[mark/Given],
        Y#[type/aux, given/Given, result/Result],
        Z#[mark/Result]):-!, delete(mark,X,Z).
        % deleteは組み込み述語で, delete(mark, X, Z)はXから
        % markスロットをとり除いたものとZをマージする.
```

```
% 連体修飾句: (はなこ が かう)+(くるま)
mother(
        X#{type/soa, ass/Ass},
        Y#{type/obj},
        {type/xobj, xobj/Y, cond/X}):- fillSlot(Ass, Y).

% 連体修飾句: (たろう が  すく)+ ((はなこ が すく)+(くるま))
mother(
        X#{type/soa,ass/Ass},
        Y#{type/xobj},
        {type/xobj, xobj/Y, cond/X}):- fillSlot(Ass, Y).

% (たろう  が  かう)+(か)
mother(
        X#{type/soa},
        Y#{type/soa, ass/{obj/X}},
        Y).

% スロット・フィリング
fillSlot(Y,A):-getRole(Y,K,V), unbound(V),!,V=A.
fillSlot(_,_).

%辞書引き
lexical_item(X,Y#{var/V}):-lex1(X,Y),vardefault(V).

%単語のデフォルトは非変数
vardefault(no):-!.
vardefault(_).

%登録されていない単語は人名とする.
lex1(X,Y):-dict(X,Y),!.
lex1(X, {
        name/X,
        type/obj,
        rel/human
        }).

%単語辞書
dict(sayounara,{type/soa,rel/exit}). % さようなら
dict(dare, {            %変数 誰
        type/ obj,
        rel/human,
        var/yes
        }).
dict(nani, {            %変数 何
        type/ obj,
        var/ yes
        }).
dict(kuruma, {          %名詞 くるま
        type/ obj,
        rel/car
        }).
dict(wo, {              %格助詞 を
        type/gsoa,
        rel/wo}).
dict(ga, {              %格助詞 が
        type/gsoa,
        rel/ga}).
dict(ni, {              %格助詞 に
        type/gsoa,
        rel/ni
```

```
        }).
dict(kara, {            %格助詞から
        type/gsoa,
        rel/kara
        }).
dict(de, {              %格助詞 で
        type/gsoa,
        rel/de
        }).
dict(rareru, {          %助動詞 られる
        type/aux,       %  甲が乙に丙られる >乙が甲を丙する.
        rel/rareru
        given/ A#{ga/X, wo/Y},
        result/B#{ga/Y, ni/X}
        }):-masked_merge(A,[ga/_,wo/_],B).
        %Aからgaおよびwoスロットを取り除いたものをBにマージする.
dict(kau, {             %動詞 かう
        type/soa,
        rel/buy,
        ass/{agt/A,obj/0},
        mark/{ga/A,wo/0}}).
dict(uru, {             %動詞  うる
        type/soa,
        rel/sell,
        ass/{agt/A,obj/0},
        mark/{ga/A,wo/0}}).
dict(suku, {            %動詞  すく
        type/soa,
        rel/like,
        ass/{agt/A,obj/0},
        mark/{ga/A,wo/0}}
dict(saseru, {          %助動詞  させる（動詞と同じ扱い）
        type/soa,
        rel/cause,
        ass/ {agt/A, obj/{type/soa, ass/{agt/R}}, recip/R},
        mark/{ga/A, ni/R}
        }).
dict(ka, {              %助詞  か       （動詞と同じ扱い）
        type/soa,
        rel/question,
        ass/{obj/{type/soa}}
        }).
```

%対話例

```
| ?- dialog.
|: {taro, ga, kau, kuruma, ga, hanako, ni, suku, rareru).
        %(たろうが  かう  くるまが  はなこに  すかれる)

x(_5273), WHERE
    x(_5273)={ass/x(_3468),mark/x(_4125),rel/like,type/soa,var/no},
    x(_3468)={agt/x(_2778),obj/x(_1965)},
    x(_1965)={cond/x(_1773),type/xobj,xobj/x(_1889)},
    x(_1889)={rel/car,type/obj,var/no},
    x(_1773)={ass/x(_1014),mark/x(_974),rel/buy,type/soa,var/no},
    x(_974)={ga/x(_413),wo/x(_1889)},
    x(_413)={name/taro,rel/human,type/obj,var/no},
    x(_1014)={agt/x(_413),obj/x(_1889)},
    x(_2778)={name/hanako,rel/human,type/obj,var/no},
    x(_4125)={ga/x(_1965),ni/x(_2778)}
        % x(_)の形は内部変数.
```

```
|: [hanako, ga, nani, wo, suku, ka]. %(はなこが　なにを　すくか)

%応答
x(_1965), WHERE
    x(_1965)={cond/x(_1773),type/xobj,xobj/x(_1889)},
    x(_1889)={rel/car,type/obj,var/no},
    x(_1773)={ass/x(_1014),mark/x(_974),rel/buy,type/soa,var/no},
    x(_974)={ga/x(_413),wo/x(_1889)},
    x(_413)={name/taro,rel/human,type/obj,var/no},
    x(_1014)={agt/x(_413),obj/x(_1889)}
        %（たろう　が　かう　くるま）という内容を表わしている.
|: [sayounara].

SAYOUNARA
```

[例題 2　状況を用いた談話の解釈のモデル]

```
% 談話状況(DISCOURSE SITUATION)の定義
discourse_situation({sit/S, sp/I, hr/ You, dl/ Here, exp/ Exp}):-
        member(soa(speaking, (I, Here),yes),S),      % sp :  話者
        member(soa(addressing,(You, Here),yes),S),   % hr :  聴者
        member(soa(utter,(Exp, Here),yes), S).

% メンバの定義
member(X, [X|Y]).
member(X,[Y|Z]):-member(X,Z).


% 談話制約(DISCOURSE CONSTRAINT)の定義
discourse_constraint([],[]):-!.
discourse_constraint([X],[Y]):-!,meaning(X,Y).
        % Y is an interpretation of Y
discourse_constraint([X,Y|Z], [Mx,My|R]):-
        meaning(X,Mx),          % Mx: 談話状況の解釈
        change_role(X, Y),    % 聞き手と話し手が反対になる.
        time_precedent(X, Y), % X は Yに時間先行する.
        discourse_constraint([Y|Z],[My|R]).

change_role({hr/X,sp/Y},{hr/Y,sp/X}@discourse_situation).

time_precedent({dl/loc(X)},{dl/loc(Y)}):-
        constr(X+1=:=Y).
                % constrは遅延制御実行用の組み込み述語

% 文法 - DCG(DEFINITE CLAUSE GRAMMAR) -
meaning(X#{exp/E},Y):-sentence(E-[],{ip/Y,ds/X}).          % ip  : 解釈

sentence(A-B, {ip/SOA,ds/DS}):-
        noun(A-A1,  {ip/Ag,ds/ DS}),
        verb(A1-A2, {ip/SOA, ds/DS, ag/Ag, obj/ Obj}), % ag : 行為者
        noun(A2-B,  {ip/Obj, ds/ DS}).                  % obj: 対象者

% 語彙項目
noun([{jack|Q}-Q, {ip/jack}).          % Jack
noun([{betty|Q}-Q,{ip/betty}).         % Betty
noun([{i|Q}-Q,    {ip/X, ds/{sp/X}}). % I
noun([{you|Q}-Q,  {ip/X, ds/{hr/X}}). % You
verb([love|Q]-Q, {ip/ soa(love,(X, Y, Loc), yes), % love
```

```
                ds/ {dl/Loc},
                ag/ X,
                obj/Y}).

% 実行および結果

% 同じ文が談話状況(discourse situations)によって
% いろいろな解釈(interpretation)を持つ.

?- discourse_constraint(
              {{sit/ {soa(speaking,  (jack, _), yes),
                      soa(addressing,(betty, _),yes)|_},
                exp/ {i,love,you},
                dl/  loc(1)}@discourse_situation,
               {exp/ {i,love,you}}@discourse_situation},
              Interpretation).

   % sit: 状況(situation)
   % soa: 事態(state of affair)
   % exp: 表現(expression)
   % dl : 談話時空領域(discourse location)

