ICOT Technical Memorandum: TM-0667

TM-0667

仮説推論システムAPRICOT/Oによる知識コンパイル

井上克巳、太田好彦

January, 1989

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456 3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

仮説推論システムAPRICOT/0による知識コンパイル

Knowledge Compiling on Hypothetical Reasoning System APRICOT/0

井上　克巳　・　太田　好彦
Katsumi INOUE　　Yoshihiko OHTA

（財）新世代コンピュータ技術開発機構　第五研究室
Fifth Research Laboratory, Institute for New Generation Computer Technology

Abstract

This paper describes the knowledge(rule) compiler on a hypothetical reasoning system called APRICOT/0. This compiler is an incremental compiler for Rete-like network. It is implemented by using ATMS. This method has a unique feature of the function of removing rules. If a user indicates condition elements to remove rules then this system removes all rules which contains the condition elements in their if-parts. The removing efect remains valid after that. After the removal, if the user adds a rule which contains the condition elements, the system neglects the rule. This method has the functions of rule-base maintenance and rule compiling.

１．はじめに

仮説推論システムＡＰＲＩＣＯＴ／０は、ＡＴＭＳ［ｄｅ　Ｋｌｅｅｒ　８６ａ］とルール・ベースの問題解決器とを融合した仮説推論システムＡＳＴＲＯＮ［藤原　他　８８］を発展させたものであり、ＡＰＲＩＣＯＴ［Ｉｎｏｕｅ　８８］の試作第０版という位置付けである。

ＡＳＴＲＯＮは、従来のヒューリスティック・ルールに加え、デフォルト知識・論理的推論ルールが扱えるようになっている。さらに、制約も扱えるようにすることを課題としていた。［ｄｅ　Ｋｌｅｅｒ　８６ｂ］　は、ＡＴＭＳの問題解決インタフェースとして制約言語を採用した研究である。これは、コンシューマ（データフロー解析後の制約）をノードに付加してデモンによる起動を提案していたが、ＡＰＲＩＣＯＴ／０では、制約の集合の解析（プリコンパイル）を通じて、ルール形式に変換して扱う。これにより、従来のヒューリスティック・ルールに加え、深い知識（構造や機能の制約知識）や常識（デフォルト知識）を統一的に利用できるようにしている。一般に、知識コンパイルとは深い知識を浅い知識に変換する技法であるが、本稿で知識コンパイルはこの考察によりルール・コンパイルとしている。

ＡＳＴＲＯＮと同様にＡＴＭＳとルール・ベースの問題解決器とを融合した仮説推論システムはいくつか提案されている。一方、不完全な知識（常に成立するとは限らない知識）等を取り扱う仮説推論システムは、ルールの追加・削除が頻繁におこると考えられる。したがって、インクリメンタルなルール・コンパイルが要求される。［飯島　他　８７］や［飛鳥井　他　８８］はＡＴＭＳの問題解決インタフェースとしてルール・ベースを採用した研究であり、ＡＴＭＳとＲｅｔｅアルゴリズム［Ｆｏｒｇｙ　８２］とを融合しラベル計算とパターン・マッチングをＲｅｔｅネットワーク内で同時に行い推論速度の向上を図っている。しかしながら、［飯島　他　８７］や［飛鳥井　他　８８］は、Ｒｅｔｅネ

ットワークの構築法を目的とした研究ではない。Ｒｅｔｅネットワークのインクリメンタル構築法については、［荒屋　他　８８］等で論じられている。これは、Ｒｅｔｅネットワークのインクリメンタル構築法にＲｅｔｅアルゴリズムを用いるというところに着目した研究である。

本稿では、ＡＰＲＩＣＯＴ／０のルール・コンパイル方法について述べることとする。これの特徴は、独自のＲｅｔｅ－ｌｉｋｅネットワークのインクリメンタル構築法を提案し、ルールの追加・削除を容易にしているところにある。これのインクリメンタル構造化法は、ＡＴＭＳのラベル計算に着目している。したがって、［荒屋　他　８８］とは異なった知識ベース管理機構が提供されている。また、［飯島　他　８７］や［飛鳥井　他　８８］と同様に、ＡＴＭＳとＲｅｔｅアルゴリズムとを融合しラベル計算とパターン・マッチングをＲｅｔｅ－ｌｉｋｅネットワーク内で同時に行い推論速度の向上を図っている。

