知識工学と人工知能　59－12
（１９８８．７．７）

知識編集に基づく学習

荒屋真二
（福岡工業大学　工学部　通信工学科）

Ａｎｄｅｒｓｏｎは，ＡＣＴ* 理論において知識編集を一般的な学習機構としてとらえている．その知識編集は，合成と手続化という二つの操作よりなる．合成は，特定の問題解決に使用された複数のルールを編集し，同一効果を持つ単一ルールを生成する．手続化は，過去に長期記憶から検索しなければならなかった事実をルールの中に組み入れる．この編集機構は，汎化や分化といった，これまで帰納的学習と考えられていたものと同一の効果を生み出す．本論文では，これら二つの編集機構に加え，“構造化”という知識編集機構を提案する．構造化は，ルールの相互関係に関する知識（メタ知識）を抽出し，それを用いて断片的ルールをネットワーク状に結合する．メタ知識としては，ルール相互の類似性，ルール相互の支持関係の二つが利用される．本論文では，合成によって引き起こされる諸問題を明らかにすると共に，構造化がそれらの問題点の大部分を回避できることを示す．また，知識の追加／削除に伴って，構造化を効率的に行う逐次編集アルゴリズムを提案する．これは知識獲得に不可欠な基本的情報処理と考えられる．

Learning based on Knowledge Compilation

Shinji ARAYA
Fukuoka Institute of Technology
3-30-1, Wajirohigashi, Higashi-ku, Fukuoka, 811-02, Japan

In ACT* theory, Anderson insists that knowledge compilation is a general learning mechanism. It includes two operations called “composition” and “proceduralization”. Composition compiles multiple rules used to solve a paticular problem and produces a single rule which has the same effect. Proceduralization puts facts which had to be retrieved from the long term memory into rules. This compilation mechanism gives the same effect as generalization and discrimination occured by inductive learning. This paper, in addition to above two types of compilation, proposes a new compilation mechanism called “structuring”. It extracts knowledge about relations among rules (a kind of meta-knowledge), and connects fragmentary rules into a network form. The meta-knowledge includes similarities among rules and supporting relations. This paper clarifies several problems occured by composition, and demonstrates how the structuring can avoid most of those problems. An incremental compilation algorithm for efficient structuring is also proposed, which is driven by addition or deletion of rules. It may be considered as a fundamental information processing which is indispensable to knowledge acquisition.

1．まえがき

Ａｎｄｅｒｓｏｎ等は，ＡＣＴ* 理論において知識編集を一種の学習機構としてとらえた[1]．その知識編集は，合成と手続化の二つよりなる．合成とは，複数のルールから，それ等と同一の効果を持つ単一のルールを生成する過程であり，最初，Ｌｅｗｉｓによって提案された[2]．手続化は，過去に長期記憶から検索しなければならなかった情報をルールの形で生成する過程である．当初，Ａｎｄｅｒｓｏｎ等は，これらの知識編集は既存の処理を単に効率化するものであり，新しい行動を獲得するためには，汎化や分化といった帰納的学習機構が必要であると考えていた．しかし，その後，この知識編集は帰納的学習機構の効果と考えられていたものと同一の効果を再生可能であるということを明らかにし[3]，一般的学習機構として位置付けるようになった．即ち，帰納プロセスは新しい事態に対処するための基礎知識を見つけ出すという意識的な問題解決プロセスであり，そこで用いられた知識を編集すると汎化や分化が生じるというわけである．

特定の課題を繰り返し解くにつれて人間の問題解決速度は指数関数的に向上する(power law of practice)という実験データをうまく説明するモデルとして Chunking 機構がある[4]．これは低レベルの記述を高レベルの記述にまとめ上げるという点で，Ａｎｄｅｒｓｏｎ等のいう知識編集と本質的に類似している．この Chunking 機構も当初は単純な訓練の学習モデルであったが，汎化や分化の効果をも実現できる一般的学習機構に発展しつつある[5]． Chunking 機構は，結果の記憶により同一問題に対する再計算を省略するという意味で，テーブルルックアップ[6],[7]，マクロオペレータ[8]などの学習とも類似している．

一方，プロダクションシステムの高速化という工学的目的のもとに行われてきた研究の流れがある[9],[10],[11]． この研究の基本的考え方は，ルール相互の関係に関する知識（メタ知識）を用いてルールを構造化することによって，システムの性能を向上させるという点にある．メタ知識としては，ルール相互の類似性[9],[10],[11]，ルール相互の排他性[10]，ルール相互の支持関係[11]などが利用され，断片的なルールがネットワーク状に構造化される．これまでは，この“構造化”という知識編集は学習機構として捉えられていなかった．

