知識工学と人工知能　45 — 6
（１９８６・３・１０）

文字列領域における
一般化と統一化のアルゴリズム

赤間　清
（北海道大学　文学部　行動科学科）

１．　まえがき

変数の入った表現ｅの集合Ｅと，Ｅ上の写像である代入ｓの集合Ｓからなる枠組みは，基礎的で重要な道具の１つである．Ｅの元で変数の入らない表現全体の集合をＣとする．Ｅの元ｅはパターンと呼ばれ，ＥあるいはＣの部分集合に対応させることができる．ｅに対応するＣの部分集合をＣ（ｅ）とする．ｅにｓ１を適用してＣ（ｅ）の元ｃ１が得られ，ｅにｓ２を適用してＣ（ｅ）の元ｃ２が得られるとする．ｅはｃ１とｃ２の共通点を表わし，ｓ１とｓ２はｃ１とｃ２の差異を表わしているとみなすことができる．

このような枠組みにおいて一般化は，複数のＣ（ｅ）の元からそれらに共通するパターンｅとそれらの差異ｓ１，ｓ２，…を導き出すことと定義される．また統一化は，パターンが複数個与えられたときにそれらを一致させる代入ｓを求めることである．

与えられた情報から仮説的知識体系を作り出し，それを用いて新しい質問に答えるシステムを帰納的学習システム 1) － 4) と呼ぶ．帰納的学習システムにおいては，一般化と統一化のアルゴリズムは最も基本的な要素である．与えられる情報がＣ（Ｅ）の元の列であるとき，その中のいくつかの情報から一般化によってパターンを作る操作は，知識を組み上げていくための基礎となる．また，そのようにして得られた知識を新しい質問に適用して応答を生成する過程で，知識の部分をなすパターンｅ１と質問を表わすパターンｅ２の統一化が必要になる．

このような一般化や統一化のアルゴリズムは，ＥとＳなどの決める表現の枠組みの性質によって，大きく変化する．例えば，ＰＲＯＬＯＧで用いられているようなＳ式の世界を対象としたパターン表現の枠組みにおいては，一般化も統一化も高々１つの解を持つ単純な構造をしている．

本論文で扱うのは，文字列領域に典型的に見られるような表現の枠組みを対象とした一般化と統一化のアルゴリズムである．この場合には解は任意個ありうる．そしてすべての解を求めるのが必ずしも得策ではない場合が存在する．例えば帰納的学習システムへの応用を想定するときには，ある限定された範囲内の解だけに限定することを許して高速に求めることが重要になる．本論文の背景には，帰納的学習システムの理論を構築する準備を与える意図があるので，アルゴリズムを考察するときに，必ずしも全解を求めることにはこだわらず，むしろ今後一般化や統一化の問題設定を拡大して，他の知識のもとでの一般化と統一化の問題を導入するための基礎を与えることを狙いとしている．

２．　文字列領域における一般化の定義

次の２つの文字列のペアを考える．

Ｐ１＝（鳥は飛ぶか，飛びます）
Ｐ２＝（飛行機は飛ぶか，飛びます）

これら２つを見れば，

Ｐ３＝（＄は飛ぶか，飛びます）

がこれら２つの文字列のペアに共通のパターンであり，Ｐ３の＄にそれぞれ，鳥，飛行機という文字列を入れると，それぞれＰ１，Ｐ２ができることを導くことができる．このように文字列のｎ項組が複数個与えられたとき，それらに共通のパターンと，それらの差異を示す代入を得ることを（文字列領域での）一般化と呼ぶ．

問題の設定を明確にし，一般化を達成するアルゴリズムについて考察するのに便利なように，問題や答えをＳ式で表現する方法を次のように指定する．対象である文字列のｎ項組に出現する文字はアトムで，文字列はアトムのリストで，文字列のｎ項組はアトムのリストのリストで表現する．扱う文字は，変数と定数であり，変数にはｎ文字変数と１文字変数を準備する．ｎ文字変数は任意の長さの定数列を代入できる変数であり，１文字変数は１個の定数を代入できる変数である．それらは，アトムの表現において次のように区別される．