２．構成

図１は、本システムの構成を示すものである。

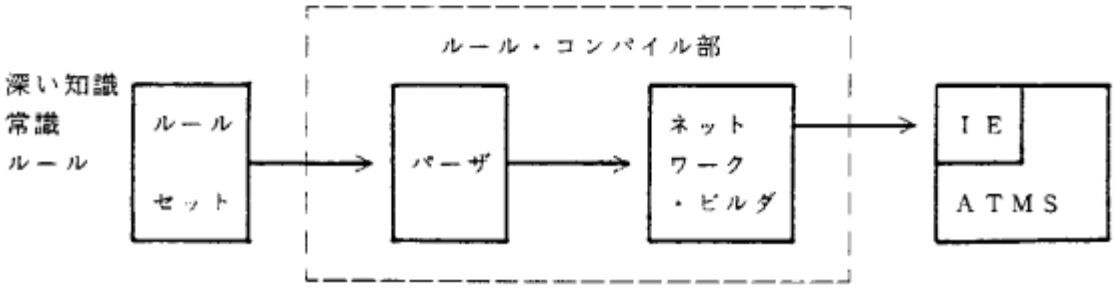

図１　構成図

ユーザが、直接編集したルール・ベースをパーザにかけ、ネットワーク・ビルダが、Ｒｅｔｅ－ｌｉｋｅネットワークを構築する。パーザとネットワーク・ビルダが、ルール・コンパイル部である。ここで、ネットワーク・ビルダは、ＡＴＭＳを利用している。

ルール・コンパイル部が、出力するＲｅｔｅ－ｌｉｋｅネットワークに基づき、推論エンジンＩＥが推論を実行する。推論エンジンＩＥは、前向き推論エンジンとなっている。この前向き推論エンジンのワーキング・メモリーＷＭは、Ｒｅｔｅ－ｌｉｋｅネットワークにより、分散されたワーキング・メモリー（分散ＷＭ）となっている。また、ワーキング・メモリーの要素であるワーキング・メモリー要素ＷＭＥは，ＡＴＭＳのノードとなっている。

ルールは、それまでに得られた事実や仮説とのマッチングを、全てのコンテキストを考慮に入れて行う＜条件部＞と、事実・仮説の追加及び、それに対する理由付けをＡＴＭＳに与える＜実行部＞からなる。条件部の各条件要素は、対象となる事実や仮定に対応するＡＴＭＳノードが信じられている（あるコンテキストに含まれている；これをＩＮと呼ぶ）ならば真となる述語・変数・アトム、あるいは手続きの実行が成功するならば真となる数値計算式・述語呼び出しのいずれかである。条件部の全ての条件が真になるとルールが発火し、その結果、＜条件部＞　＝＞　＜実行部＞の形式で、理由付けがＡＴＭＳに与えられる。

３．ルール・コンパイル方法

本システムでは、独自のＲｅｔｅ－ｌｉｋｅネットワークのインクリメンタル構築法を採用し、ルールの追加・削除を容易にしている。本方法は、Ｒｅｔｅネットワークのインクリメンタル構築法にＡＴＭＳが利用できるというところに着目している。すなわち、ＡＴＭＳが提供する環境束（Ｅｎｖｉｒｏｎｍｅｎｔ　Ｌａｔｔｉｃｅ）は、Ｒｅｔｅ－ｌｉｋｅネットワークに類似している。この点に着目し、ネットワーク・ビルダをＡＴＭＳモジュールを利用して効率的にインプリメントした。

本章では、ルール・コンパイル方法について述べるために、まずＡＴＭＳモジュールのインプリメントを示す。また、ルール・コンパイル方法について述べた後にその出力を用いる推論エンジンについて述べる。

３．１　ＡＴＭＳモジュール

ＡＴＭＳが管理するデータは、アトム、述語である。述語の場合、変数を含まないものとしている。

データに対応してＡＴＭＳ内部では、ノードが生成される。ここに、ノードは、データで一元的に管理されている。すなわち、等しいデータのノードは、だだ一個に限定される。以下に、ノードのデータ構造を示す。

Ｄａｔｕｍ：データ
Ｌａｂｅｌ：環境の集合（環境へのポインタの集合）
Ｊｕｓｔｉｆｉｃａｔｉｏｎｓ：理由付けノードＡＮＤ結合の集合
（ノードへのポインタの集合の集合）
Ｃｏｎｓｅｑｕｅｎｔｓ：Ｊｕｓｔｉｆｉｃａｔｉｏｎｓの逆ポインタ
（ノードへのポインクの集合）
Ｃｏｎｔｒａｄｉｃｔｏｒｙ：０＝ＩＮ，１＝ＯＵＴ

環境は、ラベル計算の高速化のためにビット・ベクタ表現［ｄｅ　Ｋｌｅｅｒ　８６ａ］を採用し以下のデータ構造である。また環境は、ビット・ベクタで一元的に管理されている。

