ICOT Technical Memorandum: TM-0676

TM-0676

仮説推論システムAPRICOT/O
ユーザズ・マニュアル

太田好彦、井上克己

January, 1989

©1989, ICOT

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

1．はじめに

仮説推論システムＡＰＲＩＣＯＴ／０は、ＡＴＭＳ（［ｄｅ　Ｋｌｅｅｒ　８６－１］［飯島　他　８８］）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　８８－１］［藤原　他　８８－２］）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。ここに、ＰＳＩ－ＩＩ、ＳＩＭＰＯＳ（［ＩＣＯＴ　８８－２］［ＩＣＯＴ　８８－３］）、ＥＳＰ［ＩＣＯＴ　８８　１］上のＡＰＲＩＣＯＴ／０を開発したのでその特徴と仕様、方式、使用例について報告する。

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　８６－２］　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、制約の集合の解析を行いルール形式で表現して扱う。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。

ＡＰＲＩＣＯＴ／０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。仮説推論システムは、不完全な知識（常に成立するとは限らない知識）等を取り扱うものであり、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［飛烏井　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［飛烏井　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な構築法については、「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。

ＡＰＲＩＣＯＴ／０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。このインクリメンタル構築法は、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、［飯烏　他　８７］や［飛烏井　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。

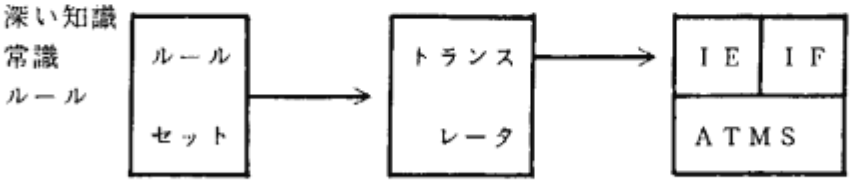

図１　構成図

ユーザが、直接編集したルール・セットを対象に、トランスレータが、Ｒｅｔｅ－ｌｉｋｅネットワークを構築する。

このトランスレータが、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。また、推論結果の表示等の機能をもったユーザ・インタフェースＩＦがある。

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（複合項）・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組込み述語、メソッド・コールのいずれかである。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．各論

以下、ＡＴＭＳ、トランスレータ、ＩＥおよびＩＦの方式・仕様・使用例について述べる。最初に、ＡＰＲＩＣＯＴ／０を使用する前には、ルール・セット・シンタックス、システム・メッセージとメニュー・ウィンドゥについて理解されれば良いものと考える。カスタマイズする時には、他の詳細部分について理解が必要になってくる。

＜凡例（以下、同様）＞

- ＃ｘｘｘ：クラス・オブジェクトｘｘｘ
- ＄ｘｘｘ：インスタンス・オブジェクトｘｘｘ
- ＃ｘｘｘ！ｙｙｙ：クラス・オブジェクトｘｘｘのクラス・スロットｙｙｙ
- ＄ｘｘｘ！ｙｙｙ：インスタンス・オブジェクトｘｘｘのインスタンス・スロットｙｙｙ

３．１　ＡＴＭＳ

３．１．１　方式

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。

ｄａｔｕｍ：データ（＄ｆｒｏｚｅｎ＿ｔｅｒｍ）
ｌａｂｅｌ：環境の集合（環境へのポインタの集合，＄ｌｉｓｔ）
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，＄ｌｉｓｔ）
ｃｏｎｓｅｑｕｅｎｔｓ：ｊｕｓｔｉｆｉｃａｔｉｏｎｓの逆ポインタ
　　（ノードへのポインタの集合，＄ｌｉｓｔ）
ｃｏｎｔｒａｄｉｃｔｏｒｙ：０＝ＩＮ，１＝ＯＵＴ

図２　　ノード（＄ｎｏｄｅ）のデータ構造

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（＃ｍａｔｍｓによる）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。図２に、ノード（＄ｎｏｄｅ）のデータ構造を示す。

また、本ＡＴＭＳでは、［飯島　他　８８］で採用されていなかった仮説のビット・ベクタ表現［ｄｅ　Ｋｌｅｅｒ　８６－１］を採用しラベル計算を高速化している。

いま、仮説空間がｎ個（必要に応じてシステムが動的に変えるようになっている。）の仮説ｐ１，ｐ２，・・・，ｐｎを持つとき、このｎ個の仮説をｎ個のビット列ｂ１，ｂ２，・・・，ｂｎと対応させる。各ビットは、対応する仮説が有るとき１、無いとき０の値をとる。すると、０個以上の仮説を組み合わせた環境Ｅもまたビット・ベクタで表現できる。このように表現しておくと、図３のように、環境の組み合わせおよび包含関係のチェックが論理演算で高速化できる。

