知識工学と人工知能　49－2
（1986. 11. 11）

仮説探索システムHYPOSE

赤間　清

（北海道大学　文学部）

Poole 等の仮説生成の枠組みと，それを実現する Theorist システムは，非常に基本的かつ重要である．しかし，それを広く応用したり，他の技術と結合するには，現在の完全な第1階述語論理を基礎とした実現法は，推論速度やシステムの複雑さの点で困難がある．本論文では，Theoristを単純化して，考察の対象をHorn 節の形に制限し，仮説を追加する述語を加えたシステムを導入する．そうすることにより，推論結果の論理的な意味付けを失わずに，完全で速いシステムを作ることができる．これは，このシステムを広くいろいろな問題に適用する可能性を産み出す．このシステムが問題をどのように解くかを，例を用いて示す．

Hypothesis Searcher HYPOSE

Kiyoshi AKAMA

(Faculty of Letters, Hokkaido University, Sappro-shi, 060, Japan)

The framework of theory formation and Theorist system by Poole et. al. is only a very basic tool for acquisition of hypothetical knowledge in inductive learning systems, but they may become an important programming tool if we provide it with inferential efficiency and ability to propose new hypotheses. At the first step toward this, we give a Horn clause version of Theorist, called HYPOSE, which has a built-in predicate that can add new hypotheses into the currrent set of hypotheses. This allows faster implementation preserving the logical meaning for the result of inferences. We illustrate the character of HYPOSE by solving a puzzle in prolog and HYPOSE and comparing them with each other.

1．　はじめに

Poole 等は，次のような仮説生成の枠組み[6]を提案した．扱う知識は，可能な仮説の集合H（possible hypotheses），確実で無矛盾な知識の集合F（facts），観測事象G（observations）からなる．仮説生成とは，仮説の集合Hのある部分集合Dを選び出して，F∪Dが無矛盾という条件のもとで，GをF∪Dの論理的帰結とすることである．彼らは，この枠組みを実現するシステムとして，節形式による完全な第1階述語論理を基礎とした Theorest というシステムを作成した．それは，H＝φのとき Prolog に帰着するという意味で，Prolog の1つの拡張とも見ることができる．

Poole 等の定式化した仮説生成の枠組みは，「仮説生成」機能が本来持つ意味の大きさからいえば，あまりにも基本的で，仮説生成を研究していくための基礎的な道具の1つに過ぎないとみなすべきである．Poole は，複数の理論の比較[7]などを試みているが，それを含めて困難な問題が山積しており，仮説生成の研究はむしろこれからなのである．帰納的学習システムの研究[2]では，人間（知能）とは，外界から受け取る情報を基にして，仮説的知識体系を構成し，外界により良く応答する存在と規定されている．そのような場合の仮説的知識体系の構築問題への適用は，Poole 流の定式化では，まだまだ遠く及ばない．

しかしながら，Theorist の枠組みは非常に基本的で重要である．仮説生成という名称に惑わされずに考えると，応用範囲も広いと我々は予想している．Theorist の枠組みを道具として使う時に何ができるか．それが，我々の最も関心がある点である．もちろんこれには，帰納的学習システムの研究に応用する方法を発見して，両者の研究を発展させたいという希望も含まれる．

改めて道具として見る時，Theorist は最善ではない．完全な第1階述語論理を基礎としているために，完全なシステムを実現するのが困難である．少なくとも高速なシステム（仮説生成などでは，探索量の爆発は最も重大な問題の1つであるから，高速性は非常に重要である）を望むことができない．また，応用を広げるために，最良優先探索[3]や，継承階層機構[1]などの技術と連結するのも困難が多い．さらに，帰納的学習システムなどに応用するには，新しい仮説を提案していく力に乏しい．

これらの認識の下に，本論文では，Theorist を単純化して，考察の対象を Horn 節の形に制限したシステムを考え，仮説を追加する述語を導入する．そうすることによって，Theoristと同様に論理的な意味付けを失わずに，完全で速いシステムが作れる可能性がでてくる．このシステムが基礎的な道具として，どう実現され，どんな役割を果たし得るか．帰納的学習システムとのギャップを埋めるためには何を追加すべきか．これを出発点として考察していきたい．

2．　仮説生成の枠組み

Poole 等は，仮説生成の最も基礎的な枠組みを次のように定式化した．扱われる知識は，可能な仮説の集合H（possible hypotheses），確実な知識の集合F（facts），観測事象G（observations）からなる．ここでの推論は，仮説の集合Hのある部分集合Dを選び出して，F∪Dが無矛盾という条件のもとで，GをF∪Dの論理的帰結とすることである．もちろんFはそれ自身無矛盾と仮定されている．この枠組みは Theorest というシステムとして実現されている．