%結果
Interpretation =
    {soa(love,(jack,betty,loc(1)),yes),    % 文1   "I love you."
     soa(love,(betty,jack,loc(2)),yes)}    % 文2   "I love you."
```

付録2

部分的に指定された項の意味論

要旨

部分的にタグ付けされた木の領域(Partially Tagged Tree)の上の制約論理プログラム(Constraint Logic Program)言語を新しく定義する．これは，Jaffar 他 86 のCLPの新しい具体例と見ることができる．本言語は通常のホーン節の拡張であり，論理の整合性および完全性も成り立つ．本言語の応用として単一化ベースの文法形式(Unification-based Grammar Formalism)の意味論を与える．

0. はじめに

Prologを拡張して，単一化ベースの文法形式，状況など意味・語用論的オブジェクトを統一的に表現するための試みがある．Prologの拡張言語CIL〔Mukai 他 85〕も，そのような試みの例である．その過程で，レコード形式のデータ構造をPrologに導入することが，構文論，意味論，そして語用論の処理上，多くのことに，統一的な視点を与えることが明らかになった．

本付録では，CILの，中心的なサブセットのモデルを示す．まず，部分的にタグ付けされた木(PTT)の領域を導入し，つぎに，その上の制約論理プログラムを定式化し，その整合性と完全性を調べる．

最近，DCG，PATR-II，LFG，あるいは，GPSGなど，いくつかの有力な文法理論があらわれている．そして，それら間の基本的な違いは，記法上のちがいであるというコンセンサスが形成されつつある〔Shieber 他 86〕．本稿は，制約論理プログラム〔Jaffar 他 86〕の枠組み(CLP)を用いて，その見方を支える一つの根拠を与える．本言語の応用として，PATR-II〔Shieber 他 86〕の完全性と整合性を論じる．

本稿は Ait-Kaci 84と，独立になされたのであるが，関連性が非常に高いと思われる．Ait-Kaci 84は，より一般的に，束論的領域を用いている．われわれの場合は，PTTという具体的な意味領域を用いているので，完全性に関する結果を得ている．Ait-Kaci 84では，その点は，まだ未解決である．しかしながら，詳細な理論的比較は今後の課題である．

また，素性の構造に「否定」および disjunction を導入するという新しい試みがあり（たとえば Kasper他 86，文法形式上，重要なトピックスであるが，それは本稿の範囲ではない．

1. タグ付けされた木

要素 a1，...，anからなる列を〈a1，...，an〉と書く．その長さはnである．空列を〈〉と書く．空列の長さは0である．

定義．（列の接合 *）列の接合演算を次の等式で定義する：

〈〉*x=x，
x*〈〉=x，
〈a1,...,an〉*〈b1,...,bm〉=〈a1,...,an,b1,...,bm〉．

定義．（木）有限列からなる非空集合Tが接頭部分をとる演算で閉じているときTを木という．すなわち，x*y∈Tならば x∈Tである．木の要素を節点という．木は，有限集合と限っていない．

木は，定義より，空列〈〉を必ず含むことがわかる．Tの節点xの分岐の全体

{{y∈T | ∃z x*y∈Tかつy=〈a〉*z} | aはラベル}

が各点で有限とは，限っていない．

〔Fig.1　参照のこと〕

定義.（葉節点）　木Tの節点xは,任意のyについて，y=x*zならばzが空列(⟨⟩)であるとき，xをT の葉節点という．　木Tの葉節点全体をLEAF(T)で示す．

〔Fig.2　参照のこと〕

定義．T1,T2を木として,T1⊂T2のときT2はT1の部分木という．

命題1.1．　二つの木T1とT2の和T1∪T2は木である．T1とT2の積T1∩T2も木である．
証明　自明．

定義．　列 x，木Tに対してx*T={z | ∃u ∃y∈T z*u=x*y}と定義する．

x*Tは木であることがただちにわかる．

(例)　⟨a⟩*{⟨⟩,⟨b⟩,⟨c⟩,⟨c,d⟩}={⟨⟩, ⟨a⟩,⟨a,b⟩,⟨a,c⟩,⟨a,c,d⟩}

〔Fig.3 参照のこと〕

命題1.2．任意の木Tに対して,ラベルa1,...,anと木T1,...,Tnが存在して，T=a1*T1∪...∪an*Tnとなる．またこのような表現は一意的である．n≧0は,必ずしも有限ではない．

証明 L={a | ∃x⟨a⟩*x∈T},a∈Lに対してTa={x | ⟨a⟩*x∈T}とおく．
T= ∪{a*Ta | a∈L}．a*Ta∩b*Tb={⟨⟩} (a≠b)である．また，
x*A⊂Tならば，あるa∈Lが存在してx=a，A⊂Taであることは容易に分かる．
これを用いて，一意性がいえる．
QED

このとき，T=a1*T1+...+an*Tn と書くことがある．

〔Fig.4 参照のこと〕

定義．tを木とする．パス式α=⟨a1, ..., an⟩にたいして，t//α を帰納法により，次のように定義する．ただし，t=a1*t'+ ...., とする:
t//⟨a1, ..., an⟩ = t'//⟨a2, ..., an⟩.

点対の集合fは,(x,y),(x,z)がともにfの元ならばy=zであるという性質を持つとき関数という．

定義.(関数のマージ)関数fとgの和集合がふたたび関数hであるときfとgは両立可能であるといい,hをfとgのマージとよぶ．

(例) f={(a,1),(b,2)}, g={(b,2),(c,3)}ならばh={(a,1),(b,2),(c,3)}はfとgのマージである．　一方,関数{(a,1),(b,2)}と {(b,4),(c,3)}のマージは存在しない．

〔Fig.5 参照のこと〕

定義．（タグ付きの木）木Tとその木の葉節点からなる集合を定義域とする関数f,すなわち dom(f)⊂LEAF(T)なる関数fとの対(T,f)を(部分的に)タグ付けされた木(Partially Tagged Tree, PTT)といい，またfを(T,f)のタグという．dom(f)=空集合であってもよい．タグ付きの木 ({⟨⟩},φ)を自明なタグ付き木という．

〔Fig.6 参照のこと〕

また({⟨⟩},{(⟨⟩,a)})をaと略記することもある．そして木の場合に導入した略記法をPTTの場合にも拡張しておく．たとえば，上例のタグ付きの木tを
t=a*(b*《未定義》 + c*1) + d*(c*2 + b*3)
などと書くことができる．

定義．t1=(T1,f1)とt2 = (T2,f2)をタグ付きの木として，t=(T1∪T2, f1∪f2)がふたたびタグ付きの木になるときおよびそのときに限りtをt1とt2のマージといい，
t=MERGE(t1,t2)と記す．あるいは
t=t1+t2
とも書く．また，t1≦t2 ⇔ T1⊂T2かつf1⊂f2 とPTTの順序を定義する．

自明なタグ付きの木は，このマージ演算に関して，単位元として振る舞う．

(例) t1=({⟨⟩,⟨a⟩},{(⟨a⟩,1)}),t2=({⟨⟩,⟨b⟩},{(⟨b⟩,2)})のときMERGE(t1,t2)
={{⟨⟩,⟨a⟩,⟨b⟩},{(⟨a⟩,1),(⟨b⟩,2)}}である．

MERGE({{⟨⟩,⟨a⟩},φ),({⟨⟩,⟨a⟩,⟨a,b⟩},{⟨a,b⟩,1}))={{⟨⟩,⟨a⟩,⟨a,b⟩},{(⟨a,b⟩,1)}).
任意のタグ付きの木tについて，明らかにMERGE(t,({⟨⟩},φ))=tである．
MERGE(t1,({⟨⟩,⟨a⟩},{(⟨a⟩,3)}))は，定義されない．
MERGE(({⟨⟩},{(⟨⟩,4)}),({⟨⟩,⟨a⟩},φ))も，定義されない．

〔Fig.7 参照のこと〕

注意．(T, g) gをTで定義された関数で，{m | υ*⟨m⟩ ∈ T}=arg(g(υ))とすれば，普通の無限木の定義となる．

命題1.3． PTTの領域はcompleteである．
証明 容易である．

2． 部分的に指定された項

記号として，無限個のラベル，変数，定数記号を固定する．部分的に指定された項(部分項)を次のように定義する．補助記号として{,},/,,,(,)が許されている．
述語記号も関数記号もないことに注意する．等号記号(=)もない．

定義． (部分的に指定された項，Partially Specified Term: 部分項，PST)
(a)定数は部分項
(b)変数は部分項
(c)a1,...,anが相異なるラベル，p1,...,pnが部分項のときラベルと部分項の対の集合 {(a1/p1),...,(an/pn)}は部分項
(d)以上のみが部分項である．

上の(c)の場合の部分項を特に真の部分項と呼ぶ．

定義．部分項に関する諸定義
KEYS({a1/x1,...,an/xn})={a1,...,an},
ASSOC(a,M)=x: (a/x)∈Mの場合，
=未定義:その他，
MERGE(X,Y,E)=(E-{X,Y})∪{X∪Y},
PAIRS(x,y)={ASSOC(a,x)≡ASSOC(a,y)|a∈KEYS(x)∩KEYS(y)},
VARS(x):xに出現する変数の全体集合．

定義．pを部分項とする．パス式α=⟨a1, ..., an⟩にたいして，p//αを帰納法により，
p//⟨a1, ..., an⟩ =ASSOC(a1, p)//⟨a2, ..., an⟩
と定義する．

(例)
PAIRS({a/x},{a/1,b/y})={x≡1},
PAIRS({a/1},{a/2,b/y})={1≡2}.

定義．(FRONTIER)
FRONTIER(x,x)=φ :xが定数の場合，
FRONTIER(x,y)={x≡y}: x,yの少なくとも一方が定数のとき，

FRONTIER(x,y)={x≡y}: x,yの少なくとも一方が変数のとき，
FRONTIER(x,y)=∪{FRONTIER(u,v) | u≡v ∈ PAIRS(x,y)}:その他の場合．

(例)
FRONTIER({a/2,b/{c/x}},{a/3,b/y})={2≡3, {c/x}≡y}.

3． 等式と環境

定義．変数に対してタグ付きの木を対応させる部分関数を割り当て(assignment)と呼ぶ．

ρを割り当てとする． ρを部分項に対しても拡張定義する．kを定数,aiをラベルとする．
(a) ρ(k)=({⟨⟩}, {(⟨⟩,k)}),
(b) ρ(φ)=({⟨⟩},φ),
(c) ρ({a1/p1,...,an/pn})= a1*ρ(p1)+...+an*ρ(pn).

(例) ρを任意の割り当てとして，
ρ({a/1,b/{c/2}})=({⟨⟩,⟨a⟩,⟨b⟩,⟨b,c⟩},{(⟨a⟩,1),(⟨b,c⟩,2)}).
割り当てρが，ρ(x)=({⟨⟩},φ),ρ(y)={{⟨⟩,⟨c⟩}, {(⟨c⟩,3)}}であるとき，
ρ({a/x,b/y})= ({⟨⟩,⟨a⟩,⟨b⟩,⟨b,c⟩},{(⟨b,c⟩,3)}) である．

〔Fig.8 参照のこと〕

部分項の意味論の基礎となるのは，等式の解という概念である．その準備として，照合割り当てを定義する．

定義．(照合割り当て)pを部分項,tをタグ付き木とする.割り当てθがつぎの条件を満たすとき，θをpからtへの照合割り当てという．pからtへの照合割り当ての全体集合をmatch(p,t)と書く．

(1) pが変数のとき：θ(p)=t，
(2) pが定数のとき：θ(p)=t，
(3) p={a1/p1,...,an/pn}のとき： t=a1*t1+...+an*tn+s，かつ，各i(1≦i≦n)について，θがpiからtiへの照合割り当てであるようなPTT t1, ..., tn, sが存在する．