本論文では，筆者等が提案している連想ＲＥＴＥネット方式[11]が， ルール合成よりも単純かつ強力な知識編集による学習機構であることを示す．連想ＲＥＴＥネットはルール相互の支持関係を用いてＲＥＴＥネット[9]を更に構造化したもので，ＲＥＴＥネットをサブネットとして完全に包含する．まず，ルール合成には，ルール数の増大，整合性維持の複雑化，遮蔽現象の発生，合成ルールの妥当性，低い学習効率などの問題点があることを明らかにする．合成の安全性（safty）は論じられているが[12]，安全性を保証する具体的アルゴリズムは提案されていない．また，これら合成にまつわる諸問題が連想ＲＥＴＥネット方式によって回避できることを示す．

まず，２章において知識編集による学習の基本的考え方を述べる．３章では，ルール合成の概要を述べたあと，その諸問題を考察する．４章では，連想ＲＥＴＥネットにおけるルールの構造化について述べ，合成と構造化の両者を比較考察する．また，構造化処理を効率化するために逐次アルゴリズムを提案する．５章では本論文をまとめるとともに今後の課題についてふれる．

なお，本稿ではプロダクションシステムとしてＯＰＳ５[13]を取り上げて議論を展開する．

2．知識編集による学習

学習の本質は，ある意図に基づいて，外界から必要な知識を取り込み，それらを既存知識も含めて適切な表現形式に変換するという知識編集にあると筆者は考えている（図１）．編集に当っての意図としては，１）これまで解けなかった問題に対処できるようにする，２）問題解決をより高速化する，３）よりうまい説明をできるようにする，４）知識の整合性解析を効率良くできるようにする，５）意味解析を効率良くできるようにする，など様々なものが考えられる．編集の対象となる知識は，編集の意図を達成するために必要となる知識の全てであり，それは既存の知識やそれから演繹される知識だけでなく，必要に応じて外界から取り込まれる．入力知識の形式も，自然言語，イメージ，図形，あるいはプログラム，プロダクションルール，ファクト（事実）など様々である．また，知識表現が知識利用と密接に関連しているのと同様に，編集後の知識表現形式も編集の意図に大きく依存する．

これら様々な編集意図や知識形式の全てに対処可能な知識編集方法，および膨大な情報量を有する外界からの知識取り込み方法は，余りにもテーマが大きく，かつ大部分が未解明の状態である．故に，本論文では，

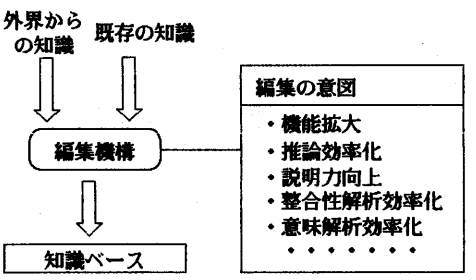

図１　知識編集による学習

編集の意図を問題解決の高速化に絞り，知識形式をルールと事実に限定し，外界からの知識は既に与えられているものとする．問題解決の高速化とは推論速度の向上を意味し，知識編集によって，これまで解けなかった問題が解けるようになるわけではない．Ａｎｄｅｒｓｏｎは，汎化や分化といった，これまで帰納的学習と考えられていたものを知識編集が再現できることを示したが，類推や誤り訂正を行うための一般的問題解決知識の存在を前提にしている[3]．つまり，一見すると機能の拡大を引き起こすように見えるが，知識編集を行う前から同一機能を持っているわけである．ただし，知識編集によって推論の経路が短縮されるため，高速化が達成される．外界から新しい知識を取り込むことなく，システムの機能を拡大することは不可能であることをここで強張しておきたい．

問題解決の高速化を意図した知識の編集機構として，これまでに提案されているものに，①合成(composition)，②手続化(proceduralization)，③chunking，④macro-operator，⑤構造化（structuring)などがある．①～④は特定の問題解決過程のトレースを記憶し，そこで用いられた知識をまとめることにより，同一あるいは類似の問題に対する再計算を回避するという共通点を持つ．これら相互の比較については既に報告されているので[4]，ここでは取り上げない．⑤の構造化は，問題解決過程を編集する①～④に対し，ルールベースの静的解析から得られるメタ知識を用いて断片的ルールを有機的に結合する．この構造化は，新しいルールを生成しないという理由からか，これまでは学習機構として捉えられていなかった．しかし，構造化は，問題解決の高速化という意図のもとに，断片的ルールをネットワーク状に形式変換しており，知識編集による学習と考えるのは自然である．

合成ルールも既存ルールと共に構造化することが可能であり，逆に，構造化されたルールから新しいルールを合成することもできるという意味で，合成と構造化は併用可能である．後述するように，この構造化は合成と同じような効果を発揮するだけでなく，合成にはない種々の利点を持っている．