n文字変数－－－$から始まるアトム
1文字変数－－－%から始まるアトム
定数　　－－－それ以外のアトム

例えば，

Ｐ１＝（飛行機は飛ぶか，飛びます）
Ｐ３＝（$は飛ぶか，飛びます）

は（漢字やかな文字が１文字でアトムになると仮定して），それぞれ，

Ｑ１＝（（飛　行　機　は　飛　ぶ　か）（飛　び　ま　す））
Ｑ３＝（（$　は　飛　ぶ　か）（飛　び　ま　す））

と表現される．

このような定式化に基づく理論は，対象としていわゆる文字列以外に，例えば単語列も扱い得る．単語列を扱う例を次に挙げる．

Ｑ４＝((WHAT COLOR IS SNOW) (IT IS WHITE))
Ｑ５＝((WHAT COLOR IS APPLE) (IT IS GREEN OR RED))

の２つの単語列のペアについては，共通のパターンを

Ｑ６＝((WHAT COLOR IS $1) (IT IS $2))

とし，$1，$2に入れるべき単語列を，Ｑ４とＱ５でそれぞれ，

$1＝(SNOW)，　　$2＝(WHITE)
$1＝(APPLE)，　　$2＝(GREEN OR RED)

とすればよい．

一般化の解は，一般には，明かに一意ではない．また１つの（定数を含む）一般化パターンがあれば，その定数を変数化して得られるパターンはやはり元の対象の一般化パターンとなる．この関係は順序をなす．この順序に関して極大な一般化パターンをすべて求めるのが一般化の１つの自然な問題設定になる．

３．　一般化のアルゴリズム

文字列のｎ項組が２つ与えられたとき，それらの一般化パターンを（要請に応じて）次々に求めるPROLOGプログラムについて述べる．それは図１の述語GEN-PBで実現される．

（ＧＥＮ－ＰＢ　＊Ｌ１　＊Ｌ２　＊Ｐ　＊Ｂ）

＊Ｌ１と＊Ｌ２は入力の文字列のｎ項組である．出力は２つある．＊Ｐは＊Ｌ１と＊Ｌ２の共通な構造を示すパターン，＊Ｂは＊Ｌ１と＊Ｌ２の差異を示す変数束縛リストである．ただし，トリビアルなパターン（述語CHECK 参照）は認めない．GEN-PBは可能な＊Ｐと＊Ｂをしらみつぶしにすべて求める．