環境Ｅ１と環境Ｅ２との組み合わせによる環境Ｅ３
　　ｏｒ（Ｅ１，Ｅ２，Ｅ３）
環境Ｅ１が環境Ｅ２のｓｕｂｓｅｔである。
　　ａｎｄ（Ｅ１，Ｅ２，Ｅ１）

図３　　論理演算による環境の組み合わせ
　　および包含関係のチェック

環境（Ｓｅｎｖ）は、図４のデータ構造である。本システムでは、原理的に無限長のビット・ベクタが表現できる。つまり、正の整数を要素としたリストで外部表現している。順序は、右側の方が上位となっており、［下位３１ビット十進整数<—>上位３１ビット十進整数］である。内部表現は、外部表現と同じ順序で、＄ｌｉｓｔとしている。格納場所は、＄ｅｎｖ！ａｓｓｕｍｐｔｉｏｎｓ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。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（＄ａｔｍｓ！ｅｎｖｉｒｏｎｍｅｎｔ＿ｔａｂｌｅは、ビット・ベクタの下３１ビットをインデックスとする＄ｍｏｎｏｇｙｎｙ＿ｈａｓｈ＿ｉｎｄｅｘである。その要素は、＄ｌｉｓｔであり、ビット・ベクタの集合となっている。また、その＄ｌｉｓｔの要素も＄ｌｉｓｔとなっており、ビット・ベクタを保持している。）はハッシュ・テーブルとなっている。

ａｓｓｕｍｐｔｉｏｎｓ：ビット・ベクタ（＄ｌｉｓｔ）
ｎｏｄｅｓ：ｌａｂｅｌの逆ポインタ
　　（ノードへのポインタの集合，＄ｌｉｓｔ）

図４　　環境（Ｓｅｎｖ）のデータ構造

他に、ｎｏｇｏｏｄ　ｄａｔａｂａｓｅ（＄ａｔｍｓ！ｎｏｇｏｏｄ＿ｄａｔａｂａｓｅは、＄ｌｉｓｔであり、ビット・ベクタの集合となっている。また、その＄ｌｉｓｔの要素も＄ｌｉｓｔとなっており、ビット・ベクタ各要素を保持している。）があり、これは矛盾する環境を極小（ｍｉｎｉｍａｌ）で管理している。

３．１．２　仕様

（１）ＡＴＭＳメソッド

＜凡例（以下、同様）＞

・＋：入力

・－：出力

・特記しない限りウィンドゥは、＄ｐｍａｃｓ＿ｗｉｎｄｏｗである。

・特記しない限りメソッドは、ユーザのミス（入・出力誤りやウィンドゥ生成失敗等）以外、ｓｕｃｃｅｓｓで返る。

:create(+#atms,-$atms)

－＄ａｔｍｓ（ＡＴＭＳ）を生成する。

:make_fact(+$atms,+Datum,-$node)

事実ノード－Ｓｎｏｄｅを生成する。ここに、－＄ｎｏｄｅ！ｌａｂｅｌ＝［０］である。

:make_assumption(+$atms,+Datum,-$node)

仮説ノード－＄ｎｏｄｅを生成する。＋Ｄａｔｕｍは、アトム、述語である。述語の場合、変数を含まないものでなければならない。

:make_justification(+$atms,+Antecedents,+Datum,-$node)

＋Ａｎｔｅｃｅｄｅｎｔｓを理由付けに持つ導出ノード－＄ｎｏｄｅを生成する。＋Ａｎｔｅｃｅｄｅｎｔｓは、［＋＄ｎｏｄｅ，・・・・］である。－＄ｎｏｄｅ！ｌａｂｅｌが変化しない（矛盾な理由付けをした場合等）とｆａｉｌする。

:add_justification(+$atms,+Antecedents,+$node)

＋＄ｎｏｄｅに＋Ａｎｔｅｃｅｄｅｎｔｓの理由付けを送る。＋＄ｎｏｄｅ！ｌａｂｅｌが変化しないとｆａｉｌする。

:add_contradiction(+$atms,+Antecedents)

＋Ａｎｔｅｃｅｄｅｎｔｓを矛盾とする。（矛盾ノードに＋Ａｎｔｅｃｅｄｅｎｔｓの理由付けを送る。）矛盾ノードを＄ｎｏｄｅとして、＄ｎｏｄｅ！ｌａｂｅｌが変化しないとｆａｉｌする。

:solve1(+$node,+Nij,+$atms)

＋Ｎｉｊは、仮説ノードのオールタナティブのリストのリストである。オールタナティブのリスト中の仮説ノードを１つずつ選び無矛盾な仮説組み合わせ（縦型探索）をウィンドゥに表示する。表示後、”Ｃ／Ｒ”を入力すると、ウィンドゥは消える。＋＄ｎｏｄｅ！ｗｏｒｋ（ｌａｂｅｌと同様のデータ構造）に無矛盾な環境が保持されている。＋Ｎｉｊは、［［＋＄ｎｏｄｅ１１，＋＄ｎｏｄｅ１２・・・］，［＋＄ｎｏｄｅ２１，＋＄ｎｏｄｅ２２・・・］］である。