３．合　成

3.1 合 成 と そ の 効 果

合成は，ある特定の問題を解く際に使用された一連のルールを編集して，それと同じ効果をもつ単一のルールを生成する．合成の簡単な例を以下に示す．今，ＯＰＳ５で書かれた次の四つのルールを考える．

```
(P R1 (PERSON ^NAME Mr.A ^HAVE BALL)
   -->(REMOVE 1)
      (MAKE PERSON ^NAME Mr.B ^HAVE BALL))
(P R2 (PERSON ^NAME Mr.B ^HAVE BALL)
   -->(REMOVE 1)
      (MAKE PERSON ^NAME Mr.C ^HAVE BALL))
(P R3 (PERSON ^NAME Mr.C ^HAVE BALL)
   -->(REMOVE 1)
      (MAKE PERSON ^NAME Mr.D ^HAVE BALL))
(P R4 (PERSON ^NAME Mr.D ^HAVE BALL)
   -->(HALT))
```

もし，作業記憶（ＷＭ）の初期値が

```
((PERSON ^NAME Mr.A ^HAVE BALL))          (1)
```

であるならば，R1,R2,R3,R4 の順に四つのルールが連続して発火する．これらを合成すると次のルール R1' が生成される．

```
(P R1'(PERSON ^NAME Mr.A ^HAVE BALL)
   -->(REMOVE 1)
      (MAKE PERSON ^NAME Mr.D ^HAVE BALL)
      (HALT))
```

これ以後には，（１）のWMに対して，R1とR1'の二つが競合集合に入る．R1とR1'の条件部は全く同じなので，ＯＰＳ５では任意の一つが選択され適用される．このような場合，合成ルールが優先的に発火するようにしておけば，一つのルールR1'を適用するだけで R1,R2,R3,R4 を順次適用したときと同一の結果が得られ，推論効率が向上する．

図２は合成による推論効率の向上を示した実験結果である．横軸は R1～R4 と同様のルールを増やしていったときのルール総数であり，縦軸はＷＭの初期値が（１）のときの推論に要するＣＰＵ時間を表している．合成ルールが存在しないときには，ルール数と共に推論時間は増大するが，合成ルールが存在するときにはほとんど一定である．これらのデータは，ＯＰＳ５に合成機能を追加し，その競合解消戦略によって一意にルールを決定できないときには合成ルールを優先させるように修正したものからとった．

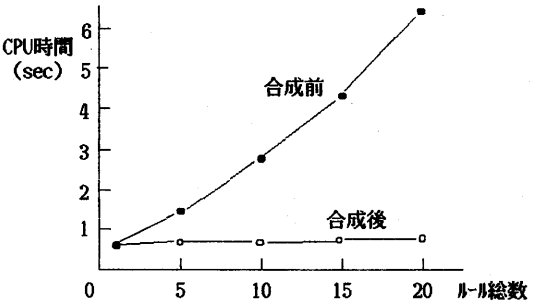

図２　ボール渡しにおける推論時間の比較
(Sun3/260C Sun Common Lisp Interpreter)

3.2 合成の問題点

前節で述べたように，合成は推論効率を向上させるという利点がある反面，以下のような問題点も存在する．

（1）ルール数の増大：前記の R1～R4 に対して，WMの初期値が，

```
((PERSON ^NAME Mr.B ^HAVE BALL))          (2)
```

のときには，R2,R3,R4が連続して発火し，次のルールが合成される．

```
(P R2'(PERSON ^NAME Mr.B ^HAVE BALL)
   -->(REMOVE 1)
      (MAKE PERSON ^NAME Mr.D ^HAVE BALL)
      (HALT))
```

また，WMの初期値が，

```
((PERSON ^NAME Mr.C ^HAVE BALL))          (3)
```

のときには，R3,R4 が連続して発火し，次のルールも合成される．

```
(P R3'(PERSON ^NAME Mr.C ^HAVE BALL)
   -->(REMOVE 1)
      (MAKE PERSON ^NAME Mr.D ^HAVE BALL)
      (HALT))
```

これら単純な四つのルールからも三つのルールＲ１'～Ｒ３'が合成され，合計７個のルールになる．

このように，特定の問題を解く度に合成を行うと，実際的なルールベースは一般に大規模複雑なので，合成ルールの数は次第に膨大なものになる．これは，プロダクションメモリを浪費するだけでなく，競合集合を求める計算量の増大，競合解消の手間の増大を引き起こす．故に，合成されたルールを無条件で追加せずに，その使用頻度などを勘案して取捨選択する方法が実用上は必要になってくる．

（2）整合性維持の複雑化：合成前のルール（原ルール）と合成されたルール（合成ルール）とが混在した状態では，ルールの追加／削除に際してルールベースの整合性を維持することが厄介な問題となる．例えば，原ルールＲ１～Ｒ４と合成ルールＲ１'～Ｒ３'が混在している状態において，ルールＲ３を次のように変更した場合を考える．

```
(P R3*(PERSON ^NAME Mr.C ^HAVE BALL)
   -->(REMOVE 1)
      (MAKE PERSON ^NAME Mr.E ^HAVE BALL))
```