例１：

＊Ｌ１＝（（Ｄ　Ｏ　Ｇ　Ｉ　Ｓ　Ｄ　Ｏ　Ｇ））
＊Ｌ２＝（（Ｈ　Ａ　Ｗ　Ｋ　Ｉ　Ｓ　Ｈ　Ａ　Ｗ　Ｋ））

のとき，得られる＊Ｐと＊Ｂは，それぞれ，

＊Ｐ＝（（$　Ｉ　Ｓ　$））
＊Ｂ＝（（$　（Ｄ　Ｏ　Ｇ）（Ｈ　Ａ　Ｗ　Ｋ））））

である．

例２：

＊Ｌ１＝（（Ａ　Ｂ））
＊Ｌ２＝（（Ｂ　Ａ））

のときには２とうりの解がある．

＊Ｐ＝（（$１　Ａ　$２）），＊Ｂ＝（（$１（）（Ｂ））（$２（Ｂ）（））））
＊Ｐ＝（（$１　Ｂ　$２））），＊Ｂ＝（（$１（Ａ）（））（$２（）（Ａ））））

例３：

＊Ｌ１＝（（Ａ）（Ｂ　Ｃ））
＊Ｌ２＝（（Ｘ　Ｙ）（Ｚ））

のとき，唯一の一般化パターン

＊Ｐ＝（（$１）（$２））

はトリビアルである．従って，＊Ｐ，＊Ｂに当てはまるものはない．

ここで，＊Ｐがトリビアルであるとは，＊Ｐが文字列のｎ項組であることを除けば何の制約もないこと

```
(define gen-pb
  ((*l1 *l2 *p *b) (gpb:b *l1 *l2 () () *prr *b) (check *prr) (rev-rev *prr () *p)))

(define check
  (((*a . *r)) (not (m:var$1-str? *a)) (cut))
  (((*a . *r)) (member *a *r) (cut))
  (((*a . *r)) (check *r)))

(define gpb:b
  ((() () *p *b *p *b) (cut))
  (((*a1 . *l1) (*a2 . *l2) *pl1 *b1 *pl3 *b3)
   (gpb:a () *a1 *a2 () *b1 *pa *b2) (gpb:b *l1 *l2 (*pa . *pl1) *b2 *pl3 *b3)))

(define gpb:a
  ((*z () *e2 *p *b *pp *bb)
   (var-name *z *e2 *p *b *pp *bb))
  ((*z (*a . *r) *e2 *pl *b1 *p3 *b3)
   (append *mae (*a . *ato) *e2)
   (var-name *z *mae *pl *b1 *p2 *b2)
   (gpb:a () *r *ato (*a . *p2) *b2 *p3 *b3))
  ((*z (*a . *r) *e2 *p *b *pp *bb)
   (append *z (*a) *z1)
   (gpb:a *z1 *r *e2 *p *b *pp *bb)))

(define var-name
  ((() () *p *b *p *b) (cut))
  ((*x *y ? ? ? ?) (common *x *y) (cut) (false))
  ((*x *y *p *b (*v . *p) *b) (member (*v *x *y) *b) (cut))
  ((*x *y *p *b (*v . *p) ((*v *x *y) . *b) ) (gensym $ *v)))

(define rev-rev
  ((() * *) (cut))
  (((*a . *r)  *l *) (reverse *a *b) (rev-rev *r (*b . *l) *)))

(define common
  (((*a . *r) *y) (member *a *y) (cut))
  (((*a . *r) *y) (common *r *y)))
```