(例) xを変数として，θ， ρ， p， tをつぎのようにおく．θ(x)=c*1+d*2，ρ(x)=c*1，p={a/x},そして t=a*(c*1+d*2)+b*(c*3+f*4)．このとき，θはxからt への照合割り当てであるが，ρはそうではない．

〔Fig.9 参照のこと〕

定義．(等式の解)割り当てρが等式p≡qの解であるとは,帰納的に,次の条件が成り立つことである．
(a) pが変数のときρ∈match(q,ρ(p)),
(b) qが変数のときρ∈match(p,ρ(q)),
(c) pあるいはqが定数のときρ(p)=ρ(q),
(d) p,qが定数でも変数でもない場合，ρはPAIRS(p,q)の各要素の解である．

割り当てρが等式系Pに含まれるすべての等式の解であるとき，ρをPの解という．Pの解全体集合をsol(P)と書く．

補題3.1. tをタグ付きの木，θを割り当て，pとqを部分項とする．θがpからtへの,そして同時に，qからtへの照合割り当てならばθはp≡qの解である．すなわち，
match(p,t)∩match(q,t)⊂sol({p≡q})である．

証明
pとqのそれぞれの高さの最小値hに関する帰納法によって示す．
1) h=1とする．pとqがともに変数の場合．θ(p)=θ(q)=tとなるから，θはp≡qの解である．pが変数のとき，θ(p)=tだから，θ∈match(q,t)=match(q,θ(p))．ゆえに，θはp≡qの解である．qが変数の場合も同様である．pとqが定数の時は，与えられた条件より， θ(p) = θ(q) = t であるから，θはp≡qの解である．

2) h(kについて定理が成り立っているとする．pあるいはqが変数あるいは定数ならば 1)と同様である．したがって，一般性を失うことなくpとqが真の部分項と仮定してよい．
あるラベルaについて，(a/u)∈p，(a/v)∈q，t//(a)=t'とすると，

$$\theta \in match(u,t') \cap match(v,t')$$

である．すると，帰納法の仮定より，θは u≡v の解である．u≡vはPAIRS(p,q)の任意の元であったから，　θ∈sol(PAIRS(p,q))=sol({p≡q}) が証明された．

QED.

補題3.2　ρがp≡qの解ならば　ρ∈match(p,ρ(p)+ρ(q)).
証明
(1) pあるいはqが定数の場合は明らかである．
(2) pが変数ならばρ(p)=ρ(q)であるからρ(p)=ρ(p)+ρ(q)．ゆえに，

$$\rho \in match(p,\rho(p)) = match(p,\rho(p)+\rho(q))$$

である．
(3) qが変数ならばρ(q)=ρ(p)であるからρ(q)=ρ(p)+ρ(q)．ゆえに，

$$\rho \in match(p,\rho(q)) = match(p,\rho(p)+\rho(q))$$

である．
(4) p,qともに真の部分項のときpとqの高さの最小値hに関する帰納法を用いる．
ρ∈sol({p≡q})の定義より，ρ∈sol(PAIRS(p,q))である．
p'≡q'∈PAIRS(p,q)とすると，あるラベルaについて，(a/p')∈p，(a/q')∈q，
ρ∈sol({p'≡q'})である．
h=1のときは，p',q'のいずれかは変数あるいは定数であるから，既に示したように

$$\rho \in match(p',\rho(p')+\rho(q')).$$

h>1のときは，帰納法の仮定より，

$$\rho \in match(p',\rho(p')+\rho(q')).$$

ゆえに　ρ∈match({a/p'},a*(ρ(p')+ρ(q')))である．照合割り当ての定義により，
ρ∈match(p,ρ(p)+ρ(q))である．

QED

定義．部分項からなる集合の集合Eで，各変数xについてx∈w∈Eなるwが高々一つ*存在*するとき，Eを環境という．環境E，変数xにたいしてx∈w∈Eなるwが*存在*するとき
w=CLASS(x,E)と書く

....(例) {{x},{y,z,{a/φ}},{u,{b/1,c/y,d/x}}},

定義．(合同的) 任意の要素w∈E，p,q∈wについて u≡v⊂FRONTIER(p,q) ならば，あるw'∈Eが*存在*してu∈wかつv∈w'であるとき，Eは合同的であるという．

定義．ρを割り当て，Eを環境とする．任意のw∈E,x∈w,y∈wについて，ρがx≡yの解であるとき．ρはEの解であるという．Eの解の全体集合をsol(E)と書く．

定義．すべてのw∈E,x∈w,y⊂wについて，θ(x)とθ(y)はマージ可能であるとする．このような割り当てθをEと両立するという．

定義．割り当ての単調列θ0,...,θn,...,すなわち，θn(x)≦θn+1(x)にたいして
θ(x)=sup{θn(x)｜n≧0}で定義される変数割り当てをθ= sup{θn｜n≧0}と書く．

命題3.3．割り当ての単調列θ0,...,θn,...,および部分項pにたいして

$$sup\{\theta n(p) \mid n \geqq 0\} = (sup\{\theta n \mid n \geqq 0\})(p)$$

である．

証明　θ=sup{θn｜n≧0}とおく．pについての，構造帰納法を用いる．

① pが定数のとき：両辺ともにpであるから命題が成り立っている．
② pが変数xのとき：sup{θn｜n≧0}の定義により，
(sup{θn｜n≧0})(x)=sup{θn(x)｜n ≧0}

である．
③ p={a1/p1, ..., ar/pr}のとき
sup{ θn(p) | n≧0}= sup{ θn({a1/p1, ..., ar/pr}) | n≧0}
= sup{a1* θn(p1)+ ... +ar* θn(pr) | n≧0}
= a1* sup{ θn(p1) | n≧0}+ ... +ar* sup{ θn(pr) | n≧0}
= a1*θ(p1) +...+ ar*θ(pn) (帰納法を使った)
= θ(p)
QED

定義．両辺が異なる定数であるか，あるいは，一方の辺が定数で，他方の辺が真の部分項である等式をコンフリクトと呼ぶ．FRONTIER(x,y)がコンフリクトを含まないとき，xとyはコンフリクトなしであるという．
(例) {a/1,b/1}と{a/1,b/2}はコンフリクトをもつ．

命題3.4．Eを合同的な環境とする．Eがコンフリクトを持たなければ,Eは解を持つ．

証明　φをEと両立する，割り当てとする．Φ(φ)(x)=sup{φ(y) | y∈CLASS(x,E)}で定義される割り当てΦ(φ)が存在し，Eが合同的であることから，Eと両立する．
θ0, θ1,..., θn,...を帰納的に定義する： θ0(x)=未定義． θn+1=Φ(θn)．これらは

すべて，Eと両立する．またすべてのx について θn(x)≦ θn+1(x) である．したがって，θ(x)=sup{ θn(x) | n≧0}で定義される割り当てθが存在し，θはEと両立する．
θはΦの不動点である：

θ(x)= sup{ θn | n≧1}(x)
= sup{ θn(x) | n≧1}
= sup{Φ(θn-1)(x) | n≧1}
= sup{sup{ θn-1(y) | y∈CLASS(x, E)} | n≧1}
= sup{sup{ θn-1(y) | n≧1} | y∈CLASS(x, E)}
= sup{ θ(y) | y∈CLASS(x, E)}
= Φ(θ)(x)

ゆえにθ=Φ(θ)．ここで，PTTのマージが「連続」であることを使っている．

つぎにθが環境Eの解であることを証明する．w∈E, x∈wを変数，p∈wを部分項，そしてyをp//α=yなる変数とする．θの構成から，θ(p//α)≦θ(x)//αである．よって，θ(x)//α≦θ(y)であることを証明すれば十分である．Q⊂CLASS(y, E)を用いると，

θ(x)//α=sup{sup{ θn(q) | q∈Q} | n≧ 0} ≦ sup{sup{ θn(q) | q∈CLASS(y, E)} | n≧0} =θ(y).

ゆえにθ(x)//α = θ(y) である．
QED

4．単一化

定義．等式系Pと環境Eの対(P,E)を計算状態という．

定義．計算状態(P,E)に対してsol(P,E)=sol(P)∩sol(E)と定義する．

単一化アルゴリズムを定義する．そのために，与えられた環境Eのもとで部分項 pとqを単一化した直後の新しい環境Dと派生した等式の集合Qを定義する．

定義．(⟨D,Q⟩=NEXT(p,q,E)) 項p,qおよび，環境Eが与えられたとき，環境と等式の集合の対u=NEXT(p,q,E) をつぎのように定義する：

(a) u=(E, φ)： pが変数でq∈CLASS(p,E)のとき，
(b) u=(E, φ)： qが変数でp∈CLASS(q,E)のとき，
(c) u=⟨MERGE(CLASS(p,E),CLASS(q,E),E),
{u≡v| u∈CLASS(p,E), v∈CLASS(q,E)}⟩ ：

p,qがともに変数でかつCLASS(p,E)≠CLASS(q,E)のとき，
(d) u=(E,PAIRS(p,q)): p,qがともに真の部分項のとき，
(e) u=(E-CLASS(p,E))∪{CLASS(p,E)∪{q}}, {s≡q | s∈CLASS(p,E)}) :
pのみが変数のとき，
(f) u=NEXT(q,p,E):qのみが変数のとき，
(g) u=(E,φ): ともに定数でfail のとき，
(h) u=未定義: その他の場合．

(例) xとyを変数とする．Eを環境とする．
NEXT(1,1,E)=(E,φ).
NEXT(1,2,E)=未定義.
NEXT(x,y,{{x},{y}})=({{x,y}},{x≡y}).
NEXT(x,y,{{x,y}})=({{x,y}},φ).
NEXT({a/1,b/x},{b/y,c/2},E)=(E,{x≡y}).

定義．(計算状態遷移 =)) あるp,qについてp≡q∈A かつ NEXT(p,q,E)=(F,B)のときおよびそのときに限り (A,E)=)((A-{p≡q})∪B, F)と書く．

単一化アルゴリズム:
入力:等式系
出力:環境,あるいは「失敗」
アルゴリズム:

P0を与えられた等式系P0={e1,...,em}, E0を環境E0={{x1},...,{xn}}とおく．ここでx1,...,xnはe1,...,emに現れるすべての変数とする．計算状態の遷移関係 =) に関してつぎつぎと計算状態を求める:

(P0,E0)=)...=)(Pn,En)

最後の計算状態を(Pn,En)とする．Pn=φならば環境Enを出力する．さもなければ「失敗」を出力する．

(例1)X,Yを変数とする．
({{a/X,b/X}={b/Y,a/1}},{{X},{Y}})
=) ({{X≡1},{X≡Y}},{{X},{Y}})
=) ({X≡Y},{{X,1},{Y}})
=)(φ,{{X,Y,1}}).

(例2)X,Yを変数とする．
({X≡{a/Y,b/Y},Y≡{a/X,b/X},X≡Y},{{X},{Y}})
=)({X≡{a/Y,b/Y},Y≡{a/X,b/X},X≡Y},{{X,{a/Y,b/Y}},{Y}})
=)({Y≡{a/X,b/X},X≡Y},{{X,{a/Y,b/Y}},{Y}})
=)({Y≡{a/X,b/X},X≡Y},{{X,{a/Y,b/Y}},{Y,{a/X,b/X}}})
=)({X≡Y},{{X,{a/Y,b/Y}},{Y,{a/X,b/X}}})
=)({X≡Y,Y≡X},{X,Y,{a/Y,b/Y},{a/X,b/X}})
=)({Y≡X},{X,Y,{a/Y,b/Y},{a/X,b/X}})
=)(φ,{X,Y,{a/Y,b/Y},{a/X,b/X}})

〔Fig.10 参照のこと〕

命題4.1. 単一化アルゴリズムは停止する．