このとき，WMの初期値が(１)ならば，Ｒ３の変更に関わらず合成ルールＲ１'が発火し，WMは((PERSON ^NAME Mr.D ^HAVE BALL))となり終了する．しかし，原ルールだけのときには，Ｒ１，Ｒ２，Ｒ３*が連続して発火し，WMは最終的に((PERSON ^NAME Mr.E ^HAVE BALL))となり，異なる結果が得られる．本来，合成ルールは原ルールと同じ結論に到達しなければならない．そのためには，Ｒ３の変更と同時にＲ１'も次のように変更する必要がある．

```
(P R1'*(PERSON ^NAME Mr.A ^HAVE BALL)
    -->(REMOVE 1)
       (MAKE PERSON ^NAME Mr.E ^HAVE BALL))
```

同様の理由で，Ｒ３を含む合成ルールＲ２'およびＲ３'も変更しなければならない．逆に，合成ルールを変更したときにも，原ルールとの整合性がとれなくなるので，適当な修正が必要になる．これは，合成ルールが本質的に冗長であることに起因する．言い替えれば，合成は冗長性の増大という代価を払って問題解決の高速化を図っているわけである．

（3）合成による遮蔽現象：合成ルールの存在によって，生起すべき原ルールの発火系列が遮蔽されることがある．この遮蔽現象によって推論結果が誤ったものになる場合の例を示そう．原ルールとしてＲ１～Ｒ４，および次のルールＲ５を考える．

```
(P R5 (PERSON ^NAME Mr.B ^HAVE BALL
              ^STATE BUSY)
   -->(HALT))
```

WMの初期値が（１）のときには，Ｒ１～Ｒ４が連続発火し，Ｒ１'が合成される．その後，WMが次のように初期化されたとする．

```
((PERSON ^NAME Mr.A ^HAVE BALL)(PERSON
  ^NAME Mr.B ^HAVE NIL ^STATE BUSY))      (4)
```

このとき競合集合は（Ｒ１，Ｒ１'）となり，合成ルールであるＲ１'が発火し，ボールはＤ氏に渡されて停止する．しかし，もしＲ１'がないときには，異なる結果となる．つまり，まずＲ１が発火し，競合集合は（Ｒ２，Ｒ５）となる．次に，Ｒ５の方がＲ２よりも条件がきついのでＲ５が発火し，ボールはＢ氏に渡されて停止する．このように，合成ルールの存在が，正しいルール発火系列を遮蔽してしまう．

システムが経験する問題解決の順序によっては，この遮蔽現象が起きないこともある．ルールベースがＲ１～Ｒ５だけのときに，即ち，Ｒ１'がまだ合成されていない状態において，WMが（４）に初期化される

と，Ｒ１，Ｒ５が連続して発火し，次のルールが合成される．

```
(P R1"(PERSON ^NAME Mr.A ^HAVE BALL)
      (PERSON ^NAME Mr.B ^STATE BUSY)
   -->(REMOVE 1)
      (MODIFY 2 ^HAVE BALL)(HALT))
```

この合成後に，WMが（１）に初期化されるとＲ１～Ｒ４が連続発火し，前記のルールＲ１’が合成される．この段階，即ち，Ｒ１～Ｒ５，Ｒ１’Ｒ１”が存在している状態においては遮蔽現象は発生しない．即ち，（１）のWMに対してはＲ１’が適用され，（４）のWMに対してはＲ１”が適用されるからである．このように，Ｒ１”が合成された後にＲ１’が合成されれば問題は生じないが，最初にＲ１’が合成されるとＲ１，Ｒ５という発火は起こらないのでＲ１”は永久に合成されない．

（４）合成ルールの妥当性：Ｒ１～Ｒ４からＲ１’を合成するにあたっては，Ａ氏の持っていたボールが最終的にＤ氏に渡されればよいという前提をおいていた．故に，合成ルールＲ１’の動作部を見ればわかるように，途中誰を介してボールがＤ氏に渡ったのかは省略されている．もし，ボールではなく回覧版であったならば，Ｄ氏だけでなく，途中のＢ氏やＣ氏にも渡す必要があるので，途中結果を省略することはできない．この例が示すように，合成にあたってはルールの意味も考慮しないと問題が生じる可能性がある．

筆者等は，ＯＰＳ５（Common Lisp 版）の一部修正により合成機能を持つプロダクションシステムを試作した．本システムは対話型であり，推論終了後に表示される合成ルールをユーザの判断により取捨選択できるようになっている．故に，合成ルールの意味的妥当性を人間がチェックでき，また，前述のルール数の不必要な増大や原ルールとの不整合などにも対処できる．特に，ルールベースの開発段階では誤った推論が行われる確率が高いが，その都度不適当な合成ルールを棄却することが可能となる．

（５）学習の効率性：合成はある特定の問題解決が実行された後で，その過程を編集することによって行われる．故に，問題解決の経験を積むにつれて合成ルール数が次第に増加し，システムはそれらが適用可能な問題を以前より迅速に解決できるようになる．しかし，過去に経験していない問題に対しては，原ルールのレベルで推論が行われるので性能は初期状態と同じである．従って，システムがカバーしている問題領域全般に渡って性能を向上させるためには，数多くの問題を漏れなくシステムに与えてやる必要がある．そして，その都度，推論ならびに合成処理を実行しなければならない．これは人間の学習と類似してはいるが，適切な問題を漏れなく自動生成する機能をシステム自身に持たせたとしても，学習効率が良いとは言いがたい．