図１：一般化のプログラムの例（主要部分）．PROLOG/KRに類似のシンタクスを持つ拡張PROLOGであるＰＡＬで書かれている．

である．トリビアルか否かの判定は，トリビアルでないとき成功する述語 CHECKで行なっている．

（ＣＨＥＣＫ　＊Ｐ）

CHECKの各節は次のような処理をする．

(1) ＊Ｐの最初の文字列が変数１個だけの文字列でなければトリビアルではない．

(2) ＊Ｐの最初の文字列が＊Ｐにもういちど出現すればトリビアルではない．

(3) そうでなければ，＊ＰのＣＤＲを CHECKで判定する．

GPB:B はGEN-PBの主なサブ述語である．

（ＧＰＢ：Ｂ　＊Ｌ１　＊Ｌ２　＊Ｐ　＊Ｂ　＊ＰＰ　＊ＢＢ）

＊Ｌ１と＊Ｌ２は文字列のｎ項組である．＊Ｐはこれまでに得られたパターン，＊Ｂはこれまでに得られ

た束縛リストである．ここまでが入力で，あとの２つが出力である．＊ＰＰは＊Ｌ１と＊Ｌ２の共通な構造を＊Ｐに加えたパターン，＊ＢＢは＊Ｌ１と＊Ｌ２の差異を示す変数束縛を＊Ｂに加えた変数束縛リストである．＊Ｐと＊ＰＰは，計算の都合上，逆転した記号列の逆転したリストで表現されている．
例：＊Ｌ１，＊Ｌ２，＊Ｐ，＊Ｂが
　　＊Ｌ１＝（（Ａ）（Ｂ　Ｃ））
　　＊Ｌ２＝（（Ｘ　Ａ）（Ｃ））
　　＊Ｐ　＝（（＄１　Ｗ　＄２））
　　＊Ｂ　＝（（＄２（Ｂ）（））（＄１（Ａ）（Ｂ　Ｄ））））
のとき，
　　＊ＰＰ＝（（Ｃ　＄２）（Ａ　＄３）（＄１　Ｗ　＄２））
　　＊ＢＢ＝（（＄３（）（Ｘ））（＄２（Ｂ）（））（＄１（Ａ）（Ｂ　Ｄ））））
である．
GPB:A はGPB:B の主要なサブ述語である．
　　（ＧＰＢ：Ａ　＊Ｚ　＊Ｅ１　＊Ｅ２　＊Ｐ　＊Ｂ　＊ＰＰ　＊ＢＢ）
＊Ｚは棚あげにされた文字列であり，＊Ｅ１と＊Ｅ２は未処理の文字列である．＊Ｐはこれまでに得られたパターン，＊Ｂはこれまでに得られた束縛リストである．ここまでが入力で，あとの２つが出力である．＊ＰＰは＊Ｚ＋＊Ｅ１と＊Ｅ２の共通な構造を＊Ｐに加えたパターン，＊ＢＢは＊Ｚ＋＊Ｅ１と＊Ｅ２の差異を示す変数束縛を＊Ｂに加えた変数束縛リストである．
(1) ＊Ｅ１＝（）のとき，最終段階である．残りのベクトル（＊Ｚ　＊Ｅ２）を述語VAR-NAMEで判定，処理して，＊ＰＰと＊ＢＢを得る．この場合だけ失敗の可能性がある．
(2) ＊Ｅ１＝（＊Ａ　．　＊Ｒ）で，＊Ａを定数にする場合．述語APPENDで＊Ｅ２を分解して，＊Ａの相手を見つける．残りベクトル（＊Ｚ　＊ＭＡＥ）を述語VAR-NAMEで判定処理して，＊ＰＰと＊ＢＢを得る．
(3) ＊Ｅ１＝（＊Ａ　．　＊Ｒ）で，＊Ａを変数にする場合．＊Ａを棚あげスタックに追加し，＊Ｅ１を＊Ｒに変えた問題に移る．
VAR-NAMEはベクトルのペアが変数になれるか否かを判定し，なれる場合には，すでに出現したベクトルか否かに応じてパターンや束縛の変更を行なう．
　　（ＶＡＲ－ＮＡＭＥ　＊Ｘ　＊Ｙ　＊Ｐ　＊Ｂ　＊ＰＰ　＊ＢＢ）
＊Ｘと＊Ｙは問題のベクトル，＊Ｐはこれまでに得られたパターン，＊Ｂはこれまでに得られた束縛リストである．ここまでが入力で，あとの２つが出力である．＊ＰＰは＊Ｘと＊Ｙの情報を＊Ｐに加えたパターン，＊ＢＢは＊Ｘと＊Ｙの情報を＊Ｂに加えた変数束縛リストである．各節は次のとおり．
(1) ベクトルが（（）（））ならパターンと束縛は変化しない．
(2) ベクトルの両成分が共通の文字を含むなら極大性に反するので失敗である．
(3) すでに変数が対応したベクトルなら，その変数を用いる．パターンだけ変わる．
(4) 新しいベクトルなら，変数を生成してパターンと束縛を更新する．