Theorist の知識表現には，完全な第1階述語論理が使われている．Theorist システムの受け取るのは，つぎのような一般の節形式の式である．

$$L_1 \vee L_2 \vee \cdots \vee L_n \ <- \ L_{n+1} \wedge L_{n+2} \wedge \cdots \wedge L_{n+m}$$

ここで，L_iは任意のリテラル（literal）であり，否定も表現できる．従って，Theorist における矛盾とは，あるアトム（原子論理式）とその否定がともに導かれることである．

Theorist のシステムを作るには，第1階述語論理全体を扱う節形式の定理証明機が必要である．彼らはそれを Prolog で実現している．その定理証明機の基本となるアイディアは，(1) 一般の節形式の式のすべての contrapositive form を知識に追加し，Prolog と同じくゴール駆動，後ろ向き推論をすることと，(2) 後ろ向き推論で，得た新しいゴールの否定がその祖先のゴールとユニファイ可能なときは，背理法によって祖先のゴールが証明されたとすることである．

Theorist の基本的なアルゴリズムは，与えられた 観測事象の証明を得るための後ろ向き推論を進めるときに利用できる（head がマッチする）知識を，ＦまたはＨからみつけて蓄えて行くことである．この場合，Ｆの知識は無条件で利用可能な知識であり，Ｈの知識はＦ∪Ｄが無矛盾という条件のもとで利用可能な知識である．一般に，ある式の集合が矛盾すれば，その集合に幾つかの式を追加した集合もやはり矛盾する．したがって，Ｈの元ｈを追加していくとき，集合Ｆ∪Ｄは，

ａ：最後まで矛盾しない．

ｂ：ある時点以降矛盾する．

のいずれかになる．Theorist は，仮説ｈをＤに新しく採用するたびに矛盾のチェックを行ない，矛盾を発見した時点で，backtrack する．したがってTheorist は，制約付きの解探索をおこなう Prolog であるといえる．ただしその制約は，有限個の式の無矛盾性の条件だけである．

３．　仮説探索システムＨＹＰＯＳＥ

Theorist のシステムは，第１階述語論理全体を扱うため，複雑化し，高速実現も困難である．本論文では，Theorist を単純化して，考察の対象を Horn 節の形に制限し，仮説を追加する述語を利用できるシステムーＨＹＰＯＳＥ（ HYPOthesis SEarcher)と仮称する ーを導入する．その意図は，Theoristに含まれる「制約付き問題を扱う Prolog」としての機能を，論理的整合性を保ったまま，高速に実行するシステムを実現して，Poole等の示した暗黙推論，故障診断などだけでなく，広く一般のプログラミングの基礎的な道具とする可能性を探ることである．

ＨＹＰＯＳＥでは，仮説である規則 : head <− body は，

```
(hypo head . body)
```

と書かれる．また，事実である規則： head <− body は，

```
(fact head . body)
```

と書かれる．事実の集合Ｆと選択された仮説の集合Ｔに対する制約条件として，contr という述語が使われる．この述語は（少なくとも）２種の意味で，用いられる．１つは，統合性制約条件（integrity constraint）として．定義するには述語 contr を用いて，

```
(fact (contr) . body)
```

と書き， body が成立すれば矛盾であることを宣言する．この場合， contrは，矛盾（contradiction）の略と見ればよい．もう１つの用い方は，仮説の追加などを行なうことである．例えば，

```
(fact (contr) (parent *p *c) (add (child *c *p)))
```

は，*pが*cの親であることがわかったときには，*cが*pの子供であるという仮説もＤに追加することを示す．この場合， contrは，制御（control）の略と見ればよい．

ＨＹＰＯＳＥの推論は，Theoristにおける矛盾の条件を（contr）が証明可能であるという条件に置き換えたものである．ただし，(contr) の証明可能性を調べる時，述語add が出現すると，add で示された仮説を追加採用しなければならない．add の出現する節（add 節）は最後には失敗すると約束されている．したがって，add 節は（contr）の証明可能性には直接影響しないが，追加した仮説が（contr）の子孫ゴールをさらに展開する可能性があるので，間接的に影響する．

簡単な暗黙推論の例を示す．鳥は飛ぶ．emu は鳥だ．Tweety は鳥で，Edna は emu だ．emu は飛ばない．このような知識は，次のように書かれる．

```
(hypo (fly *) (bird *))
(fact (bird *) (emu *))
(fact (bird Tweety))
(fact (emu Edna))
(fact (not-fly *) (emu *))
(fact (contr) (fly *) (not-fly *))
```