<ネット名>	<利用メタ知識>
①ＲＥＴＥネット	条件部相互の類似性
②排他ＲＥＴＥネット	条件部相互の類似性 条件部相互の排他性
③連想ＲＥＴＥネット	条件部相互の類似性 ルール相互の支持関係

表１　各種ＲＥＴＥネットで利用されるメタ知識

４．構　造　化

4.1 基 本 的 考 え 方

構造化はルール相互の関係に関する知識（一種のメタ知識）を抽出して，ルールベースを組織化する．この方法はプロダクションシステム実行に要する計算負荷の大部分を占めるパタン照合を高速化する目的で研究されてきた．この流れをくむ研究としては，①ＲＥＴＥネット[9]，②排他ＲＥＴＥネット[10]，③連想ＲＥＴＥネット[11]などがある．これら３種類の構造化で利用されているメタ知識は表１の通りである．

①はプロダクションシステム記述言語ＯＰＳのために開発されたもので，有名なＲＥＴＥアルゴリズム[4]で使用されているネットである．②及び③は筆者等が開発したもので[7),5]，ＲＥＴＥアルゴリズムの利点を全て継承すると共に，無駄なパタン照合の除去に成功している．特に，③は①および②の利点を含んでおり，これら三つの中で最も推論効率がよい．また，③は前述の合成と機能的に類似した側面を持つだけでなく，合成にまつわる諸問題を回避できるという大きな長所を有している．故に，次節以降では，連想ＲＥＴＥネットを説明すると共に，合成における問題点がいかにして解消されるかについて考察する．

構造化は，基本的に，既存のルールベースにルールを追加／削除する時点で行われる[14),15]．そこでの基本的情報処理は，追加／削除ルールと既存ルールとのパタン照合であり，その結果得られるルール相互の関係に基づいてルールを逐次編集する．人間の場合も，新しい知識が入ってきたときに意識的あるいは無意識的に既存知識との照合を行い，知識を相互に関連付けながら記憶していると思われる．類似の知識があれば両者の違いを明確化したり，矛盾した知識があればどちらが正しいのかを調べたりする．そして，その関連性を反映した形で人間の記憶に入るものと思われる．

合成が特定の問題解決過程を編集するのに対し，構造化は問題解決とは独立に行われる．推論実行時に編集が行われるという意味で合成を“動的編集”と呼ぶならば，ルールベースの静的解析により編集を行う構

造化は“静的編集”と言える．また，合成は特定の問題解決に使用された一部のルールだけを編集の対象とする“局所的編集”であるのに対し，構造化はルールベース全体を対象とした“大局的編集”である．

4.2 連想ＲＥＴＥネット

合成のところで用いたルールＲ１～Ｒ４を再び取り上げて連想ＲＥＴＥネットの知識編集を説明する．連想ＲＥＴＥネットは，ルールＲ１～Ｒ４を図３のような形に編集したものである．ここで，矢印の付いた曲線のリンクを“連想リンク”と言い，これはルール相互の支持関係を表している．連想ＲＥＴＥネットからこの連想リンクを全て除去したものがＲＥＴＥネットに対応している．つまり，ＲＥＴＥネットは連想ＲＥＴＥネットのサブネットである．ＲＥＴＥネットの部分では，知識の類似性を利用して同一条件を一つのノードにまとめている（例えば，図３の PERSON および HAVE= BALL）．

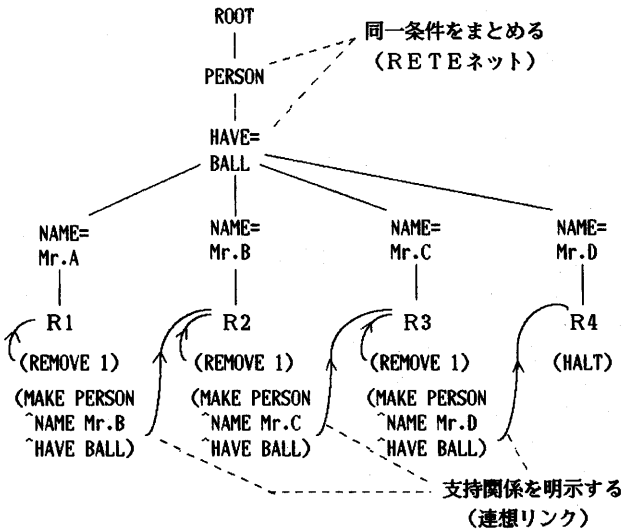

図３　構造化によりできる連想ＲＥＴＥネット

ＷＭの初期値が（１）のとき，そのＷＭ要素はトークンとしてＲＯＯＴノードから流され，各ノードでのテストを受けた後，端末ノードＲ１に到達する．次に，Ｒ１の動作部が実行され，追加／削除されるＷＭ要素が，ＲＯＯＴノードからではなく，連想リンクをたどって流される．故に，この例では，Ｒ１の発火後はＷＭとルール条件部とのパタン照合を全く行わずにＲ２，Ｒ３，Ｒ４が引続き発火する．故に，連想ＲＥＴＥネットでは合成とほぼ同じような推論効率が得られる．