４．　統一化の定義

２つのパターンが与えられたとき，それらを一致させる代入を求めるのが統一化の問題である．そのような代入は，存在しない場合もあるし，複数個（有限個または無限個）ある場合もある．例えば，パターンが，
　　Ｑ８＝（（％１　は　飛　ぶ　か）（飛　び　ま　せ　ん））
　　Ｑ９＝（（鳥　は　＄１　か）（＄２　ま　せ　ん））
のとき，両者を一致させる代入は
　　％１＝（鳥）
　　＄１＝（飛　ぶ）
　　＄２＝（飛　び）
だけである．無限の解の存在するのは，例えば，
　　Ｐ８＝（Ａ　＄）
　　Ｐ９＝（＄　Ａ）
のときで，この解は，
　＄＝（），＄＝（Ａ），＄＝（Ａ　Ａ），＄＝（Ａ　Ａ　Ａ）…
となる．これは定数とｎ文字変数と１文字変数とを並べる表現だけでは有限の表現にはならない．

```
(define unify:a
    ((nil nil *s *s) (cut))
    (((*a . *r) (*a . *l) *s1 *s2) (cut) (unify:a *r *l *s1 *s2))
    ((*sa *sb *s1 *s3)
     (var-cases *sa *sb *s1 *s2 *sal *sbl)  (eval-1 (m:ap-sub:a '*s2 '*sal) *saln)
     (eval-1 (m:ap-sub:a '*s2 '*sbl) *sbln)  (unify:a *saln *sbln *s2 *s3)))

(define var-cases
    (((*a . *r) (*b . *l) *s ((*a *b) . *s) *r *l)
     (pred-1 (m:var$? '*a)) (pred-1 (m:var$? '*b)))
    (((*a . *r) *e *s1 *s2 *r *vt)
     (pred-1 (m:var$? '*a)) (sub&vt *a *e *s1 *s2 *vt))
    ((*e (*b . *r) *s1 *s2 *vt *r)
     (pred-1 (m:var$? '*b)) (sub&vt *b *e *s1 *s2 *vt))
    (((*a . *r) (*b . *l) *s ((*a *b) . *s) *r *l)
     (pred-1 (m:var%? '*a)) (pred-1 (m:var%? '*b)))
    (((*a . *r) (*b . *l) *s ((*a *b) . *s) *r *l)
     (pred-1 (m:var%? '*a)) (pred-1 (m:const? '*b)))
    (((*a . *r) (*b . *l) *s ((*b *a) . *s) *r *l)
     (pred-1 (m:const? '*a)) (pred-1 (m:var%? '*b))))

(define sub&vt
    ((*x *e *s ((*x . *h) . *s) (*v . *t))
     (append *h (*v . *t) *e) (not (member *x *h))
     (not (pred-1 (m:var$? '*v))) (not (pred-1 (m:var$1-str? '*h))))
    ((*x *e *s ((*x . *hv) (*v *v1 *v2 *%) . *s) (*v2 *% . *t))
     (append *h (*v . *t) *e) (not (member *x *h)) (pred-1 (m:var$? '*v))
     (not (= *x *v)) (not (pred-1 (m:var$1-str? '*h))) (eval-1 (gensym '$) *v1)
     (eval-1 (gensym '$) *v2) (eval-1 (gensym '%) *%) (append *h (*v1) *hv))
    ((*x (*w *v . *t) *s ((*x *w *%1 *$1) (*v *%1 *$1 *$2 *%2) . *s) (*$2 *%2 . *t))
     (pred-1 (m:var$? '*v)) (not (= *x *v)) (pred-1 (m:var$? '*w)) (eval-1 (gensym '%) *%1)
     (eval-1 (gensym '$) *$1) (eval-1 (gensym '$) *$2) (eval-1 (gensym '%) *%2))
    ((*x *e *s ((*x . *h) . *s) *ht)
     (append *h (*x . *t) *e) (not (member *x *h)) (not (m:var$-str? *h)) (append *h *t *ht))
    ((*x *e *s ((*x . *e) . *s) nil)
     (not (member *x *e)) (not (pred-1 (m:var$1-str? '*e)))))
```

図２：統一化のプログラムの例（主要部分）．使用言語はＰＡＬ．

５．　統一化のアルゴリズム

統一化のアルゴリズムの１例を示す．これはPROLOGで問題還元しながら束縛をPUSHしていく方法を採用している．また問題還元のやり方を複数とおり許しており，バックトラックによって複数の解を求めることができる．ただし，必ずしもすべての解が求められるとは限らない．すべての解を求めることを放棄し，対象とする解の範囲をうまく限定することによって計算の時間を抑えることが，本アルゴリズムにおける我々の主要な関心である．

UNIFY:A が統一化の中心となる述語である（図２参照）．

（ＵＮＩＦＹ：Ａ　＊Ｅ１　＊Ｅ２　＊Ｓ１　＊Ｓ２）

＊Ｅ１と＊Ｅ２は，統一化すべき入力パターンである．＊Ｓ１と＊Ｓ２は代入（変数束縛リスト）である．＊Ｓ１に＊Ｅ１と＊Ｅ２を統一化するための束縛を追加して＊Ｓ２が得られる．

以下にUNIFY:A の手続きを少し簡略化した問題還元の概要を示す．ただし，()は空文字列を，小文字のアルファベットは文字を，大文字のアルファベットは文字列を表わすと約束する．また，＋は文字や文字列の連接を，∋は文字が文字列のなかに出現しない関係を表わす．

(1) Ｅ１＝Ｅ２＝()のときは，すでに得られた束縛が答えである．

(2) Ｅ１とＥ２の先頭が等しいときは，束縛は変らず，各々のＣＤＲが新しい問題を構成する．

(3) Ｅ１とＥ２の先頭が共に１文字変数のときは，それらが等しいという束縛を追加して，各々のＣＤＲの問題に移る．

(4) Ｅ１とＥ２の先頭が，定数と１文字変数であるとき，その１文字変数がその定数に等しいという束縛を追加して，各々のＣＤＲの問題に移る．

(5) Ｅ１の先頭がｎ文字変数であるとき，Ｅ１＝ｘ＋Ｒ　とおく．

5-ａ：ｘをＥ２全体に対応させる場合．

条件：Ｅ２∋ｘ

束縛：ｘ→Ｅ２

問題：Ｎ１＝Ｒ，Ｎ２＝()