Ａｓｓｕｍｐｔｉｏｎｓ：ビット・ベクタ
Ｎｏｄｅｓ　　：Ｌａｂｅｌの逆ポインタ（ノードへのポインタの集合）

すなわち、等しいビット・ベクタでの環境は、だだ一個に限定している。これを管理するのが、環境表であり、環境表はハッシュ・テーブルとなっている。

他に、Ｎｏｇｏｏｄ　Ｄａｔａｂａｓｅがあり、これは矛盾する環境を極小（Ｍｉｎｉｍａｌ）で管理している。

３．２　ネットワーク・ビルダ

以下、上記ＡＴＭＳモジュールを利用したＲｅｔｅ－ｌｉｋｅネットワーク・ビルダのインプリメント方法について述べる。先に示したノードのデータ構造で分かるように、ノードの集合は、Ｊｕｓｔｉｆｉｃａｔｉｏｎｓ及びＣｏｎｓｅｑｕｅｎｔｓのポインタ連鎖でネットワーク表現される。しかしながら、これを利用しただけでは、依然として、Ｒｅｔｅアルゴリズムの要旨である２入力ノードによるファクタリングの問題が残るので、本方法では、このポインタ連鎖は利用していない。本方法は、ＡＴＭＳの環境に着目している。

まず、ＡＴＭＳモジュールを以下のように修正する。
ＡＴＭＳの環境のデータ構造に以下のスロットを追加する。

Ｓｕｐｅｒ：無矛盾な上位環境へのポインタの集合
Ｓｕｂ：下位環境へのポインタの集合
Ｗｍ：分散ワーキング・メモリ（分散ＷＭ），ＡＴＭＳノードの集合

［ｄｅ　Ｋｌｅｅｒ　８６ａ］によれば、ビット・ベクタ表現された環境の組み合わせ計算は、論理演算ｏｒで高速化できる。このとき、普通は、２環境ごとにｏｒを計算する。したがって、この点に着目すればＲｅｔｅアルゴリズムの要旨である２入力ノードの実現が簡単に行える。そこで、ネットワークを表現するために、環境の組み合わせを生成するメソッドｏｒに以下の機能を追加し、Ｓｕｐｅｒ，Ｓｕｂリンクで表現されるようにする。すなわち、ｏｒ（Ｅ１，Ｅ２，Ｅ３）：（環境Ｅ１及び環境Ｅ２組み合わせＥ３を求める）のとき、

環境Ｅ１のスロットＳｕｐｅｒに環境Ｅ３へのポインタを追加する。
環境Ｅ２のスロットＳｕｐｅｒに環境Ｅ３へのポインタを追加する。
環境Ｅ３のスロットＳｕｂに環境Ｅ１へのポインタを追加する。
環境Ｅ３のスロットＳｕｂに環境Ｅ２へのポインタを追加する。

なる操作を追加する。
また、ある環境が矛盾となった場合，その環境の　Ｓｕｐｅｒ　Ｓｅｔ　は全て矛盾となる。このとき、矛盾する環境Ｅのスロット Ｓｕｂに含まれる全ての環境のスロットＳｕｐｅｒから環境Ｅを削除することにより　Ｓｕｐｅｒ，Ｓｕｂ　リンクを切ることができる。
次に、このように修正したＡＴＭＳを利用してＲｅｔｅ－ｌｉｋｅネットワークの構築法について述べる。
まず、ｒｏｏｔノードとしてＰｒｅｍｉｓｅノードを１個生成しておく。
次に、１入力ノードであるが、各条件要素をＤａｔｕｍとしたＡｓｓｕｍｐｔｉｏｎノードを生成する。ここで注意することは、ＡＴＭＳのＤａｔｕｍは、命題（変数を含まない述語）のレベルでなければならないということである。しかし、ルールの条件要素は、変数を含む述語となっている。この問題を解決するために、ここでは変数として扱わずその位置に変数がある述語であると解釈する。例えば、ｔｅｍｐｅｒａｔｕｒｅ（Ｒ，Ｔ）などは、第０要素がｔｅｍｐｅｒａｔｕｒｅ、第１要素と第２要素が変数であるスタック・ベクタと解釈するわけである。Ｘは、第０要素が変数であると解釈するわけである。したがって、ｔｅｍｐｅｒａｔｕｒｅ（Ｒ，Ｔ）とＸとは、ユニファイしないと解釈する。同様に、ｔｅｍｐｅｒａｔｕｒｅ（Ｒ，Ｔ）とｔｅｍｐｅｒａｔｕｒｅ（ａ，Ｔ）とは、ユニファイしないと解釈する。先に述べた、ＡＴＭＳノードのＤａｔｕｍ一元性は、Ｒｅｔｅ－ｌｉｋｅネットワークの構築に用いられる時は、このようなユニフィケーション原理にもとづいて行われる。これにより、生成されるＲｅｔｅ－ｌｉｋｅネットワークは、自然にファクタリングがなされる。
次に、２入力ノードであるが、条件要素（１入力ノード）のＡＮＤ結合をＪｕｓｔｉｆｉｃａｔｉｏｎにもつテスト・ノードを生成すれば自然にできる。そのテスト・ノードの生成過程で生成される中間的な環境が２入力ノードとなる。このとき、同時にリンクも張られる。