証明 アルゴリズムの実行途中に現われうる環境は，簡単な組み合わせ論的考察により，高々有限であることが分かる．また，環境は単調に粗くなっていくのみである．すなわち,w∈Eかつ (P,E)=)(P',E')ならばwはある w'∈E' に包含される:w⊏w'．従って，もし停止しないとすれば,ある自然数kから先では,環境が飽和する.すなわち,

...=)(Pk,Ek)=)(Pk+1,Ek)=)(Pk+2,Ek)=)...

となる.ここでj≧kならば，Pj≠φである．計算状態遷移の定義と環境が飽和すること

から次の①または②が成り立つ：

① $P_{j+1} \subset P_j$, $P_{j+1} \neq P_j$

② ある真の部分項p,qについて$p \equiv q \in P_j$となり，$P_{j+1}=(P_j-\{p \equiv q\}) \cup PAIRS(p,q)$となる．

このような条件を満たす等式の有限集合の無限列が存在しないことは Pkの要素数とPkが含む等式の大きさの最大数に関する2重帰納法により証明することができる．これは矛盾である．よって停止性がいえた．

QED

命題4.2．単一化は解集合を保存する．すなわち，
$(P,E)=\rangle(P',E')$ならば$sol(P,E)=sol(P',E')$
である．

証明 あるp,qについて$p \equiv q \in P$ かつ $NEXT(p,q,E)=\langle E',D\rangle$のときおよびそのときに限り $(P,E)=\rangle(P',E')$であった．ただし$P'=(P-\{p \equiv q\}) \cup D$．このNEXTの定義の場合分けに応じて証明する．

(a) pが変数で$q \in CLASS(p,E)$の場合：$E'=E$，$P'=P-\{p \equiv q\}$となる．
$sol(P,E) \subset sol(P',E)$は明らかである．$\theta \in sol(P',E)$ならば，pとqは，Eの同一の元に含まれから，θはp≡qの解である．したがって，θはPの解である．ゆえに，$\theta \in sol(P,E)$である．

(b) qが変数で$p \in CLASS(q,E)$のとき：(a)と同様である．

(c) p,qがともに変数でかつ$CLASS(p,E) \neq CLASS(q,E)$のとき：そのとき，
$P'=(P-\{p \equiv q\}) \cup \{u \equiv v \mid u \in CLASS(p,E),\ v \in CLASS(q,E)\}$
$E'=MERGE(CLASS(p,E),CLASS(q,E),E)$，
$\theta \in sol(P,E)$とする．$p=q \in P$であるから，$\theta(p)=\theta(q)$

である．$u,v \in CLASS(p,E) \cup CLASS(q,E)$とする．$\theta \in match(u,\theta(p)) \cap match(v,\theta(p))$であるから，補題3.1よりθはu≒vの解である．ゆえにθはP'の解でありかつE'を満たす．ゆえに，$\theta \in Sol(P',E')$．
逆に，$\theta \in sol(P',E')$とする．pとqはともに，$CLASS(p,E) \cup CLASS(q,E)$の元であるから，θはp≡qの解である．E'はEのふたつの元をマージした得られたものであるから，θがE'の解ならばEの解であることも明らかである．したがって，θはPとEの解であることが言えた：$\theta \in sol(P,E)$．

(d) pとqがともに真の部分項のとき：$E'=E$，$P'=P-\{p \equiv q\} \cup PAIRS(p,q)$．$p \equiv q$の解全体は定義により$PAIRS(p,q)$の解全体であるから$sol(P,E)=sol(P',E')$である．

(e) pのみが変数のとき：$E'=(E-\{CLASS(p,E)\}) \cup \{CLASS(p,E) \cup \{q\}\}$，
$P'=P-\{p \equiv q\} \cup \{u \equiv q \mid u \in CLASS(p,E)\}$である．

$\theta \in sol(P,E)$と仮定する．$\theta \in match(q,\theta(p))$であるから，任意の$u,v \in CLASS(p,E) \cup \{q\}$について，$\theta \in match(u,\theta(p)) \cap match(v,\theta(p))$である．よって，補題3.1より，θはu≡vの解である．ゆえにθはE'を満たす．v=qとおけばθはu≡qの解であることがわかる．ゆえに，θはP'の解である．ゆえに，$\theta \in sol(P',E')$である
逆に，$\theta \in sol(P',E')$と仮定する．まず，$CLASS(p,E) \subset CLASS(p,E) \cup \{q\}$であるから，$sol(E') \subset sol(E)$であり，$\theta \in sol(E)$である．$p,q \in CLASS(p,E) \cup \{q\} \in E'$であるから，θはp≡qの解である．ゆえに，θはPの解である．ゆえに$\theta \in sol(P,E)$である．

(f) qのみが変数のとき：(e)と同様である．

(g) ともに定数で同一のとき，すなわち$P'=P-\{p \equiv q\}$，$E'=E$：このとき，$sol(P,E)=sol(P',E')$は明らかである．

(b) p≡qがコンフリクトの場合:P',E'が定義されないので,考慮しなくてよい.
QED

注意. u,vを部分項,ρ,σを割り当てする. ρ(u)とρ(v)がマージ可能であってもu とvが単一化可能であるとはかぎらない. またσ(v)≧σ(u)であっても,uとvが単一化可能とはかぎらない.

(例) u={a/X,b/X}, v={a/{c/1},b/{c/2}}とおくとuとvは単一化できない.しかし ρ(X)={d/3}なる割り当てρについて, ρ(u)とρ(v)はマージ出来る.またσ(X)=φならばσ(v)≧σ(u)である.

〔Fig.11 参照のこと〕

命題4.3. Pを等式系とする. そのとき, 条件①と②は同値である.
① Pが解を持つ.
② Pは単一化可能である.

証明 一般性を失うことなく, P={s≡s'}としてよい. P0=P, E0={{x}|xは,sまたはs'に含まれる変数}とおく.

①-)②. (P0,E0)=)...=)(Pn,En) なる任意の極大列を考える.単一化アルゴリズムの停止性からn は有限である.解集合の不変性より, sol(Pi-1, Ei-1)=sol(Pi, Ei)である. また与えられた条件よりθ∈sol(P0, E0)である. したがってすべてのi(0≦i≦n)について sol(Pi,Ei)≠φである. もしPiがコンフリクトを含むと仮定すればsol(Pi, Ei)=φとなり,矛盾である.よって各Piはコンフリクトを含まない. 列の極大性より,Pn=φでなければならない. これはsとs'が単一化可能であることに他ならない.

②=)①. sとs'が単一化可能であれば, 上記のEnは合同的でありかつコンフリクトをもたない. したがってEnは解を持つ. 単一化は解集合を変えないからsol(P)=sol(En). よって, Pが解を持つ. QED

系 ρ(u)=ρ(v)ならばuとvは単一化可能である.

証明 等式u≡vが解ρを持つから, 命題4.3より, uとvは単一化可能である.
QED

5. プログラムとSLD導出

定義. (プログラム)部分項p,部分項の有限集合Bの順序対(p, B)をプログラム節という. 有限個のプログラム節の集合をプログラムという.

プログラム節(p, φ)のことを単位節とよび,
p.
と書く. ボディが空でないプログラム節(p,{p1,...,pn}) (n≠0)を
p:-p1,..., pn.
と書く.

ゴールGと環境Eの順序対(G,E)を導出状態という.
定義. つぎの条件を満たす, ゴールG={A1,...,An}, G', 環境E,E'の間の関係を(G,E)-)(G',E')と書く.

① n個の, あるプログラム節のコピー, Bi:-B(i,1), ..., B(i,ki) が存在して,
G'={B(1,1),...,B(1,k1),...,B(n,1),...,B(n,kn)}.
② ({A1≡B1,...,An≡Bn},E)-)...=)(φ,E'). (単一化)

使用したプログラム節を明記する必要性のある場合,
Θ={(Ai, Bi:-B(i,1), ..., B(i,ki))|0≦i≦n}とおいて

(G,E)-)(G',E') (Θ)

と書く．

集合{(x) | xはGに出現する変数}をゴールGの初期環境と呼ぶ．

定義．E0をG0の初期環境として，(G,E0)-〉...-〉(φ,E)なる列をGの導出という．

6．モデル

定義．(モデル)Hをプログラムとする．タグ付きの木(PTT)からなる集合 M が次の性質をもつときMを，プログラムのモデルという：

θを任意の割り当て，C(-C1,...,CnをHの任意のプログラム節とする．各i(1≦i≦ n)について0∈match(Ci,ti) なるMの元tiが存在するとき，0∈match(C, t)なる任意のPTT tがMの元である．

定義．(変換Ψ) Sを木からなる集合として，

Ψ(S)={t | あるプログラム節 C(-C1,...,Cn∈H，Sの元t1,...,tnが存在して，θ∈∩{ match(Ci,ti) | 1≦i≦n}，そしてθ∈match(C,t)}．

命題6.1．Ψは単調である．
証明 自明．

命題6.2．モデルの系の共通部分は，ふたたびモデルである．
証明 自明．

命題6.3．モデルの集合は完備(complete)である．すなわち，M1⊂...⊂Mn⊂... ならばM=∪{Mi | i≧1}はモデルである．

証明
t1,...,tn∈M,C(-C1,...,Cn∈H，(∩{ match(Ci,ti) | 1≦i≦n})∩match(C,t)≠φと仮定する．{t1, ..., tn}⊂Mkなる，k≧1が存在する．Mkはモデルであったからt∈Mkである．ゆえに，t∈M．よってMはモデルである．

QED

命題6.4．次の命題は同値である．
① Ψ(S)⊂S．
② Sはモデルである．
証明 自明

Ψはcompleteな領域の上の単調関数であることが分かった．一般的な定理により次の3定理が成り立つ{Lloyd 84}．M0-φ，Mn=Ψ(Mn-i) (n≧1)とおく．

定理6.5．最小モデルMが存在する．
定理6.6．Ψの最小不動点Fが存在する．
定理6.7．Mn↑ω=M=F．

7．SLD導出の整合性と完全性

プログラムHの最小モデルをMとする．つぎの定理が成り立つ．

命題7.1．(整合性) G={A1,...,An} (n≧1)をゴール，EをGの初期環境，(G,E)-〉...-〉(φ,E')，そして，ρ∈sol(E')とする．そのとき，任意の部分項A∈GについてMのある元mが存在し，ρはAからmへの照合割り当てである： ρ∈match(A,m)．

証明

(1) 一回のステップで成功したと仮定する．Hのユニット節のコピーB1,...,Bn 存在して ({A1≡B1,...,An≡Bn},E)=〉...=〉(φ,E')である．ρがE'の解であるから，命題4.1 よりρは各Ai≡Biの解である．したがって，m=ρ(Bi) = ρ(Ai)とおいて，命題

3.2より，$\rho \in match(Bi, m)$である．Mはモデルであるから，mはMの元である．同じく補題3.2より，$\rho \in match(Ai, m)$である．