:solve2(+$node,+Nij,+$atms)

＋Ｎｉｊは、仮説ノードのオールタナティブのリストのリストである。オールタナティブのリスト中の仮説ノードを１つずつ選び無矛盾な仮説組み合わせ（横型探索）をウィンドゥに表示する。表示後、”Ｃ／Ｒ”を入力すると、ウィンドゥは消える。＋＄ｎｏｄｅ！ｗｏｒｋ（ｌａｂｅｌと同様のデータ構造）に無矛盾な環境が保持されている。

（２）ＩＦメソッド

:create(+#interface,-$interface)

－＄ｉｎｔｅｒｆａｃｅを生成し、このとき、ウィンドゥが生成される。

:kill(+$interface)

ウィンドゥが消滅する。以後、＋＄ｉｎｔｅｒｆａｃｅは使用できない。

:et(+$interface,+$atms)

無矛盾な環境とそれをラベルにもつノードのｄａｔｕｍ、ｌａｂｅｌ、ｊｕｓｔｉｆｉｃａｔｉｏｎｓの内容をウィンドゥに表示する。表示フォーマットは、図５に示すとおりである。

＊＊＊　Ｃｏｎｓｉｓｔｅｎｔ　Ｅｎｖｉｒｏｎｍｅｎｔ　＊＊＊　％タイトル
＊＊＊　Ｅｎｖｉｒｏｎｍｅｎｔ：［１］
　　％無矛盾な環境［１］のタイトルである。リストの要素は、ビット・ベクタ十進表現した整数である。この環境をラベルに含むノードが、以下に表示される。
＜Ｎｏｄｅ＞：ａ　％ｄａｔｕｍをｍｅｌｔして表示している。
＜Ｌａｂｅｌ＞：［１］　％ラベルである。
＜Ｊｕｓｔｉｆｉｃａｔｉｏｎｓ＞
［［１］］
　　％ｊｕｓｔｉｆｉｃａｔｉｏｎｓである。本ノードは、自身を理由付けに持つ仮説ノードである。
　　・
　　・
　　・
＊＊＊　Ｅｎｖｉｒｏｎｍｅｎｔ：［１２］
＜Ｎｏｄｅ＞：ｅ
＜Ｌａｂｅｌ＞：［３］［１２］
　　％ｏｒ関係で複数ある場合は、このように環境が（横に）並ぶ。
＜Ｊｕｓｔｉｆｉｃａｔｉｏｎｓ＞
［ａｂ］
　　％リストの要素は、ａｎｔｅｃｅｄｅｎｔｓの$ｎｏｄｅのｄａｔｕｍ（ｍｅｌｔして表示している）となる。リストの内部は、ａｎｄ関係である。
［ｃｄ］
　　％ｏｒ関係で複数のｊｕｓｔｉｆｉｃａｔｉｏｎｓがある場合、このように（縦に）並ぶ。
　　・
　　・
　　・
　　図５　　ｅｔの表示フォーマット

:ng(+$interface,+$atms)
　　矛盾な環境を極小（ｍｉｎｉｍａｌ）でウィンドゥに表示する。すなわち、ｎｏｇｏｏｄ　ｄａｔａｂａｓｅの内容である。表示フォーマットは、図６のとおりである。

＊＊＊　Ｉｎｃｏｎｓｉｓｔｅｎｔ　Ｅｎｖｉｒｏｎｍｅｎｔ　＊＊＊　％タイトル
［５］　％矛盾な環境。
［２４］
［３４］
［１３６］
［１４４］
　　図６　　ｎｇの表示フォーマット

:put_node(+$interface,+Nodes)

＋Ｎｏｄｅｓで示されるノードのｄａｔｕｍ、ｌａｂｅｌ、ｊｕｓｔｉｆｉｃａｔｉｏｎｓの内容をウィンドゥに表示する。表示フォーマットは、メソッドｅｔのフォーマットのうち、タイトルが表示されないだけで、ほとんど同じである。＋Ｎｏｄｅｓ＝［＋＄ｎｏｄｅ１、＋＄ｎｏｄｅ２・・・・］　である。

:getl(+$interface,-Variable)

－Ｖａｒｉａｂｌｅをユニファイ可能モードの変数とし、ウィンドゥより行を入力する。例えば、ウィンドゥをスクロールさせたい場合は、このメソッドを用いる。また、ウィンドゥより”Ｃ／Ｒ”を入力するするとこのメソッドは終了する。