5-ｂ：ｘをＥ２の部分に対応させる場合．Ｅ２＝Ｈ＋ｖ＋Ｔ　とする．各々の場合に必要な前提条件，追加される束縛，新しい問題を構成する文字列ペアを記す．

(p) ｖがｎ文字変数でない場合

条件：Ｈ∋ｘ

束縛：ｘ→Ｈ

問題：Ｎ１＝Ｒ，Ｎ２＝ｖ＋Ｔ

(q) ｖがｎ文字変数で，ｘ≒ｖの場合

条件：Ｈ∋ｘ

束縛：ｖ→＄１＋＄２＋％，ｘ→Ｈ＋＄１

問題：Ｎ１＝Ｒ，Ｎ２＝＄２＋％＋Ｔ

(r) ｖがｎ文字変数で，ｘ＝ｖの場合

条件：Ｈ∋ｘ，Ｈ＝＄＋％

束縛：ｘ→ΣＨ

問題：Ｎ１＝Ｒ，Ｎ２＝Ｈ＋Ｔ

(6) Ｅ２の先頭がｎ文字変数であるときは，(5)でＥ１とＥ２を読み変えた処理をする．

上で，(1)から(4)までの条件に当てはまる場合は，一意的に新しい問題に展開される．(5)や(6)では，新しい問題は複数とおりありうる．問題還元の場合分けは，文字列Ｅ２の領域分割による．各領域は，ｎ文字変数ｘに対応させる範囲の終点の入るべき領域である．例えば，5-b では，ｘはＨに対応する文字列以上で，Ｈ＋ｖに対応する文字列未満の文字列に対応させる．5-b の(r)はｘ＝ｖの場合を扱っている．このときはｖはｘと独立ではないので(q)のように簡単にはいかない．この困難は次のように考えれば解決する．

Ｅ１＝ｘ＋Ｒ

Ｅ２＝Ｈ＋ｘ＋Ｔ

において，

ｘ＝＄１＋＄２＋％　　(い)

とし，Ｅ１のｘにＥ２のＨ＋＄１を対応させると考える．すなわち，

ｘ＝Ｈ＋＄１　　(ろ)

これらの(い)と(ろ)が共存するためには，条件

＄１＋＄２＋％＝Ｈ＋＄１　　(は)

が成り立つ必要がある．

またそのときＥ１とＥ２は

Ｅ１＝Ｈ＋＄１＋Ｒ　　(ろ)より

Ｅ２＝Ｈ＋＄１＋＄２＋Ｔ　　(い)より

だから，新しい問題は

Ｆ１＝Ｒ

Ｆ２＝Ｈ＋Ｔ

となる．ところで，

＄１＝ΣＨ　　(＄１はＨの０個以上の連接である)

＄２＋％＝Ｈ　　(Ｈは長さが１以上でなければならない)

は，方程式（は）の典型的な解である．この解を採用するとき，（い）と（ろ）は

　　x→ΣH＋H　　（Hの１個以上の連接）

となる．図２のプログラムでは，xがHの１個の連接，つまりHに束縛される場合だけを採用している．

６．　むすび

文字列領域における一般化と統一化のアルゴリズムについて述べた．ここに挙げた一般化のアルゴリズムは，極大でノン・トリビアルな全てのパターンを求めるPROLOGプログラムである．また，統一化のアルゴリズムは，いくつかの（主要な）代入だけを求めるPROLOGプログラムである．これらのプログラムは，高速化についてはあまり考慮していない．実際，我々が帰納的学習システムＬＳ／１で用いているのは，これらの考察とプログラムをもとにしてつくりあげた，より単純で，より高速なLISPプログラムである．本研究の今後の課題は，他の知識を持つ場合にそれらをうまくいかして，よりよい解を速く求める設定の定式化し，それを達成するアルゴリズムを見つけることである．