(2) $p(>1)$回のステップで成功したとする．G={A1,...,An},Eを初期環境として，(G,E)->(G'',E'')->...->(ϕ,E')とする．Bi(-B(i,1), ..., B(i,ki) をプログラム節のコピーとして，G''={B(1,1),...,B(1,k1),...,B(n,1),...,B(n,kn)}，と書ける．帰納法の仮定より，E'を満たす割り当てρについて，Mに含まれるk1+...+kn個の t(1,1),...,t(1,kn),...,t(n,1),...,t(n,kn)が存在して，

$\rho \in match(B(i,j), t(i,j))$ $(1 \leqq i \leqq n,\ 1 \leqq j \leqq ki)$

となる．ρは，Ai≡Biの解でもあるから，$m=\rho(Bi) + \rho(Ai)$とおくと，ふたたび，補題3.2より$\rho \in match(Bi, m)$，Mはモデルであるから $m \in M$である．補題3.2をさらに用いると，$\rho \in match(Ai,m)$である．

QED

次に，SLDの完全性を証明する．プログラムΠとして，$M0=\phi$,$Mn+1=\Psi(Mn)$とする．

補題7.2．$m \geqq 1$,Gをゴール{A1,...,An},Eを環境，$ti \in Mm(1 \leqq i \leqq n)$とする．さらに，各i($1 \leqq i \leqq n$)について，$\theta \in match(Ai,ti)$，そして$\theta \in sol(E)$とする．そのとき，次のようなゴールG'，環境 E'，n個のプログラム節のコピー Bi(-B(i,1), ..., B(i,ki) k1+...+kn 個の Mm-1の元 t(1,1),...,t(1,kn),...,t(n,1),...,t(n,kn)，θの拡張割り当てρが存在する．(m=1 ならばすべてのi($1 \leqq i \leqq n$)について，ki=0となるように選べる．)

① ({A1,...,An},E)->(G',E')．
② G'={B(1,1),...,B(1,k1),...,B(n,1),...,B(n,kn)}，
③ $\rho \in sol(E) \cap sol(E') \cap sol(\{A1 \equiv Bi, ..., An \equiv Bn\})$，
$\rho \in match(Ai,ti)$，
$\rho \in match(Bi,ti)$，
$\rho \in match(B(i,j),t(i,j))$ $(1 \leqq i \leqq n,\ 1 \leqq j \leqq ki)$．
④ ({A1≡Bi,...,An≡Bn},E)=>...=>(ϕ, E')．

証明

$ti \in Mm$であるから，Mnの定義により，あるBi(-B(i,1), ..., B(i,ki)$\in \Pi$，$t(i,1),...,t(i,ki) \in Mm-1$，そして割り当て$\rho i$が存在して，$\rho i \in match(Bi,ti)$かつ$\rho i \in match(B(i,j),t(i,j))$ となる$(1 \leqq i \leqq n,\ 1 \leqq j \leqq ki)$．異なるプログラム節のコピーは変数を共有していないとしてよいから，$\rho = \rho i$ としてよい．また，プログラム節 Bi(-B(i,1), ..., B(i,ki) の中に現れる変数は環境Eのおいてはは$CLASS(x,E)=\{x\}$の中にのみ現れ，かつ$\theta(x)$が定義されないていないとしてよい．このことと，Aiと Biが変数を共有していないことをあわせて，$\rho = \theta \cup \rho 1 \cup ... \cup \rho n$と定義する．$\rho$は，$\theta$の拡張であるから，$\rho \in match(Ai,ti)$かつ$\rho \in match(Bi,ti)$であり，$\rho$はEの解である．これは，$\rho \in sol(\{A1 \equiv Bi, ..., An \equiv Bn\},E)$に他ならない．命題4.2より ({A1≡Bi,...,An≡Bn},E)=>...=>(ϕ,E')である．単一化のステップは解集合を保つから，$\rho \in sol(E')$である．またSLD導出の定義から ({A1,...,An},E)->(G',E')．ここで，G'={B(i,i),...,B(1,k1),...,B(n,1),...,B(n,kn)} である．したがって，①から④までがすべていえた．

QED

命題7.3．$m \geqq 1$,Gをゴール{A1,...,An},Eを環境，$ti \in Mm(1 \leqq i \leqq n)$とする．各i($1 \leqq i \leqq n$)について，$\theta \in match(Ai,ti)$，そして，$\theta \in sol(E)$とする．そのときmステップの導出 (G,E)->...->(ϕ,E')が存在する．
証明 補題7.2をm回，繰返して用いることによる．

QED

命題7.3から，系としてつぎのことがわかる．xを部分項，tをタグ付き木とする．tがプログラムΠの最小モデルMに含まれ，match(A,t)≠ϕならば，({A},E)->...->(ϕ,E')なるSLD導出が存在する．

補題7.4．G={A1, ..., An}，Θ={(A1,C1(-γ1), ..., (An, Cn(- γn)}とおき，

(1) (G,E)->(G',E')　(Θ), かつ
(2) sol(E)⊂sol(F)
とする. そのとき,
(3) (G,F)->(G',F')　(Θ), かつ
(4) sol(E')⊂sol(F'
なるF'が存在する.

証明 P={A≡C | (A,C(-γ)∈Θ}とすると, (1)より(P,E)=>...=>(φ,E')である.ゆえに, sol(P,E)≠φである. (2)よりsol(P,F)⊃sol(P,E)であるから, sol(P,F)≠φ. 命題4.2を用いて, sol(F')=sol(P,F)⊃sol(P,E)=sol(E'). ゆえにsol(E')⊂sol(F').
関係->の定義より
(G,F)->(G',F')　(Θ)
は明らかである.

QED

定義. (応答環境 Answer Environment) Gをゴールとする. 次の条件を満たす環境をGの応答環境という.
① x∈VARS(G)ならばCLASS(x,E)は2つの要素をもつ.
② xがVARS(G)の元でなければCLASS(x,E)={x}.
③ u∈CLASS(x, E), かつu≠xならばVARS(u)∩VARS(G)=φである.

Mを与えられたプログラムの最小モデルとする.
定義. (正しい応答環境 Correct Answer Environment) Gをゴールとして,Gの応答環境Eがつぎの条件を満たすときEを正しい応答環境という:
任意のρ∈sol(E)とA∈Gについて, あるt∈Mが存在して, ρ∈match(A,t)である.

定義. 与えられた導出状態(G,E)の関連変数の集合を次式で定義する.
RELVARS(G,E)=VARS(G)∪(∪{RELVARS(p,E) | ∃x∈VARS(G) p∈CLASS(x,E)}).

(例) RELVARS({{a/x}, {b/y}}, {{x,r}, {y,s}, {u}, {v}})={x,y,r,s}.

命題7.5. (完全性) MをHの最小モデル, Gをゴール, EをGの正しい応答環境, E0をGの初期環境とする. そのとき, つぎの条件を満たす環境E'が存在する.

① (G,E0)->...->(φ,E') なる導出が存在する.
② sol(E)の任意の元θのRELVARS(G,E)への制限θ'はsol(E')に拡張できる.

証明 Q={yは変数 | ∃x∃p∈CLASS(x,E) p≠xかつ y∈VARS(p)}とおく. Eに現れるすべてのx∈Qをそれぞれ, 一斉に, 新しい定数記号に変更する. その結果得られた環境をFで示す. するとFもGの正しい応答環境である. したがって, 補題7.3より, ある環境E''について, つぎのような導出が存在する.

(G,F)->...->(φ,E'')　(1)

cをすぐ上で, Hで新しく導入された任意の定数とする. そのとき, c∈wなる任意のw∈E''について, u∈wかつu≠cならばuは変数である. また, y∈Qならば CLASS(y,E'')={y}であることも明らかである. したがって, この導出を「シミュレーション」することによりつぎのような環境E'''を構成することができる:

(G,E)->...->(φ,E'''). (2)

(2)が(1)のシミュレーションであることから, y∈Qならば, CLASS(y,E''')は変数のみからなることがわかる.

導出(2)に対して, 補題7.4をくりかえして使うことにより, つぎのような環境E'を構成できる:

(G,E0)->...->(φ,E'), (3)
sol(E''')⊂sol(E'). (4)

各y∈Qについて, CLASS(y,E''')は変数のみからなることから, sol(E)の任意の元θのRELVARS(G,E)への制限θ'はsol(E''') のある元に拡張できる. これと(4)をあわせて, θ'はsol(E')に拡張できることがいえた.

QED

8. 単一化ベースの文法形式

PTTの理論を単一化ベースの文法形式に応用する．単一化ベースの文法形式における主要概念は句構造形式，複合素性，そして制約である．まずこれらを説明し，つぎに，具体的に，PATR-II文法形式を 本言語の意味論で解釈する．なお，本節では， 厳密な証明を省く．

複合素性とは，属性と値の対の集合のことである．値は，定数か，あるいは，またそれ自身が複合素性である．複合素性の一般形はつぎのとうりである．

$$[(a1\ v1)\ (a2\ v2)\ \ldots\ (an\ \ vn)]\quad n \geq 0$$

ここで，aiは属性，viは値である．複合素性の意味は，ラベル付きの(サイクルなし)有限有向グラフである．われわれのモデルとあわせるために，サイクルを許すとする．値が置かれる場所に，変数を許すとして，そのような複合素性を， 単に素性項とよぶ．そのとき，句構造規則は，X，X1，...，Xnを素性項として，

X．

あるいは，

X->X1,...,Xn． n≧1

の形である．この形は，われわれのプログラム節 と同じである．(ただし，矢印の向きが逆になっている点に注意．) したがって，素性項の間の単一化の理論が与えられるならば，この文法(句構造形式の集り) の手続き的意味と宣言的意味が決まる．DCGのように制約がついている句構造形式に対する，意味論の拡張も容易である．

つぎに，PATR-II 形式を考える．PATR-IIの意味領域はDAGの集合であり，その意味論はScottの領域理論を枠組みとしている(Shieber 他 86)．しかし，PATR-IIが完全性を持つか否かは，知られていない．(少なくとも，Shieber 他 86 において，言及されていない．)

さて，Fを素性ラベル，Cを定数アトムとするとき，PTT の全体Lはつぎの，領域方程式をみたしていることがわかる。

$$L=[F->L]+C$$

したがって，PTTの領域は，PATR-IIのモデル領域である．以下，この領域の上に，本言語に基づいて，PATR-IIの意味論を定式化する．つづいて，整合性と完全性の証明の概要を述べる．PATR-IIの制約は，素性の値を表す式の間の等式を，論理結合子 and でつないだ論理式である．(ただし，or とnotを含まない版とする．) PATR-II 文法の規則は，たとえば，つぎにように，書かれる．