３．１．３　使用例

本ＡＴＭＳを用いて問題を解く例を示す。

［問題１］

ａ，ｂまたはｃ，ｄならばｅ、ａを事実とし、ｂ，ｃ，ｄを仮説としたとき、無矛盾な環境とそれをラベルにもつノードの状態を示せ。さらに、ｃ，ｄは矛盾として、全てのノードの状態を示せ。

［プログラムの例１］

```
class   example1   has
attribute
    wm  is list;
:goal(Class):-
        :create(#atms,P),
        :create(#interface,I),
    :make_fact(P,a,A),
            :add_last(Class!wm,A),
    :make_assumption(P,b,B),
            :add_last(Class!wm,B),
    :make_assumption(P,c,C),
            :add_last(Class!wm,C),
    :make_assumption(P,d,D),
            :add_last(Class!wm,D),
    :make_justification(P,[A,B],e,E),
    :add_justification(P,[C,D],E),
            :add_last(Class!wm,E),
        :et(I,P),
        :getl(I,_),
    :add_contradiction(P,[C,D]),
            :get_contents(Class!wm,Nodes),
        :put_node(I,Nodes),
        :getl(I,_),
        :kill(I);
end.
```

［問題２］
男性（ａ，ｂ，ｃ，ｄ）および女性（ｅ，ｆ，ｇ，ｈ）から男性・女性のペアを全員について作りたい。このとき以下に示す仲が良いペアを考慮する。何通りの作り方があるか。
＜仲が良いペア＞：（ａ，ｅ），（ａ，ｇ），（ｂ，ｅ），（ｂ，ｈ），（ｃ，ｈ），（ｃ，ｇ），（ｄ，ｆ），（ｄ，ｈ）

［プログラムの例２］

```
class
        testp
has
nature
    node
    ;

 :goal(Class,Node,S) :-
    :new(Class,Node),
    :create(#atms,S),
    :make_assumption(S,pair(a,e),AE),
    :make_assumption(S,pair(a,g),AG),
    :make_assumption(S,pair(b,e),BE),
    :make_assumption(S,pair(b,h),BH),
    :make_assumption(S,pair(c,h),CH),
    :make_assumption(S,pair(c,g),CG),
    :make_assumption(S,pair(d,f),DF),
    :make_assumption(S,pair(d,h),DH),
    :add_contradiction(S,[AE,BE]),
    :add_contradiction(S,[BH,CH]),
    :add_contradiction(S,[AG,CG]),
    :add_contradiction(S,[BH,DH]),
    :add_contradiction(S,[CH,DH]),
    :solve1(Node,[[AE,AG],[BE,BH],[CG,CH],[DF,DH]],S)
    ;
end.
```

```
[pair(a,e),pair(b,h),pair(c,g),pair(d,f)]
[pair(a,g),pair(b,e),pair(c,h),pair(d,f)]
```
で２通り。

３．２　トランスレータ

３．２．１　方式

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ネットワークに類似している。この点に着目し、トランスレータをＡＴＭＳモジュールを利用

して効率的にインプリメントした。

以下、上記ＡＴＭＳモジュールを利用したルール・セットからＲｅｔｅ－ｌｉｋｅネットワークへ変換するトランスレータ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図７に示す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（＄ｌｉｓｔ）
ｓｕｂ：下位環境へのポインタの集合（＄ｌｉｓｔ）
ｗｍ：分散ワーキング・メモリ（分散ＷＭ），
　　ＡＴＭＳノードの集合（＄ｌｉｓｔ）
図７　　環境の追加スロット

［ｄｅ　Ｋｌｅｅｒ　８６－１］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に図８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。
図８　　ｏｒ（Ｅ１，Ｅ２，Ｅ３）追加操作

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ノードとして事実ノードを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とした仮説ノードを生成する。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。

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とした仮説ノードを生成する。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、新しい仮説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。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、仮説としてのこれらのノードまたはそれらの組み合わせのラベル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。