参考文献

１）赤間清：帰納的学習を行なうシステムの初歩的なモデル，
　HBSR M(S)，NO.7，P29（1984）
２）赤間清：帰納的学習システムＬＳ／０を実現するプログラムの概要，
　HBSR M(S),NO.8,P94（1985）
３）赤間清：帰納的学習システムの最良応答探索，情報処理学会，
　知識工学と人工知能研究会資料，41-12　（1985）
４）赤間清：知識の構造化を重視した学習のモデル，情報処理学会，
　知識工学と人工知能研究会資料，本号（1986）
５）小長谷明彦，梅村護：Shape Upの文字列照合アルゴリズムについて，
　情報処理学会，記号処理研究会資料，24-7（1983）

付録Ａ:一般化の実行例

```
P:(gen-pb ((what color is snow) (it is white))
          ((what color is apple) (it is green or red)) *p *b)
(gen-pb ((what color is snow) (it is white))
        ((what color is apple) (it is green or red))
        ((what color is $00007) (it is $00008))
        (($00008 (white) (green or red)) ($00007 (snow) (apple))))
more:y
failure

P:(gen-pb ((d o g i s d o g)) ((h a w k i s h a w k)) *p *b)
(gen-pb ((d o g i s d o g)) ((h a w k i s h a w k)) (($00009 i s $00009))
        (($00009 (d o g) (h a w k))))
more:y
failure

P:(gen-pb ((a b)) ((b a)) *p *b)
(gen-pb ((a b)) ((b a)) (($00010 a $00011)) (($00011 (b) nil) ($00010 nil (b))))
more:y
(gen-pb ((a b)) ((b a)) (($00012 b $00013)) (($00013 nil (a)) ($00012 (a) nil)))
more:y
failure

P!(gen-pb ((a) (b c)) ((x y) (z)) *p *b)
failure
```

```
P!(gpb:b ((a) (b c)) ((x y) (z)) nil nil *p *b)
(gpb:b ((a) (b c)) ((x y) (z)) nil nil (($00017) ($00016))
       (($00017 (b c) (z)) ($00016 (a) (x y))))
more:y
failure

P!(gpb:b ((a) (b c)) ((x a) (c)) (($1 w $2)) (($2 (b) nil) ($1 (a) (b d))) *p *b)
(gpb:b ((a) (b c)) ((x a) (c)) (($1 w $2)) (($2 (b) nil) ($1 (a) (b d)))
       ((c $2) (a $00018) ($1 w $2)) (($00018 nil (x)) ($2 (b) nil) ($1 (a) (b d))))
more:y
failure
```

付録Ｂ：統一化の実行例

```
P:(unify:b ((%1 ha to bu ka) (to bi ma se n)) ((tori ha $1 ka) ($2 ma se n)) nil *s)
(unify:b ((%1 ha to bu ka) (to bi ma se n)) ((tori ha $1 ka) ($2 ma se n))
        nil (($2 to bi) ($1 to bu) (%1 tori)))
more:y
failure

P!(unify:a (a $) ($ a) nil *s)
(unify:a (a $) ($ a) nil (($)))
more:y
(unify:a (a $) ($ a) nil (($ a)))
more:y
failure

P!(unify:a ($ % $$) (a b c d) nil *s)
(unify:a ($ % $$) (a b c d) nil (($$ b c d) (% a) ($)))
more:y
(unify:a ($ % $$) (a b c d) nil (($$ c d) (% b) ($ a)))
more:y
(unify:a ($ % $$) (a b c d) nil (($$ d) (% c) ($ a b)))
more:y
(unify:a ($ % $$) (a b c d) nil (($$) (% d) ($ a b c)))
more:y
failure

P!(unify:a ($ a $$ b $$$) (a b a p b q) nil *s)
(unify:a ($ a $$ b $$$) (a b a p b q) nil (($$$ a p b q) ($$) ($)))
more:y
(unify:a ($ a $$ b $$$) (a b a p b q) nil (($$$ q) ($$ b a p) ($)))
more:y
(unify:a ($ a $$ b $$$) (a b a p b q) nil (($$$ q) ($$ p) ($ a b)))
more:y
failure
```