```
X0->X1 X2
        〈X0 cat〉 = s
        〈X1 cat〉 = np
        〈X2 cat〉 = vp
        〈X0 head〉 = 〈X2 head〉
        〈X0 head subject〉 = 〈X1 head〉
```

ここで，X0,X1,X2は複合素性をあらわす変数である．等式〈X0 cat〉=sは，複合素性X0の素性catの値が定数s と等しいことを示す制約である．すなわち，等式は，割り当てに関する条件として解釈する．したがって，PATR-II文法の宣言的意味論を，本言語のそれと同様に，定義することができる． 一方，等式の手続き的意味は，(制約解決の一種としての)単一化手続きと解釈する．この2つの意味論が一致することは，自明ではない．

一致の証明の概要を述べる．CLP/Tを媒介項として，用いる．すなわち，与えられたPATR-IIプログラム P を CLP/T プログラム[[P]]= [[P]]' ∪{{rel/'=', left/X,right/X}}

に変換する．ここでH'は，P のそれぞれの規則を，たとえば上の例を，つぎのようにして，"パス式"のない，CLP/T のプログラム節に変換して，得られるものとする．

```
{cat/s, head/H, str/{car/L1, cdr/L3}} <-
        {cat/np, head/H1, str/{car/L1, cdr/L2}},
        {cat/vp, head/H,  str/{car/L2, cdr/L3}},
        {rel/ '=', left/H, right/{subject/H1}}.
```

そうすると，PとHは，宣言的および手続き的意味論それぞれについて，一致することが容易にいえる．さらに，Hではふたつの意味論は一致することを用いると，Pの両意味論は一致することが導かれる．ゆえに，PATR-II の整合性と完全性が成り立つ．

謝辞 本研究は，談話理解システムDUALSの記述言語CILの研究開発の一環として横井第2研究室室長の下に行われた．日頃，御討論頂く，横井室長はじめ，DUALSプロジェクトの同僚：杉村，瀧塚，木村，および奥西の各氏に感謝する．部分項の意味論は，Prolog IIの無限木と部分項の関係に関する，古川第1研究室室長からの質問が契機であった．敬友 坂井 公氏は モデルの定義の誤りを指摘してくれた．畏友 横田 一正 氏は，記号上の少なからぬ誤りを指摘してくれた．

参考文献

1) [Ait-Kaci 84] Hassan Ait-Kaci: A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures, A Dissertaion in Computer and Information Science University of Pennsylvania, 1984.

2) [Jaffar 他 86] J. Jaffar and J.L. Lassez: Constraint Logic Programming August, IBM Thomas J. Watson Research Center, 1986.

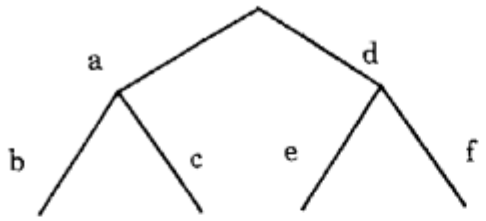
3) [Kasper & Round 86]Robert T. Kasper and William C. Rounds: A Logical Semantics for Features, ACL 86.

4) [Lloyd 84] J.W. Lloyd: Foundations of Logic Programming, Springer-Verlag, 1984.

5) [Mukai 他 85] K. Mukai, H. Yasukawa: Complex Indeterminates in Prolog and its Application to Discourse Models, New Generation Computing, 3, 1985.

6) [Shieber 85] Stuart M. Shieber : An Introduction to Unification-Based Approaches to Grammar, Presented as a Tutorial Session at the 23rd Annual Meeting of the Association for Computational Linguistics, July 8, 1985, Univ. of Chicago, Illinois, USA.

7) [Shieber 他 86] S.M.Shieber, F.C.N. Pereira, L.Karttunen, and M. Kay: A Compilation of Papers on Unification-based Grammar Formalisms Parts I and II, Report No. CSLI-86-48,April,1986.



T={<>,<a>,<a,b>,<a,c>,
<d>,<d,e>,<d,f>}

The trivial tree = {<>}

Fig. 1. Trees

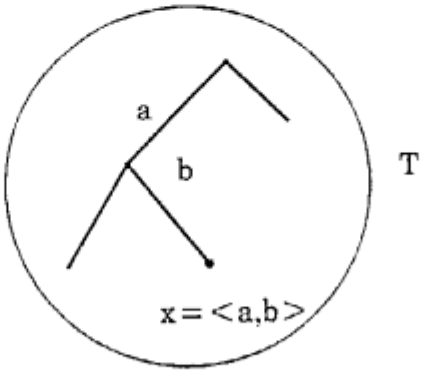


Fig. 2. x is a leaf node of T

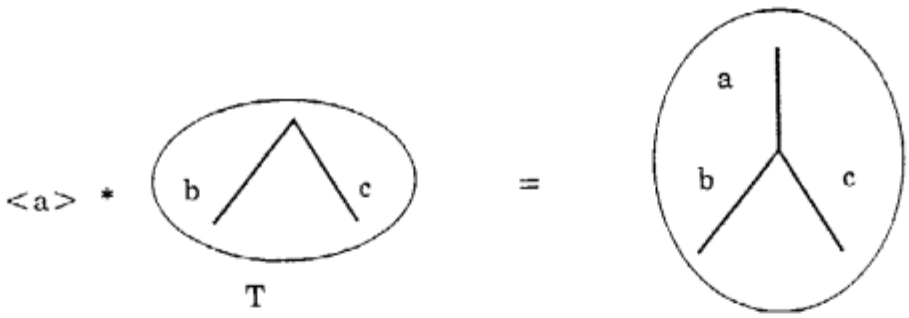


Fig. 3. <a>*T={<>,<a>,<a,b>,<a,c>}

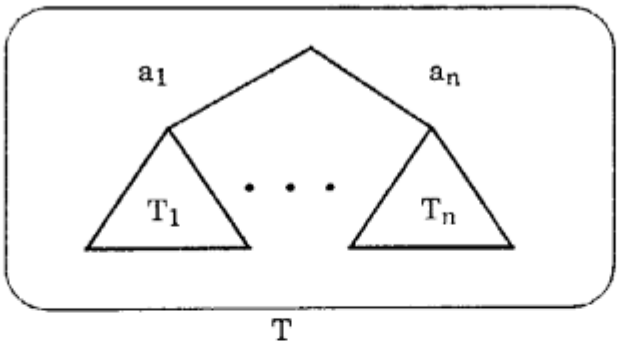


Fig. 4. $T = a_1 * T_1 + \cdots + a_n * T_n$

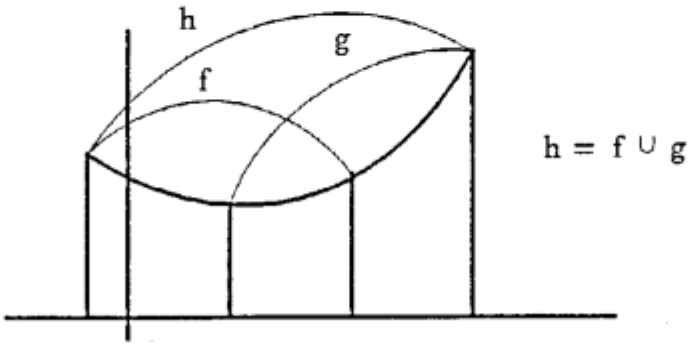
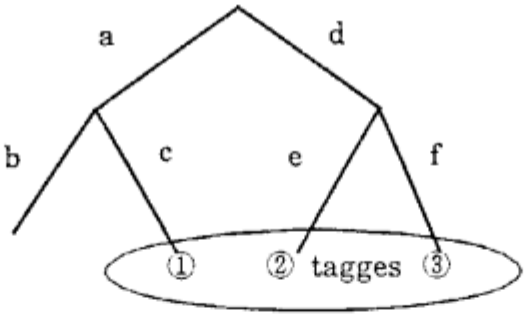


Fig. 5. h is the merge of f and g.



$T = \{ <>, <a>, <a,b>, <a,c>, <d>, <d,e>, <d,f> \}$,
$dom(g) = \{ <a,c>, <d,e>, <d,f> \}$,
$g(<a,c>) = 1$,
$g(<d,e>) = 2$,
$g(<d,f>) = 3$.

Fig. 6. $t = (T, g)$: a partially tagged tree.

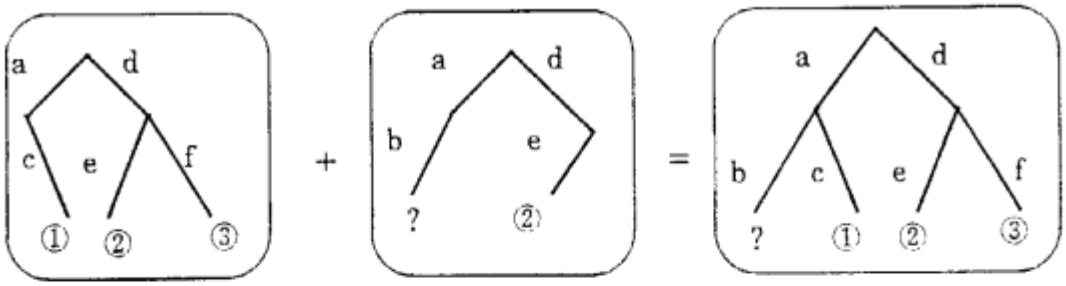


Fig. 7. Merge of trees

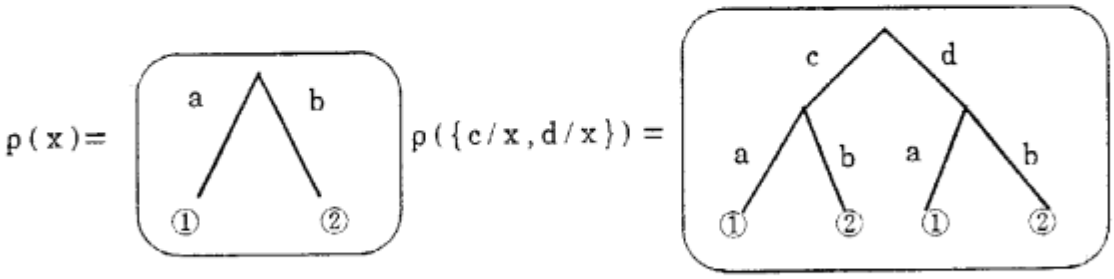


Fig. 8. Assignment

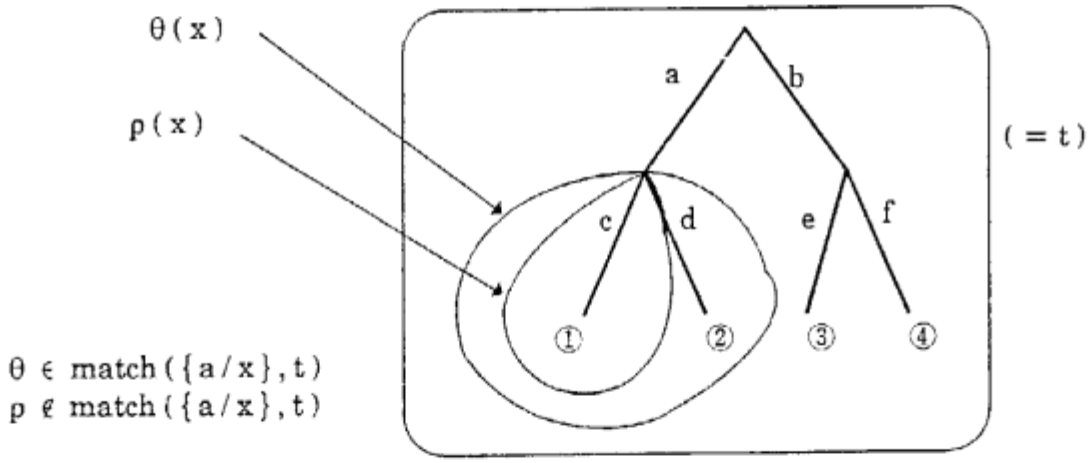
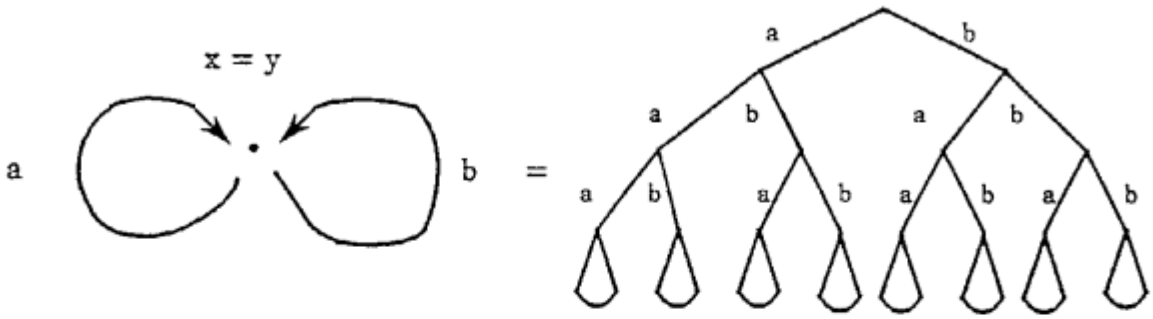


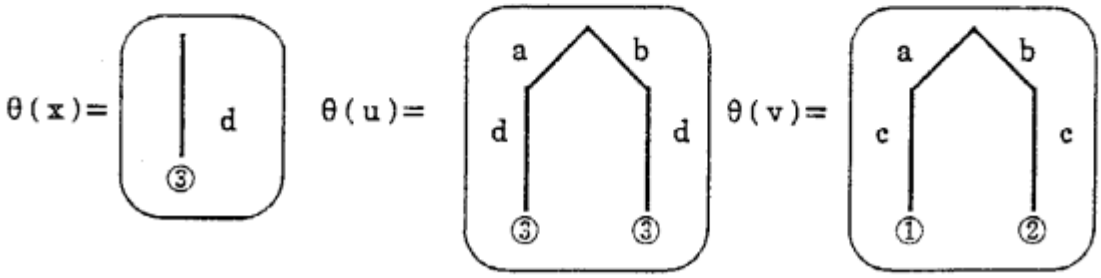
Fig. 9. Matching assignment



$\{ x, y, \{a/x, b/x\}, \{a/y, b/y\}\}$

Fig. 10. An infinite tree

$u = \{a/x, b/x\}$,
$v = \{a/\{c/1\}, b/\{c/2\}\}$.



$\sigma(x) =$ 

$\theta(u)$ and $\theta(v)$ are compatible, but, u and v are not unifiable

$\sigma(u) \leq \sigma(v) \wedge \exists t(\sigma(u) \leq t \wedge \sigma(u) \leq t)$, but,u and v are not unifiable.

Fig. 11. Unifiability and Compatibility

付録3

遅延型論理演算に基づく制約型プログラミング

Constraint Programming based on Lazy Boolean Predicates

[情報処理学会第33回(昭和61年後期)全国大会(33rd Conference of Information Processing Society of Japan, Oct., 1986)にて発表したものを若干修正した.]

概要
freeze制御を基本制御として持つPrologから簡明に制約型プログラミングを作り上げる方法を示す。自然言語理解にあらわれる複合的な制約条件を宣言的に記述するときなどに広く使える。

ABSTRACT

A simple constraint language is designed including equality and basic arithmetic relations. Constraints can be defined in the language by the user. An interpretation is given to the language based on the famous "freeze" control predicate. The basic idea for construction of the interpreter is to define the set of standard logical connectives "and", "or", "not", and so on in lazy evaluation principle by using the freeze.

INTRODUCTION

Complex structure and constraints form a very basic framework for designing complex problem areas like natural language processing. Various instances of this paradigm can be easily seen in recent computational linguistics and semantics, for example.

A brief explanation is given here about how to extend Prolog to have a more flexible power in describing structure and constraint. Our solution to the representation part is partially specified term, which is a kind of extension to the usual term. The other solution to the "constraint" part is "lazy evaluation" based on freeze(Colmeraucr 82). Though our main intention is mixed use of both constraints and structures, we must concentrate here on how to build such a constraint system on top of Prolog.

What is constraint? For our purpose, we give it a restricted characterization as follows: 1) It has a declarative semantics. 2) Positive and negative constraint are treated symmetrically. 3) Interpretation of the constraint is done without deep backtracking.

The proposed constraint sublanguage of an extended Prolog is so simple and natural, and has a so clear declarative semantics that it will be a useful set of constraints.

It should be noted, however, that "freeze-"based system can not derive the contradiction from such like conjunction x=y and x≠y unless both x and y become to be ground. This suggests that the sublanguage would be a middle level set of constraints.

FREEZE(BIND_HOOK)

The goal of the form "freeze(X, p(X))" means that the embedded goal p(X) is suspended while X is unbound, where X is a variable and p(X) is an executable goal. It is easy to see that the predicate can be used as the primitive for both stream and coroutine programming.

CONSTRAINT INTERPRETATION

"Constr" is the main predicate for constraint interpretation. For a constraint expression α, boolean expressions β, γ and δ, the meaning of of the goal of the form of constr(α,β,γ,δ) is explained as follows.

α :a given constraint expression,
β, γ, δ :true,false,undetermined.

The operational meaning, on the otherhand, is explained by listing a portion of top level part of the actual definition in DEC-10 Prolog.

γ =true if α is proved to true.
γ =false if α is proved to be fasle.
γ =undetermined: otherewise.

Both of β and δ are used as control information.