3．2．2　仕様

（1）ルール・セット・シンタックス

＜凡例（以下、同様）＞

・｛｝：：0回以上の繰り返し。

・ルール・セット・ファイルは、userディレクトリ直下のファイルでファイルタイプruleとする。

・［ICOT　88－1］を参照のこと。

＜ルール・セット＞：：＝＜ルール＞｛＜ルール＞｝．

＜ルール＞：：＝

＜事実＞”．”｜＜静的仮説＞”．”｜

＜ルールID＞”：：”＜条件部＞”－＞”＜実行部＞”．”．

＜事実＞：：＝＜アトム＞｜＜静的複合項＞．

・＜静的複合項＞は＜ストリング＞又は＜変数＞を含まない＜複合項＞である。

＜静的仮説＞：：＝”ａｓｓｕｍｅ（”＜事実＞”）”．

・静的な仮説の生成を意味する。

＜ルールＩＤ＞：：＝＜アトム＞．

・＜ルールＩＤ＞は、大域的意味を持つ。

＜条件部＞：：＜条件部要素＞｛”，”＜条件部要素＞｝．

＜条件部要素＞：：＝＜条件要素＞｜＜テスト＞．

＜条件要素＞：：＝＜アトム＞｜＜変数＞｜＜複合項＞　．

・＜複合項＞は、＜ストリング＞を許さない。（＜複合項＞中の）＜変数＞は、＜テスト＞または＜実行部＞実行の前にユニファイし、＜事実＞とならなければならない。

＜テスト＞：：＝”｛”＜手続き＞｛”，”＜手続き＞｝”｝”．

＜手続き＞：：＝＜組込み述語＞｜＜メソッド・コール＞．

・組込み述語は、アトム（！，ｃｕｔ　ａｎｄ＿ｆａｉｌ，ｆａｉｌ，ｔｒｕｅ）を除く。

＜実行部＞：：＝”［］”｜＜条件要素＞｜＜動的仮説＞．

・［］　は、矛盾を意味する。ここでの＜条件要素＞は、導出ノードの生成または理由付けの生成を意味する。

＜動的仮説＞：：＝”ａｓｓｕｍｅ（”＜条件要素＞”）”．

・動的な仮説の生成を意味する。

（２）トランスレータ・メソッド

:create(+#trans,-$trans)

－＄ｔｒａｎｓ（トランスレータ）の生成。

:goal(+$trans,+Rs)

＋Ｒｓをトランスレートしたいルール・セット・ファイルのｉｄｅｎｔｉｆｉｅｒとしトランスレータの起動（ルールの追加）を行う。

:add_contradiction(+$trans!net,+Conditions)

＋Ｃｏｎｄｉｔｉｏｎｓを含むルールを全て削除する。Ｃｏｎｄｉｔｉｏｎｓ－［＜条件要素＞｛”，”＜条件要素＞｝］。ここで、＜条件要素＞は、インスタンス・オブジェクトではなく、ｄａｔｕｍのレベルである。＋＄ｔｒａｎｓの矛盾ノードのラベルが変化しないとｆａｉｌする。

３．２．３　使用例

以下、本ルール・トランスレート方法の効果を簡単な会議開催計画問題の例で示す。

３．２．３．１　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９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１０に示す。

```
矛盾１：：
        出席可能（Ｘ，Ｐ，Ｗ），
        出席不可能（Ｘ，Ｐ，Ｗ）
  ＞
        ［］．

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））．

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９　ルール化した例

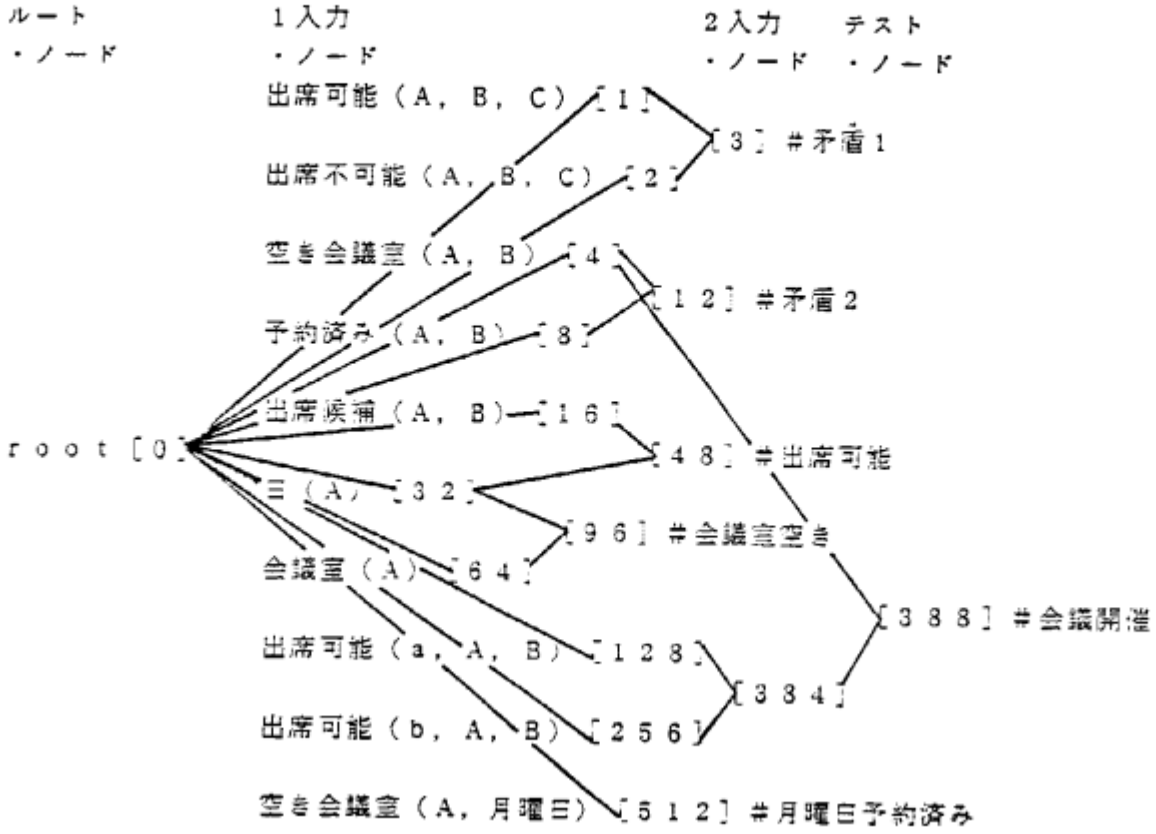

図１０　　ネットワークの例
（図９のルール・セット．A，B，Cは、変数を意味する。
〔〕は、ビット・ベクタ十進表現した環境を示す。）

本方法は、ＡＴＭＳとＲｅｔｅアルゴリズムとの融合により［飯島　他　８７］と同様なラベル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ラベル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ラベル計算し、分散ＷＭに保持しているのでラベル再計算をする必要がなく効率的である。

３．２．３．２　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１１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１２に示す。

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ラベル計算の手間が省かれる。