矛盾を起こし得る述語は，fly と not-fly の組合わせだけであり，それらが唯一の制約条件になっている．このような知識のもとで，Tweety が飛ぶかをシステムに聞いて見る．推論の過程を書

かせると次のようになる．

```
PI(solve (fly Tweety))
(--- select-h new u00011 (fly Tweety) (bird Tweety))
(select u00011 (fly Tweety) ((bird Tweety)))
(----------> check in (u00011))
(--- select-f (contr) (fly *) (not-fly *))
(--- select-h old u00011 (fly Tweety) (bird Tweety))
(--- select-f (bird Tweety) (emu Tweety))
(--- select-f (bird Tweety))
(--- select-f (not-fly Tweety) (emu Tweety))
(---------> consistent: (u00011))
(--- select-f (bird Tweety) (emu Tweety))
(--- select-f (bird Tweety))
-------
(fly Tweety)
(u00011 (fly Tweety) (bird Tweety))
```

すなわち，(fly Tweety) は肯定される．このときに採用された唯一の仮説は，Tweety は鳥だから飛ぶという意味の，

(u00011 (fly Tweety) (bird Tweety))

であり，これは矛盾を引き起こさない．次に，Edna が飛ぶかを聞くと，

```
PI(solve (fly Edna))
(--- select-h new u00012 (fly Edna) (bird Edna))
(select u00012 (fly Edna) ((bird Edna)))
(----------> check in (u00012))
(--- select-f (contr) (fly *) (not-fly *))
(--- select-h old u00012 (fly Edna) (bird Edna))
(--- select-f (bird Edna) (emu Edna))
(--- select-f (emu Edna))
(--- select-f (not-fly Edna) (emu Edna))
(--- select-f (emu Edna))
(------- contradiction found in (u00012))
failure
```

となり，答えは NO である．飛ぶことを結論するためには，Edna が鳥だから飛ぶという仮説：

(u00012 (fly Edna) (bird Edna))

を採用せざるを得ないが，（たとえどんな仮説によってであれ）命題 (fly Edna) が成立したとたん，それは (not-fly Edna) のために矛盾に導かれてしまう．

4． 例題：LOVE AFFAIR

組み込み述語 add が効果を発揮する問題の例として，次の問題：LOVE AFFAIRを考えてみる．

ゆうこ，みゆき，ともこ の３人の女性と，おさむ，やすお，かずお の３人の男性が無人島に流れついた．自然の摂理から，しばらくすると各男性は女性を，各女性は男性を，各々１人だけ愛するようになった．ただし，

２人から愛された人はおらず，
みゆきが愛した人は，ゆうこを愛し，

やすおが愛した人は，かずおを愛し，
ともこが愛した人が愛した人は，おさむを愛した．
やすおはともこを愛さなかった．

では，おさむが愛した人は，誰だったのか？

この問題は，渡辺の論文[5]のものに少し変更を加えたものである．Prolog でこの問題を解くためのプログラムの例を図１に示す．これは，典型的な generate and test のプログラムである．６人の愛する人について可能性のある3^6とうりの場合について，const1 から check3 までの６とおりの条件がチェックされる．実は const1 と const2をまとめて考えると，permutation を用いた generater が使えて，条件チェックはもっと少なくて済むが，それはまた別問題なので，ここではそれは考えない．この解を図２に示す．

図３にＨＹＰＯＳＥによるプログラムの例を挙げる．図１の６つの条件チェックは述語 contr の定義に対応している．述語 dif は，等しくないことを表わす組み込み述語である．実行するには，

```
(love-affair *1 *2 *3 *4 *5 *6)
```

と問い合せを出せばよい．図３のプログラムの宣言的な意味は，粗く言えば，

```
(and (love-affair *1 *2 *3 *4 *5 *6) (not (contr)))        ☆
```

に近い．ただし，(love-affair *1 *2 *3 *4 *5 *6) と (contr) は，Ｆ∪Ｄによって結ばれている．(love-affair *1 *2 *3 *4 *5 *6) の実行は仮説ｈをＤに供給し，述語contr はＦ∪Ｄという知識によって実行される．☆を逐次型で考えれば，図１で示したgenerate and test のプログラムに近くなり，

```
(love-affair *1 *2 *3 *4 *5 *6)
```