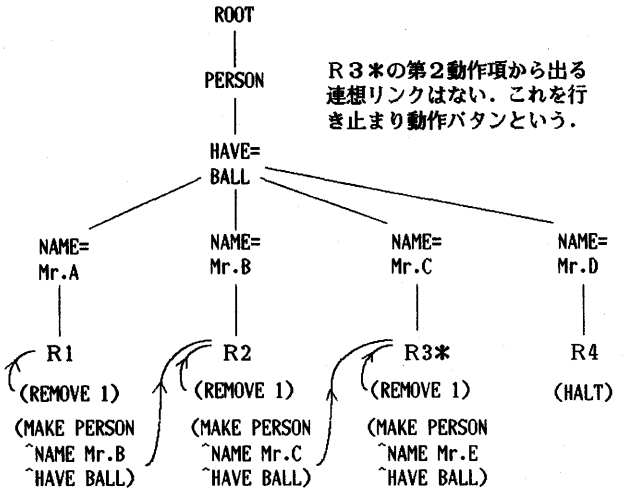

図４　ルール修正時の連想ＲＥＴＥネットの変化

この連想ＲＥＴＥネットは，ルールベースを一括編集して求めることも可能である[11]．しかし，ルールの追加／削除時に逐次的に既存連想ＲＥＴＥネットを修正する方が自然であり，かつ効率的である[15]．特に，ルールベースが大規模になり，ごくわずかのルールを追加／削除する状況では，極めて効率的である．

4.3 構造化の長所

ここでは，連想ＲＥＴＥネットの形で構造化する知識編集が，前章で述べた合成の諸問題をいかに解決できるかを示す．

（１）ルール数増大の問題：合成では，ルールＲ１～Ｒ４から三つの合成ルールＲ１’～Ｒ３’が生成され，ルール数が増大するという問題があった．一方，連想ＲＥＴＥネットでは，Ｒ１～Ｒ４が一旦図３の形に編集されると，外部から新しいルールが追加されない限り知識編集は発動されない．従って，問題解決を何回経験してもルール数は増大せず，プロダクションメモリは浪費されない．推論実行に際しては，最初に発火すべきルールを求めるためのパタン照合だけで適切なルールが連続的に発火する．これは合成ルールが存在しているときとほぼ同じ推論効率をもつ．ただし，ＷＭ要素の追加／削除と競合解消は各ルールが発火する度に行われるので，推論効率は合成に比べ若干劣る．

（２）整合性維持の問題：原ルールＲ１～Ｒ４と合成ルールＲ１’～Ｒ３’が混在している状態において，原ルールあるいは合成ルールを変更すると，原ルール

レベルと合成ルールレベルでの推論結果に食い違いが生じるという問題があった．例えば，ルールＲ３をＲ３* のように修正したときには，Ｒ３を含む合成ルールＲ１'～Ｒ３'の全てを修正する必要があった．一方，連想ＲＥＴＥネットにおいてルールＲ３をＲ３* に変更すると，図３のネットは図４のように修正される．つまり，Ｒ３の第２動作項の Mr.D が Mr.E に変化し，そこから出ていた連想リンクが削除される．この修正はネット中のＲ３に関連した部分だけで済み，合成ルールが存在するときのような整合性解析を含む複雑な処理は必要ない．にも関わらず，WMの初期値（１）～（３）に対しては，Ｒ３の修正の影響が推論結果に正しく反映される．

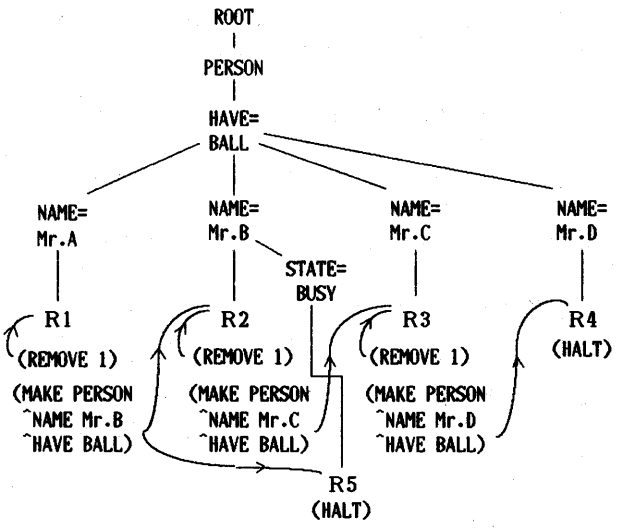

図５　ルールＲ１～Ｒ５に対する連想ＲＥＴＥネット

（３）遮蔽現象の問題：連想ＲＥＴＥネットでは合成によって生じた遮蔽現象は発生しない．ルールＲ１～Ｒ５に対する連想ＲＥＴＥネットは図５のようになる．WMの初期値が（１）のときのルール発火系列は，Ｒ１，Ｒ２，Ｒ３，Ｒ４であり，WMの初期値が（４）のときのそれは，Ｒ１，Ｒ５となる．ルールＲ１の次にＲ２が発火するか，あるいはＲ５が発火するかは，条件がきついルールを優先するというＯＰＳ５の競合解消規則によって適切に決定される．