また、１つのルールでユニファイする左辺条件要素が複数ある場合は、上記の原理に従えば，Ｓｕｐｅｒ，Ｓｕｂリンクのループができてしまう問題がある。この問題を解決するために、１つのルールでユニファイする左辺条件要素が複数ある場合は、別のノードとして扱うことにした。

このように、ＡＴＭＳを利用したＲｅｔｅ－ｌｉｋｅネットワークの構築法には、ルールのインクリメンタルな追加や削除が可能である。以下、ルールのインクリメンタルな追加や削除のアルゴリズムについて示す。ＡＴＭＳは，仮説の動的生成や、仮説の棄却（矛盾の定義）ができるようになっているので、これらの機能を利用することができる。

まず、ルールのインクリメンタルな追加について述べる。これは、新規追加と全く同様である。まず、追加するルールの各条件要素をＤａｔｕｍとしたＡｓｓｕｍｐｔｉｏｎノードを生成する。ＡＴＭＳには、以前生成したＤａｔｕｍのノードは、再び生成しないように管理する機能がある。これで、１入力ノードの共有がなされる。新規追加の条件要素は、新しいＡｓｓｕｍｐｔｉｏｎノードが生成される。このとき、新しい環境（１入力ノード）が生成される。次に、条件要素（１入力ノード）のＡＮＤ結合をＪｕｓｔｉｆｉｃａｔｉｏｎにもつテスト・ノードを生成する。このとき、環境表は、以前生成した環境は再び生成しないように管理している。これにより、２入力ノードの共有がなされる。

次に、ルールのインクリメンタルな削除について述べる。これもまた、ＡＴＭＳの機能を用いている。ルールのインクリメンタルな削除は、削除する条件要素またはそれらの組み合わせを指定する。システムは、Ａｓｓｕｍｐｔｉｏｎとしてのこれらのノードまたはそれらの組み合わせのＬａｂｅｌを計算し矛盾な環境とする。すると、これを含む環境が矛盾となり、Ｒｅｔｅ－ｌｉｋｅネットワーク（環境束）から全て削除される。また、今までに矛盾となった環境は、Ｎｏｇｏｏｄ　Ｄａｔａｂａｓｅにより、極小（Ｍｉｎｉｍａｌ）で管理されており、以後矛盾となるような条件要素またはそれらの組み合わせを含むルールは追加されないという効果がある。

３．３　推論エンジン

次に、推論エンジンの推論アリゴリズムについて述べる。

ルール・セットは、前提となる事実や仮説を直接記述できる。まずこれらがトークンの待ち行列に追加される。推論エンジンは、この待ち行列からトークンを取り出しＲｅｔｅ－ｌｉｋｅネットワークのｒｏｏｔノードに流す。次に、ｒｏｏｔノードとＳｕｂリンクで結合されている１入力ノードにそのトークンを流す。１入力ノードとｎｏｄｅｓリンクで結合されているＮｏｄｅのＤａｔｕｍとマッチングすれば、その１入力ノードとＳｕｂリンクで結合されている他の１入力ノードのＷｍに保持されているＷＭＥとのＡＮＤ結合をとり、中間的なラベル計算をおこなう。このＡＮＤ結合をとった新しいノードをその１入力ノードとＳｕｐｅｒリンクで結合されている２入力ノードに流す。以下、同様な操作を行いＳｕｐｅｒリンクで結合されている２入力ノードがなくなったときにそのノードとｎｏｄｅｓリンクで結合されているテスト・ノードを実行する。その結果新しいデータが生成される。これを、先のトークンの待ち行列に追加する。新しい理由付けが生成された場合および矛盾が生成された場合、即座にＡＴＭＳにそのタスクを送ってやればよい。次に、残っているテスト・ノード、残っているＷＭＥ、残っているＳｕｐｅｒリンクで結合されている２入力ノード、残っている１入力ノードについても同様におこなう。残っているものがなくなった場合に、新しいトークンを待ち行列から取り出し同様におこなう。つまり、推論エンジンは縦型探索をおこなう。このような操作を繰り返しトークンの待ち行列が空になった場合に推論エンジンは停止する。