の６つの変数を決定してから，それらを contr でテストすることになる．しかし，この場合３つの高速化が可能である．第１に，６つの仮説（変数）を決定してからではなく，そのうちの１つを決定するごとに部分的なテストができる．部分的なテストで contr が成功すれば backtrack する．第２に，contr の各節は並列に展開できる．その場合 contr のテストとは，知識Ｆ∪Ｄのもとで contr のすべての節を展開できるだけ展開し，その１つも成功に至らない（add 節が成功しないことはすでに述べた）ことを確認することである．第３に，前回の部分的なテストの結果の状態（全ての可能な展開を終えた状態）は次の（部分的な）テストの出発点として利用できる．

明らかに図１の Prolog プログラムより図３のＨＹＰＯＳＥのプログラムのほうが上記の３点の分だけ効率がよい．

５．　考察

図３の制約条件の記述を，組み込み述語 add を使わないで書いたらどうなるだろうか．例えば，

```
(fact (contr) (love miyuki *) (add (love * yuuko)))
```

を，

```
(fact (contr) (love miyuki *) (not (love * yuuko)))
```

とする．このように変換すれば，組み込み述語 add は必要なく，すべてが統合性制約条件の枠内に落ちる[9]．後者の条件は，みゆきが愛する人がゆうこを愛していないと矛盾を生ずるので，一見妥当なように見えるが実は問題がある．not の判定が，逐次的に増大してゆく集合Ｆ∪Ｄに対して適用されるからである．この条件は，例えば，

```
Ｆ∪Ｄ＝ { (love miyuki kazuo) }
```

に対して矛盾を宣告する．しかし後で新しい仮説 (love kazuo yuuko) が追加されて，

```
Ｆ∪Ｄ＝ { (love miyuki kazuo), (love kazuo yuuko) }
```

となり，矛盾は解消される可能性があるので，それは早計である．論理的な整合性を保つためには add が必要である．（ＨＹＰＯＳＥは両者を扱う．）

述語 add があるために，(contr) が成り立つか否かのチェックは，通常の prolog の実行とは違って，実行途中で導出に使える知識（公理）が増加する．従って，処理系に新しい工夫が必要である．

```
(define test
  ((*x *y *z *a *b *c)
   (member  *x (osamu yasuo kazuo))
   (member  *y (osamu yasuo kazuo))
   (member  *z (osamu yasuo kazuo))
   (member  *a (yuuko miyuki tomoko))
   (member  *b (yuuko miyuki tomoko))
   (member  *c (yuuko miyuki tomoko))
   (= *hypo ((yuuko *x) (miyuki *y) (tomoko *z)
             (osamu *a) (yasuo *b) (kazuo *c)))
   (not (const1 *hypo))
   (not (const2 *hypo))
   (not (const3 *hypo))
   (check1 *hypo)
   (check2 *hypo)
   (check3 *hypo)
   (print (--------> *hypo))
   (print (osamu loves *a))
   (terpri)
   (false)))


(define const1  ((*h) (member (* *1) *h) (member (* *2) *h) (not (= *1 *2))))
(define const2  ((*h) (member (*1 *) *h) (member (*2 *) *h) (not (= *1 *2))))
(define const3  ((*h) (member (yasuo tomoko) *h)))

(define check1  ((*h) (member (miyuki *) *h) (member (* yuuko) *h)))
(define check2  ((*h) (member (yasuo *) *h) (member (* kazuo) *h)))
(define check3  ((*h) (member (tomoko *1) *h)
                      (member (*1 *2) *h)
                      (member (*2 osamu) *h)))
```

図1　LOVE AFFAIR の問題を解くための Prolog プログラムの例

```
P!(test . *)
(--------> ((yuuko yasuo) (miyuki kazuo) (tomoko osamu)
            (osamu tomoko) (yasuo miyuki) (kazuo yuuko)))
(osamu loves tomoko)

(--------> ((yuuko kazuo) (miyuki yasuo) (tomoko osamu)
            (osamu tomoko) (yasuo yuuko) (kazuo miyuki)))
(osamu loves tomoko)

failure
```