```
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１１　ルール化した例（追加ルール・セット）

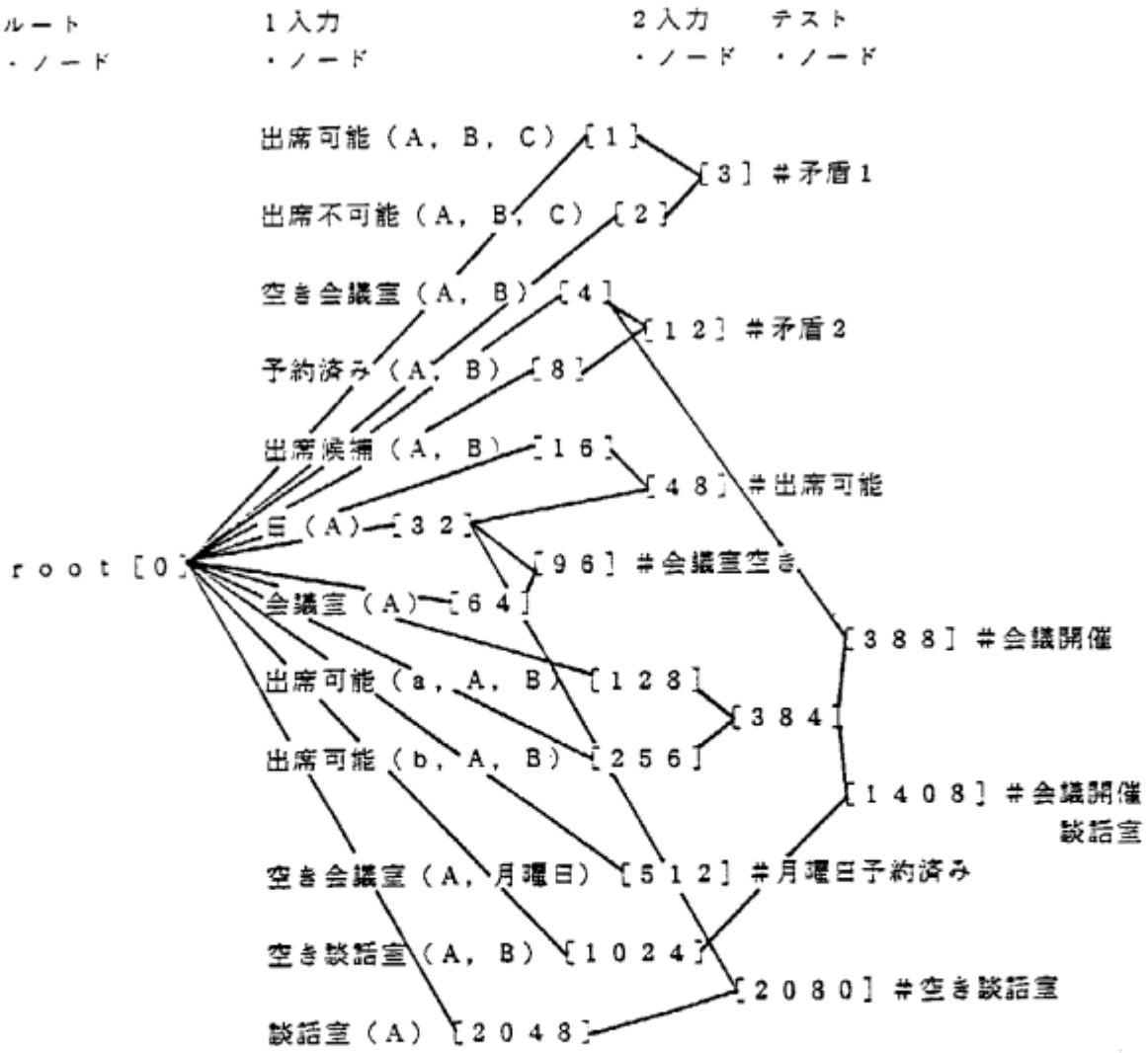

図１２　　ネットワークの例
（図１１のルール・セットを追加した場合、Ａ，Ｂ，Ｃは、変数を意味する。
［］は、ビット・ベクタ十進表現した環境を示す。）

３．２．３．３　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１０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。

３．３　ＩＥ

３．３．１　方式

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。まずこれらのＡＴＭＳノード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ｓｕｐｅｒ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。その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入力ノードのＷＭにそのトークンを保持（ａｄｄ＿ｌａｓｔ）する。次に、その１入力ノードとｓｕｐｅｒリンクで結合されている２入力ノードにそのトークンを流す。その２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（Ｄａｔｕｍ＿ｌｅｆｔ＾Ｄａｔｕｍ＿ｒｉｇｈという形の中間ノードが生成される。ここに、ｌｅｆｔは、その２入力ノードのｓｕｂスロット$ｌｉｓｔの先の要素、ｒｉｇｈは後の要素としている。）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２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（縦方向に）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、テストが成功すれば、新しいデータの生成、新しい理由付けの生成、矛盾の生成が行われる。新しいデータの生成の場合、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は、即座にＡＴＭＳにそのタスクを送る。次に、残っているＷＭＥ、残っているｓｕｂリンクで結合されている１入力ノード、残っているｓｕｐｅｒリンクで結合されている２入力ノード（横方向に）、および残っているｓｕｐｅｒリンクで結合されている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。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。つまり、推論エンジンは縦型探索をおこなう。