４．使用例

以下、本ルール・コンパイル方法の効果を簡単な会議開催計画問題の例で示す。

４．１　Ｒｅｔｅ－ｌｉｋｅネットワークの構築例

以下の例を用いてＲｅｔｅ－ｌｉｋｅネットワークの構築について示す。

ａ本人、ａ代理人、ｂ本人、ｂ代理人がいる。また、会議室２１２と会議室２１３とがある。いま、ａとｂを必ず出席させて月曜日または、火曜日に会議を開催する計画をたてる。ただし、ａの代理人は、月曜日都合が悪く、ｂ本人は、火曜日都合が悪い。また、月曜日は全会議室予約済みで、火曜日は、２１３会議室予約済みである。

これをルール化した例を図２に示す。このルール・セットを本方法によりＲｅｔｅ－ｌｉｋｅネットワークに展開したものを図３に示す。

```
%＜ｉｄ＞'::'＜ｉｆ－ｐａｒｔ＞'->'＜ｔｈｅｎ－ｐａｒｔ＞'.'
矛盾１::
        出席可能（Ｘ，Ｐ，Ｗ），
        出席不可能（Ｘ，Ｐ，Ｗ）
->
        []．%[]は、矛盾を表す。

矛盾２::
        空き会議室（Ｎｏ，Ｗ），
        予約済み（Ｎｏ，Ｗ）
->
        []．

出席可能::
        出席候補（Ｘ，Ｐ），
        日（Ｗ）
->
        ａｓｓｕｍｅ（出席可能（Ｘ，Ｐ，Ｗ））．
%ａｓｓｕｍｅ（Ａ）は、仮説Ａの動的生成を表す。

空き会議室::
        会議室（Ｎｏ），
        日（Ｗ）
->
        ａｓｓｕｍｅ（空き会議室（Ｎｏ，Ｗ））．

会議開催::
        出席可能（ａ，Ｉａ，Ｗ），
        出席可能（ｂ，Ｉｂ，Ｗ），
        空き会議室（Ｎｏ，Ｗ）
->
        開催（Ｗ，Ｎｏ，ａ（Ｉａ），ｂ（Ｉｂ））．
```

出席候補（ａ，本人）．
出席候補（ａ，代理人）．
出席候補（ｂ，本人）．
出席候補（ｂ，代理人）．
会議室（２１２）．
会議室（２１３）．
日（月曜日）．
日（火曜日）．