```
?-constr(1=2, true, _, _) => 《fail》
?-constr(X=1, true, _, _)  => X=1
?-constr(X=1, false, _, _) => X is still unbound
?-constr(X=1, _, _, _)     => X is still unbound
?-constr(X\==2, false, U, _). => U = false, X = 2
?-constr(1+X=:=Y,_,_,_), Y=4 => X = 3, Y = 4
?-constr(1+X=\=Y), X=3, Y=4 => 《fail》
```

A PORTION OF THE DEFINITION.

% "=" in the following means the extended unification of CIL system.

```
constr(X,P,Q,R):-bound(R),!.
constr(X,P,Q,R):-unbound(X),!, X=P, X=Q.
constr(true,P,Q,R):-!,P=true,Q=true.
constr(false,P,Q,R):-!,P=true,Q=false.
constr(not(A),P,Q,R):-!,
        not(V,P),
        not(U,Q),
        constr(A,V,U,R).
constr((A,B),P,Q,R):-!,
        and(X,Y,P),
        and(Z,U,Q),
        constr(A,X,Z,R),
        constr(B,Y,U,R).
constr(and(A,B),P,Q,R):-!, % Sequential "and"
        and(X,V,P),
        and(Z,U,Q),
        constr(A,X,Z,R),
        freeze(Z, constr(B,Y,U,R)).
constr(assign(X,Y), P, Q, R):-!,
        assign(X,Y,Q,R),
        P=Q.
constr(X=Y,P,Q,R):-!, equal(X,Y,P,Q,R).
constr(X\==Y,P,Q,R):-!,constr(not(X=Y),P,Q,R).
constr(X=:=Y,P,Q,R):-!,
        and(P1,P2,P),
        and(Q1,Q2,Q),
        exp(X,T,P1,Q1,R),
        exp(Y,T,P2,Q2,R).
constr(X, P, Q, R):- %user defined
        defcon(X, Y),
        constr(Y, P, Q, R).
```

LAZY BOOLEAN PREDICATES

This section is a technical core of the paper. Basic idea is explained with the following simple example: and(X, Y, Z):-Z= 《Y&Z》