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ラベルを計算し、ラベル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　８６－２］を狙ったものである。

３．３．２　仕様

（１）ＩＥメソッド

:create(+#ie,-$ie,+$trans!net)

ルール・セットをトランスレートした＋＄ｔｒａｎｓを用いて推論を行う推論エンジン－＄ｉｅを生成する。これにより、生成した推論エンジン－＄ｉｅのトークンの待ち行列は、ＦＩＦＯのキューであり、ｎｏ＿ｓｃｈｅｄｕｌｅ　ｍｏｄｅである。

:create_scheduler(+#ie,-$ie,+$trans!net)

ルール・セットをトランスレートした＋＄ｔｒａｎｓを用いて推論を行う推論エンジン－＄ｉｅを生成する。これにより、生成した推論エンジン－＄ｉｅのトークンの待ち行列は、前記スケジューリング・キューであり、ｓｃｈｅｄｕｌｅ　ｍｏｄｅである。

:goal(+$ie)

推論エンジン＋＄ｉｃの推論を開始する。

（２）マンマシン・インタフェース

+$ie!atms!atms=-ATMS

推論エンジン＋＄ｉｅの推論が終了した後、データの状態を調査するためには、この－ＡＴＭＳを先に述べたインタフェースを用いて或いは、データ構造を直接アクセスして行う。

+#apricot0!debug=+Debug_mode

＋Ｄｅｂｕｇ＿ｍｏｄｅは、０か１である。＋＃ａｐｒｉｃｏｔ０！ｄｅｂｕｇ：＝１とすると、デバッグ・モードとなり実行トレースが、ウィンドゥに表示される。元に戻すには、＋＃ａｐｒｉｃｏｔ０！ｄｅｂｕｇ：＝０とする。

（３）システム・メッセージとメニュー・ウィンドゥ

＜凡例＞

以下の行頭、項頭の記号は、以下の規則に従う。

・”＊＊”：メニュー・ラベル・タイトル
・”；”：上記または左記の項目の説明
・”・”：メニュー項目
・”！－”：システム・メッセージ
・”？－”：システム・入力要求
・システム・メニューのＡＰＲＩＣＯＴ／０を選択すると、トップ・メニューとウィンドゥが表示される。トップ・メニューのサブ・メニューに関する説明は、段を付けて示す。