出席不可能（ａ，代理人，月曜日）．
出席不可能（ｂ，本人，火曜日）．
月曜日予約済み：：空き会議室（Ｎｏ，月曜日）－＞［］．
予約済み（２１３，火曜日）．

図２　　ルール化した例

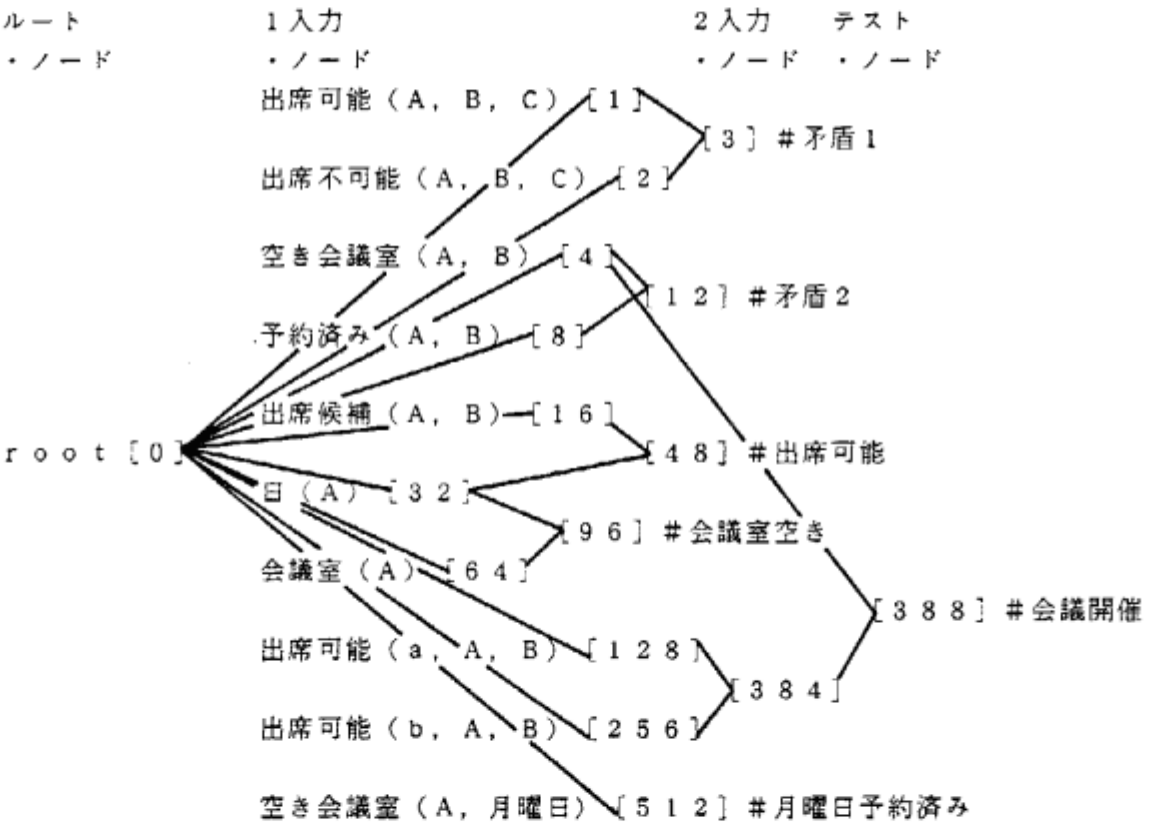

図３　　ネットワークの例
（図２のルール・セット，Ａ，Ｂ，Ｃは、変数を意味する。
［］は、ビット・ベクタ十進表現した環境を示す。）

本方法は、ＡＴＭＳとＲｅｔｅ－ｌｉｋｅアルゴリズムとの融合により［飯島　他　８７］と同様なＬａｂｅｌ計算の効率化が図られる。すなわち、ルール会議開催について、ＡＴＭＳでは、空き会議室（Ａ，Ｂ）にマッチングするものが２つ以上あった場合に、出

席可能（ａ，Ａ，Ｂ）および出席可能（ｂ，Ａ，Ｂ）にマッチングする以前に計算した仮説ノードの組み合わせについてＬａｂｅｌ再計算しなければならない。本方法によれば出席可能（ａ，Ａ，Ｂ）および出席可能（ｂ，Ａ，Ｂ）にマッチングする仮説ノードの組み合わせについては、それらにマッチングするものがそろったときに一度だけＬａｂｅｌ計算し、分散ＷＭに保持されているのでＬａｂｅｌ再計算をする必要がなく効率的である。

４．２　ルールの追加

次に、ルールを追加した場合について示す。

例えば、談話室（２２９）があり、そこでも　会議開催が可能であることがわかったとする。すると、会議開催ルールに対応して会議開催談話室なるルール及び空き会議室ルールに対応して空き談話室なる図４に示すルール・セットが考えられる。このルール・セットを追加した場合のネットワーク構造を図５に示す。

```
会議開催談話室：：
        出席可能（ａ，Ｉａ，Ｗ），
        出席可能（ｂ，Ｉｂ，Ｗ），
        空き談話室（Ｎｏ，Ｗ）
－＞
        開催（Ｗ，Ｎｏ，ａ（Ｉａ），ｂ（Ｉｂ））．

空き談話室：：
        談話室（Ｎｏ），
        日（Ｗ）
－＞
        ａｓｓｕｍｅ（空き談話室（Ｎｏ，Ｗ））．

談話室（２２９）．
```

図４　　ルール化した例（追加ルール・セット）

この例よりわかるとおり、会議開催談話室なるルールにおいて２入力・ノード［３８４］が共有され、パターン・マッチングの手間およびＬａｂｅｌ計算の手間が省かれる。

４．３　ルールの削除

次に、ルールを削除した場合について示す。

例えば、談話室で会議を開催した結果良くなかったとする。すると、この会議開催計画のルール・セットから［談話室（Ａ）］または［空き談話室（Ａ，Ｂ）］の条件要素を含むルールを削除しておくほうが推論効率の面から好ましい。本方法によれば、図３に示すもとのネットワーク構造にもどる。したがって、不要になったノードが残らない。しかも、これらの条件要素を削除した効果は残り、これらの条件要素を含むルールを追加しても以後ネットワークには反映されない。すなわち、追加した知識だけでなく、削除した知識も、いわば学習効果としてシステムに残る。例えば、同様なルールなどは、誤ってもネットワークには反映されない。これは、談話室では会議を開催してはいけないという知識が追加されているとも考えられる。