合成において遮蔽現象を回避するためには，例えば，次のような工夫が必要となる．発火系列Ｒ１，Ｒ２，Ｒ３，Ｒ４を合成する際に，これらのルールの中に，これら以外のルールと包含関係にあるルールが存在するかどうかを調べる．Ｒ５が成立するときには，必ずＲ２も成立するのでＲ２はＲ５に包含されている．そこで，これら二つのルールの条件部が互いに排他関係になるように修正する．この場合，Ｒ２を次のように変えればよい．

```
(P R2*(PERSON ^NAME Mr.B ^HAVE BALL
              ^STATE <> BUSY)
   -->(REMOVE 1)
      (MAKE PERSON ^NAME Mr.C
                   ^HAVE BALL))
```

そして，発火系列Ｒ１，Ｒ２*，Ｒ３，Ｒ４を合成すれば，Ｒ１'とは異なる次のルールが生成される．

```
(P R1* (PERSON ^NAME Mr.A ^HAVE BALL)
       (PERSON ^NAME Mr.B ^STATE <> BUSY)
   -->(REMOVE 1)
      (MAKE PERSON ^NAME Mr.D ^HAVE BALL)
      (HALT))
```

これ以降に，WMとして（４）が与えられると，R1* は発火せず，Ｒ１，Ｒ５が発火してルールＲ１" が合成される．このように遮蔽現象はなくなが，ルールベースが大規模複雑になると，以上の処理の計算手間は相当なものになる．

（４）編集の妥当性の問題：合成においては，R1' の例でもわかるように，推論の途中結果を省略することができ，それによって意味的に誤ったルールが生成されることがある．しかし，連想ＲＥＴＥネットではWMとルール条件部とのパタン照合は省略されても，動作部の実行は編集前と全く同じである．故に，原ルールが意味的に問題なければ，編集によって問題が生じることはない．連想ＲＥＴＥネットでは，ボール回しでも，回覧版回しでも，常に途中の人にそれが渡される．つまり，連想ＲＥＴＥネットは途中結果の必要性に関わらず，途中結果をWMに追加したり削除したりする．これは利点である反面，連想ＲＥＴＥネットの推論効率が合成に比べて若干劣る要因になる．推論の中間結果が多く存在し，それらが全く不必要な場合には合成ルールの存在価値は大きい．

（５）学習効率の問題：連想ＲＥＴＥネットにおける知識編集はルールの追加／削除時だけに行われる．一方，合成は特定の問題を解いたときに複数のルールが用いられると実行される．初期段階では適切な合成ルールが既に生成されているとは限らないので，合成処理が頻繁に行われる．このように，編集を行う回数が，連想ＲＥＴＥネットでは一回だけであるのに対し，合成は複数ルールが発火するたびに行われる．これは，連想ＲＥＴＥネットが大局的編集であるのに対し，合成は局所的編集であることに起因している．局所的編集を漏れなく実行するためには，数多くの問題を作り，

それをシステムに与えて実際に推論を実行し，その都度，合成処理を行わなければならない．

4.4 逐次編集による連想ＲＥＴＥネットの構築

本節では，既存連想ＲＥＴＥネットを効率的に修正する逐次編集法を提案する．ルールの追加／削除にあたって必要となる連想ＲＥＴＥネットの変更はごく一部であるため，変更後のルール集合を一括処理して新しい連想ＲＥＴＥネットを作り直す方法はかなり多くの無駄を含む．その無駄は，僅かな修正が頻繁に起きるほど，また，ルールベースが大規模になるほど増大する．この無駄を回避するには，ルールベースの変化から既存ネットの変化だけを求めればよい．この考え方に基づく連想ＲＥＴＥネットの逐次編集処理の概要を以下に示す．

[1]連想ＲＥＴＥネットの一部を構成するＲＥＴＥネットの部分だけをまず修正する．ルールの追加なら[2]へ，削除ならば[4]にとぶ．

[2]追加ルールの動作項の連想ノードを求め，連想リンクを付加する．

[3]上記[1]によって追加されたノードの中に，既存ルールの動作項の連想ノードが存在するならば，それに対応する連想リンクを付加し，処理を終了する．

[4]削除ルールの動作項から出ていた連想リンクを削除する．

[5]上記[1]によって削除されたノードに入っていた連想リンクを除去し，処理を終了する．

上記[1]はＲＥＴＥネットの逐次コンパイルであり，ＯＰＳ５（Ｌｉｓｐ版）で既に実現されている．ただし，ＯＰＳ５ではルール削除時に単にそのルールの発火が禁止されるだけであり，実際にＲＥＴＥネットは修正されない[13]．そのため，推論時には，削除したはずのルールに対応するノードにもトークンが流されるという無駄が行われる．この推論時の無駄をなくすと共に，逐次コンパイルに必要なパタン照合を高速化した改良アルゴリズムも提案されている[14]．上記[1]の処理はこのアルゴリズムをそのまま利用することができる．