図2　Prolog プログラムの実行によって得た LOVE AFFAIR の問題の解

```
(hypo (love yuuko osamu))
(hypo (love yuuko yasuo))
(hypo (love yuuko kazuo))
(hypo (love miyuki osamu))
(hypo (love miyuki yasuo))
(hypo (love miyuki kazuo))
(hypo (love tomoko osamu))
(hypo (love tomoko yasuo))
(hypo (love tomoko kazuo))
(hypo (love osamu yuuko))
(hypo (love osamu miyuki))
(hypo (love osamu tomoko))
(hypo (love yasuo yuuko))
(hypo (love yasuo miyuki))
(hypo (love yasuo tomoko))
(hypo (love kazuo yuuko))
(hypo (love kazuo miyuki))
(hypo (love kazuo tomoko))
(fact (love-affair *1 *2 *3 *4 *5 *6)
      (love yuuko  *1)  (love miyuki *2)  (love tomoko *3)
      (love osamu  *4)  (love yasuo  *5)  (love kazuo  *6))
(fact (contr) (love * *1) (love * *2) (exec (dif *1 *2)))
(fact (contr) (love *1 *) (love *2 *) (exec (dif *1 *2)))
(fact (contr) (love yasuo tomoko))
(fact (contr) (love miyuki *) (add (love * yuuko)))
(fact (contr) (love yasuo *) (add (love * kazuo)))
(fact (contr) (love tomoko *) (love * **) (add (love ** osamu)))
```

図3　ＨＹＰＯＳＥによる LOVE AFFAIR のプログラムの例

```
(love-affair yasuo kazuo osamu tomoko miyuki yuuko)
(h00005 (love yuuko yasuo))
(h00009 (love miyuki kazuo))
(h00019 (love kazuo yuuko))
(h00010 (love tomoko osamu))
(h00015 (love osamu tomoko))
(h00017 (love yasuo miyuki))

(love-affair kazuo yasuo osamu tomoko yuuko miyuki)
(h00006 (love yuuko kazuo))
(h00008 (love miyuki yasuo))
(h00016 (love yasuo yuuko))
(h00010 (love tomoko osamu))
(h00015 (love osamu tomoko))
(h00020 (love kazuo miyuki))
```

図4　ＨＹＰＯＳＥによる LOVE AFFAIR の解

ＨＹＰＯＳＥのシステムは，freeze 機構のある prolog と似ている．freezer は変数を介して本体の実行と連絡し，変数束縛が制御の移動をもたらす．ＨＹＰＯＳＥでは，contr はＦ∪Ｄを介して本体の実行と連絡しており，新たな仮説の追加が contr のチェックを起動する．

６．　むすび

推論結果の論理的意味を保証しつつ，Poole らのシステムより高速に動作する，仮説生成の基礎システムＨＹＰＯＳＥを与えた．このシステムを基礎として，いろいろな応用可能性を探せば，かなりおもしろい利用方法があると予想している．故障診断[8]や，暗黙推論[8]だけでなく，物語理解などの枠組みともつなげる[3]ことができる．ただし，それらを進めて行く過程で，少なくとも，仮説を操作するより多くの組み込み述語が必要となろう．また，継承階層機構や，最良優先探索との結合は，応用範囲を拡大するために重要であるが，それらについての検討も今後進める必要がある．

本論文では，制約や，仮説などを扱うのはトップレベルだけとしたが，

(solve_with_constraint　query　constraint)

の形の組み込み述語を導入して任意の所で制約を扱えるほうが便利かもしれない．その種の検討も今後の課題の１つである．

文献

[1] 赤間　清：継承階層 Prolog と多重継承，日本ソフトウエア科学会第３回大会論文集，B-5-2 (1986)

[2] 赤間　清：翻訳知識の帰納的学習，日本ソフトウエア科学会第３回大会論文集，B-6-2 (1986)

[3] 赤間　清：継承階層 Prolog のメタプログラミングによる最良優先探索システム，情報処理学会，知識工学と人工知能研究会資料，49-1，P1-8 (1986)

[4] 弦巻宏治，国藤進，古川康一：メタプログラミングによる仮説生成システムの試作について，日本ソフトウエア科学会第２回大会論文集，3A-4,P73-76 (1985)

[5] 渡辺　治：Prolog における functor の役割について，ソフトウェア基礎論研究会資料，7-1 P1-10 (1984)

[6] David Poole, Romas Aleliunas, Randy Goebel : Theorist : a logical reasoning system for defaults and diagnosis, Knowledge Representation, N.J.Cercone and G.McCalla (eds.), Springer-Verlag, New York

[7] David L. Poole : On the Comparison of Theries: Preferring the Most Specific Explanation, Proc. IJCAI-85, p.144-147 (1985)

[8] David L. Poole : Default Reasoning and Diagnosis as Theory Formation, Technical Report CS-86-08 p.18 (1986)

[9] Randy Goebel,Koichi Furukawa and David Poole : Using definite clauses and integrity constraints as the basis for a theory formation approach to diagnostic reasoning, Submitted to the International Logic Programming Conference 1986, London, England.