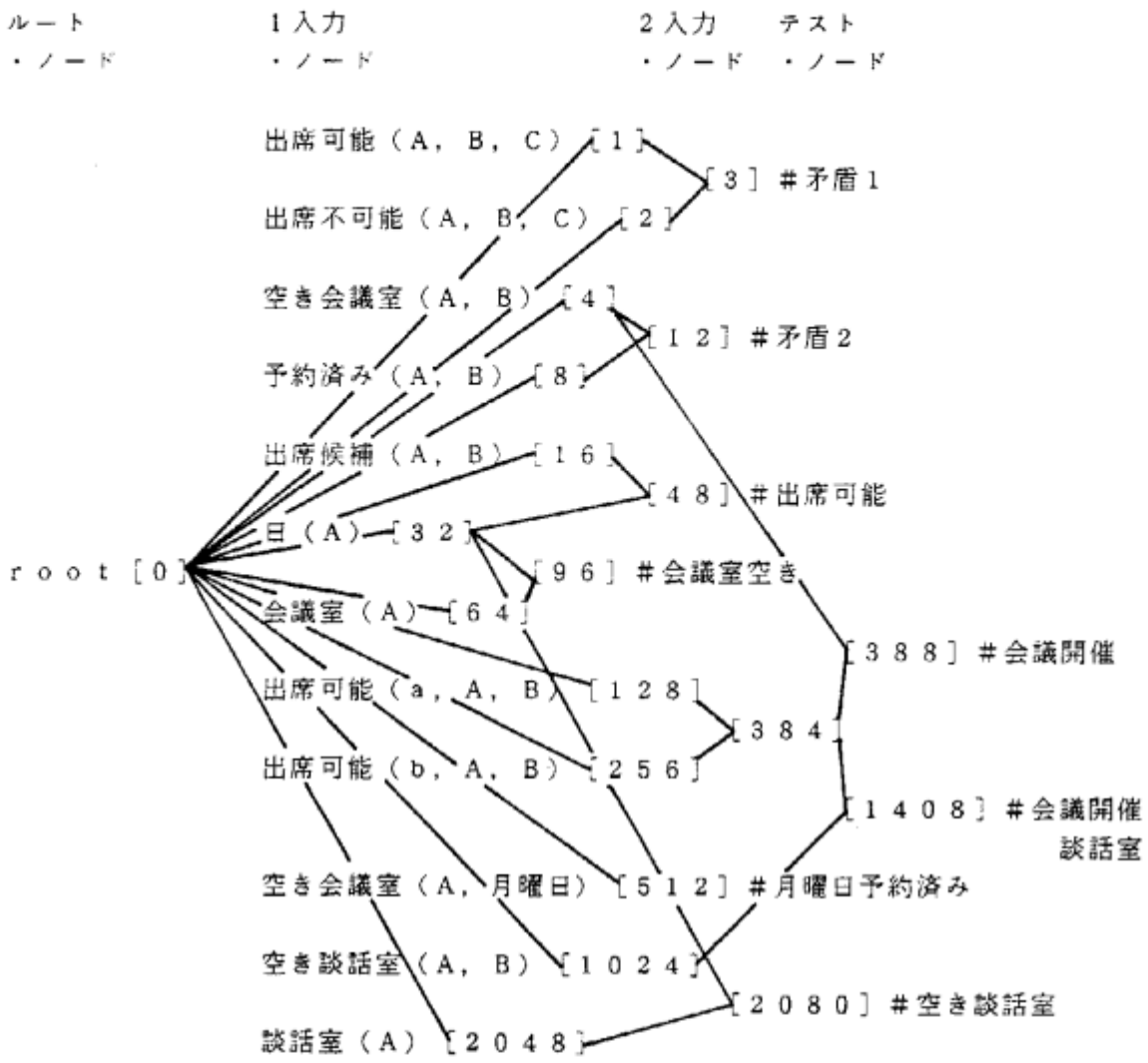

図5　　ネットワークの例
（図4のルール・セットを追加した場合、A，B，Cは、変数を意味する。
［］は、ビット・ベクタ十進表現した環境を示す。）

5．まとめ

以上、ATMSに着目したRete－likeネットワークのインクリメンタル構築法について述べた。本方法は、以下のような効果がある。

（1）ATMSとRete－likeネットワークを構築するモジュールの共有により仮説推論システムのインプリメントにあたって、Rete－likeネットワークを構築するモジュールの生産性および信頼性が向上するという効果がある。さらに、全体のプログラム・サイズが小さくなる。

（2）インクリメンタルなルールの追加、削除が可能である。追加については、共有すべきノードが共有される。また、削除については、ルール単位で行うのではなくルールの条件要素またはそれらの組み合せを指定してやる。それによって、それを条件に含むルールが全て削除される。このとき、不要になったノードが残らないという効果がある。しかも、これらの条件要素を削除した効果は残り、これらの条件要素を含むルールを追加しても以後ネットワークには反映されないという、いわば削除ルールに関する学習効果がある。