```
?-   and(X, Y, Z), Z = true.
 =) X = true, Y = true, Z = true
?- and(X, Y, false). =)X and Y are undetermined.
```

Note that in this case there is an ambiguity in the possible values of (X, Y) from the Boolean logical law. The following are sample definitions of these lazy Boolean predicates.

```
not(X,Y):-
        freeze(X,pv(F,if(X,Y=false,Y=true))),
        freeze(Y,pv(F,if(Y,X=false,X=true))).

and(X,Y,Z):-
        freeze(X,pv(F,if(X,Y=Z,Z=false))),
        freeze(Y,pv(F,if(Y,X=Z,Z=false))),
        freeze(Z,pv(F,if(Z,(X=true,Y=true),
                         andx(X,Y)))).
andx(X,Y):-
        freeze(X,if(X, Y=false)),
        freeze(Y,if(Y, X=false)).

if(true,X=Y,_):-!,X=Y.
if(_,_,X=Y):-X=Y.

pv(F,_):-bound(F),!.
pv(1,X=Y):-X=Y.
```

LAZY EQUALITY

The special case of the form constr(X=Y, Z, U, V) is reduced to equal(X, Y, Z, U, V). The following is its procedural interpretation: 1) Check equality between X and Y in lazy way, The result(true/false) will be returned to U. 2) When "true" is given to Z, X and Y are unified immediately. X and Y should be different from each other if "false" is given to Z. 3) Whenever V is bound, there is no need to continue this checking.

```
?- equal(X,Y,false,A,B), X=1,Y=4.
  =)   X = 1, Y = 4, A = false, B = _91
```

ASSIGNMENT

The predicate called assignment is an internal predicate for restricted uses. Argument passing is a main intention of its use. It lookes like an extension of the assignment in the programming languages FORTRAN, for instance. The unification works both ways as follows: unify(f(X,1), f(2,Y)) =) X=2, Y=1. On the otcr hand, assignment works only a specified way: assign(f(X,1), f(2,Y)) =) Y=1 (X: unbound) Restriction:Each variable is not allowed to occur more than once in the second argument: the form "assign(f(A,g(B)), f(Z,Z))" is not allowed.

SEQUENTIAL CONTROL

We often need some sequential controls for constraint evaluation. For example, let us consider the following definition of the "member".

```
member(X, [X|_]).
```

```
member(X, [_|Y]):-member(X, Y).
```

We can define a constraint version, say it "mem", of the predicate in our language as follows:

```
defcon(mem(X,Y),
       and(assign(Y, [H|T]), (X=H; mem(X, T)))).
```

The body of this definition tells, by using the sequential "and", that "parameter passing" must be prior to the evaluation of the body.

```
?- constr(mem(X,[Y|Z]), true, A, _), Y=1, Z=[U, B], X=2,B=3.  => U = 2, A =
true, X = 2, B = 3, Y = 1, Z = [2,3]

?- constr(mem(X,[Y|Z]), false, A), Y=1, Z=[U, B],
X=2,B=3. => X = 2, B = 3,Y = 1
```

REFERENCE

[Colmerauer 82] A. Colmerauer: Prolog II: Reference Manual and Theoretical Model, Internal Report, Groupe Intelligence Artificielle, Universite d'Aix-Marseille II, 1982.