[2]の処理は連想ノードを一括して求める方法で使用されているアルゴリズム[11]を流用できる．即ち，全ルールの動作パタンではなく，追加ルールの動作パタンだけをＲＥＴＥネットのルートノードからトークンとして流せばよい．[4] の処理は単純であり説明するまでもない．また，[5] の処理は連想リンクを逆向きにたどれるようにしておけば簡単に実現できる．故に，以下では[3]の実現方法を簡単に述べる．

ルール追加時における既存動作項の連想ノードの変化は，追加ルールの条件パタンを支持する可能性のある既存動作パタンだけをＲＥＴＥネットに流せば求められる．既存動作パタンが追加ルールの条件パタンを支持するための必要条件は，その動作パタンのクラス名と同じクラス名を持つ条件パタンが追加ルールの中に存在することである．なぜならば，もしクラス名が異なっているならば，支持する可能性は絶対にないからである．故に，ルールが追加されたときに，その条件パタンのクラス名から，それと同一のクラス名をもつ動作パタンをもつ既存ルールを取り出せるようにしておけばよい．具体的な処理アルゴリズムならびにその評価についての詳細は別途発表する予定である[15]．

５．あ と が き

本稿では，知識編集による学習機構の一つである合成の問題点を明らかにすると共に，構造化という知識編集の結果できる連想ＲＥＴＥネットが，それらの問題点を回避できることを示した．また，連想ＲＥＴＥネットを，知識の追加／削除時に逐次的に修正する効率的編集方法も提案した．合成と構造化は併用可能であるが，両者の特性を生かした使い分けについては更に研究が必要である．また，知識編集と整合性解析[16]との関係については本稿では触れなかったが，整合性解析を伴わない知識の編集は危険であり，両者の統合は今後の重要課題である．

参 考 文 献

1) Anderson, J.R.: The Architecture of Cognition, Harvard Univ. Press (1983)

2) Lewis,C.: Production System Models of Practice Effects, Ph.D.diss., Univ. of Michigan (1978).

3) Anderson, J.R.: Knowledge Compilation: The Geral Learning Mechanism, in Michalski, Carbonell and Mitchell (Eds): Machine Learning, Vol. 2, Morgan Kaufmann Pub.Inc. pp.289-310 (1986).

4) Rosenbloom,P.S. and Newell,A.: The Chunking of Goal Hierarchies: A Generalized Model of Practice, in Michalski,Carbonell and Mitchell (Eds) : Machine Learning, Vol.2, Morgan Kaufmann Pub.Inc. 289-310 (1986).

5) Laird J.E., Newell,A. and Rosenbloom,P.S.: SOAR:An Architecture for General Intelligence, Artificial Intelligence, 33, 1-64 (1987)

6) Samuel,A.L.: Some Studies in Machine Learning Using the Game of Checkers,II-Recent Progress, IBM Journal of Research and Development, 11, (1967)

7) Michie,D.: Memo Functions and Machine Learning, Nature, 218, 19-22 (1968)

8) Korf,R.E.: Macro-operators: A Weak Method for Learning, Artificial Intelligence, 26, 35-77 (1985)
9) Forgy,C.L.: Rete: A Fast Algorithm for Many Pattern/Many Object Pattern Match Problem, Artif. Intell., Vol.19, pp.17-37 (1982).
10)荒屋ほか：プロダクションシステムのための高速パターン照合アルゴリズム，情報処理学会論文誌，Vol.28, No.7,pp.768-755 (1987).
11)荒屋ほか：プロダクションシステムにおける効率的パタン照合のための連想ＲＥＴＥネットワーク表現，情報処理学会論文誌（投稿中）.
12)Lewis,C: Composition of Productions, in Klahr, D., Langley,P. and Neches,R.(Eds): Production System Models of Learning and Development, MIT Press (1987).
13)Brownston,L. et al.: Programming Expert Systems in OPS5 : An Introduction to Rule-Based Programming, Addison-Wesley (1985).
14)荒屋ほか：知識ベースの逐次構造化－ＲＥＴＥネットワークの逐次構築法－，電子情報通信学会論文誌，Vol.71, No.6 (1988).
15)荒屋ほか：連想ＲＥＴＥネットワークの逐次構築法，情報処理学会論文誌（投稿中）.
16)荒屋ほか: ルールベースのインクリメンタルメンテナンス，電気関係学会九州支部第40回連合大会 (1987).