この点に関して、ルール・ベース・メンテナンスとルール・コンパイルを同時に行う手法であると考える。

尚、本方法による２入力・ノードは、２要素間の値のチェックは行っていない。中間ノードを生成し中間Ｌａｂｅｌを計算し、Ｌａｂｅｌ再計算の手間を省くことだけに利用している。この点については、Ｒｅｔｅアルゴリズムに比較して、パターン・マッチングの手間などが無駄になることもある。しかしながら、本システムでは、手続きはルールの条件部に対応するＡＴＭＳノードがＩＮになるまで全てのテスト実行が遅延される方式を採用している。これは、発火したルールの実行部をスケジューリングする（キューイングされた実行部の列をその環境のビット・ベクタが小さい順に並べ変える）ことにより、チェックを省略できることもあるという効果［ｄｅ　Ｋｌｅｅｒ　８６ｂ］を狙ったものである。

謝辞

本研究の機会を与えて下さり常々御指導頂いているＩＣＯＴ淵所長、第五研究室藤井室長に深く感謝致します。

参考文献

［ｄｅ　Ｋｌｅｅｒ　８６ａ］ｄｅ　Ｋｌｅｅｒ，　Ｊ．：”Ａｎ　Ａｓｓｕｍｐｔｉｏｎ－ｂａｓｅｄ　ＴＭＳ”，Ａｒｔｉｆｉｃｉａｌ　Ｉｎｔｅｌｌｉｇｅｎｃｅ　２８，　ｐｐ．１２７－１６２（１９８６）

［藤原　他　８８］藤原　遠，飯島　勝美，井上　克巳：”ＥＳＰによるＡＴＭＳと問題解決器を融合した仮説推論機構　－ＡＳＴＲＯＮ－”，情報処理学会第３７回全国大会講演論文集（ＩＩ）ｐｐ．１４４７－１４４８（１９８８）

［Ｉｎｏｕｅ　８８］Ｉｎｏｕｅ，　Ｋ．：”Ｐｒｏｂｌｅｍ　Ｓｏｌｖｉｎｇ　ｗｉｔｈ　Ｈｙｐｏｔｈｅｔｉｃａｌ　Ｒｅａｓｏｎｉｎｇ”，　Ｐｒｏｃ．　ＦＧＣＳ－８８：　Ｉｎｔｅｒｎａｔｉｏｎａｌ　Ｃｏｎｆｅｒｅｎｃｅ　ｏｎ　Ｆｉｆｔｈ　Ｇｅｎｅｒａｔｉｏｎ　Ｃｏｍｐｕｔｅｒ　Ｓｙｓｔｅｍｓ　（１９８８）

［ｄｅ　Ｋｌｅｅｒ　８６ｂ］ｄｅ　Ｋｌｅｅｒ，　Ｊ．：”Ｐｒｏｂｌｅｍ　Ｓｏｌｖｉｎｇ　ｗｉｔｈ　ｔｈｅ　ＡＴＭＳ”，Ａｒｔｉｆｉｃｉａｌ　Ｉｎｔｅｌｌｉｇｅｎｃｅ　２８，　ｐｐ．１９７－２２４（１９８６）

［飯島　他　８７］飯島泰裕、成田良一、吉田裕之、泉寛幸：”仮説ネットワークを用いた仮説推論器”、人工知能学会研究会研究会資料ＳＩＧ－ＦＡＩ－８７０１－２，ｐｐ．７－１４（１９８７）

［飛鳥井　他　８８］飛鳥井正道、森沢秀一、村田真人、浅野俊昭：”エキスパートシステム構築ツールＣＨＯＲＵＳ（２）―仮説推論機能―”、情報処理学会第３７回全国大会講演論文集（ＩＩ），ｐｐ．１２１８－１２１９（１９８８）

［Ｆｏｒｇｙ　８２］Ｃ．Ｌ．Ｆｏｒｇｙ：”Ｒｅｔｅ：Ａ　ｆａｓｔ　ａｌｇｏｒｉｔｈｍ　ｆｏｒ　ｍａｎｙ　ｐａｔｔｅｒｎ／ｍａｎｙ　ｏｂｊｅｃｔ　ｐａｔｔｅｒｎ　ｍａｔｃｈ　ｐｒｏｂｌｅｍ”，Ａｒｔｉｆｉｃｉａｌ　Ｉｎｔｅｌｌｉｇｅｎｃｅ１９ｐｐ．１７－３７（１９８２）

［荒屋　他　８８］荒屋真二、百原武敏、田町常夫：”知識ベース逐次構築法―Ｒｅｔｅネットワークの逐次構築法―”、信学論Ｄ　ＶｏｌＪ７１－Ｄ　Ｎｏ．６　ｐｐ．１１００－１１０８　（１９８８）