！－Ｅｎｄ；正常終了
！－Ａｂｏｒｔ；異常終了
＊＊ＡＰＲＩＣＯＴ／０；＄ｍｅｎｕ、トップ・メニュー
・ａｄｄ；ルール・セット追加

？－Ｉｎｐｕｔ　Ａｄｄｅｄ　Ｒｕｌｅ　Ｓｅｔ：
；追加ルール・セット・ファイルｉｄｅｎｔｉｆｉｅｒ入力要求、”＜ｉｄｅｎｔｉｆｉｅｒ＞．Ｃ／Ｒ”で入力する。
・ｒｅｍｏｖｅ；ルール削除
？－Ｉｎｐｕｔ　Ｒｅｍｏｖｅｄ　Ｅｌｅｍｅｎｔ　Ｌｉｓｔ：
；削除したいルールの条件要素またはそれらの組み合せをリストの要素として指定する。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。”＜リスト＞．Ｃ／Ｒ”で入力する。
・ｒｕｎ；推論実行開始
　　＊＊ＳＴＲＡＴＥＧＹ；＄ｔｅｍｐｏｒａｒｙ＿ｍｅｎｕ、実行戦略メニュー
　　・ｎｏ　ｓｃｈｅｄｕｌｅ
　　；トークンの待ち行列は、ＦＩＦＯのキューであり、ｎｏ＿ｓｃｈｅｄｕｌｅ　ｍｏｄｅである。ａｂｏｒｔの場合このｍｏｄｅとなる。
　　・ｓｃｈｅｄｕｌｅ
　　；トークンの待ち行列は、前記スケジューリング・キューであり、ｓｃｈｅｄｕｌｅ　ｍｏｄｅである。
　　！－Ｒｕｎｎｉｎｇ；実行中
・ｂｒｏｗｓｅ；情報表示
　　＊＊ＢＲＯＷＳＥ
　　；＄ｔｅｍｐｏｒａｒｙ＿ｍｕｌｔｉｐｌｅ＿ｓｅｌｅｃｔ＿ｍｅｎｕ、情報表示メニュー
　　・ｎｅｔｗｏｒｋ；ネットワークの構造をテキスト表示する。例を図１３に示す。
　　・ｉｎ　ｄａｔｕｍ
　　；ｉｎ　ｄａｔｕｍをテキスト表示する。例を図１５に示す。
　　・ｏｕｔ　ｄａｔｕｍ
　　；ｏｕｔ　ｄａｔｕｍをテキスト表示する。例を図１５に示す。
　　　＊＊ＷＨＩＣＨ
　　　；＄ｔｅｍｐｏｒａｒｙ＿ｍｅｎｕ（ｉｎ　ｄａｔｕｍ、ｏｕｔ　ｄａｔｕｍを選択の場合）
　　　・ｐｔｔｅｒｎ；指定パターンのみ表示。
　　　？－Ｐａｔｔｅｒｎ：
　　　；指定パターン”＜条件要素＞．Ｃ／Ｒ”の形式で入力する。例を図１５に示す。
　　　・ａｌｌ　　；全パターン表示。
　　・ｅｎｖｉｒｏｎｍｅｎｔｓ
　　；無矛盾な環境とそれをラベルにもつノードのｄａｔｕｍ、ｌａｂｅｌ、ｊｕｓｔｉｆｉｃａｔｉｏｎｓの内容をウィンドゥに表示する。表示フォーマットは、メソッドｅｔと同様である。
　　・ＮＧ
　　；矛盾な環境を極小でウィンドゥに表示する。すなわち、ｎｏｇｏｏｄ　ｄａｔａｂａｓｅの内容である。表示フォーマットは、メソッドｎｇと同様である。
・ｓｃｒｏｌｌ；ウィンドゥをスクロールさせる。”Ｃ／Ｒ”を入力するとｓｃｒｏｌｌから抜ける。
・ｅｘｉｔ；ＡＰＲＩＣＯＴ／０を終了させる。

３．３．３　使用例

図９に示すルール・セット・ファイルをm．ｒｕｌｅとし、それをａｄｄ（トランスレート）した後、ＢＲＯＷＳＥ：ｎｅｔｗｏｒｋ機能でネットワークの構造をテキスト表示した状態を図１３に示す。

デバッグ・モードとし、トレースをとり、推論の一部を図１４に示す。

推論終了後、ＢＲＯＷＳＥ機能でｉｎ　ｄａｔｕｍとｏｕｔ　ｄａｔｕｍを表示する。この状態を図１５（ｐａｔｔｅｒｎ指定）に示す。

```
＊＊＊　Ｃｏｎｓｉｓｔｅｎｔ　Ｅｎｖｉｒｏｎｍｅｎｔ　＊＊＊
＊＊＊　Ｅｎｖｉｒｏｎｍｅｎｔ：［０］
＜Ｓｕｐｅｒ＞
    ％Ｅｎｖｉｒｏｎｍｅｎｔ：［０］のｓｕｐｅｒ　ｅｎｖｉｒｏｎｍｅｎｔを以下に示す。

［１］
  ・
  ・
［５１２］
＜Ｓｕｂ＞
    ％Ｅｎｖｉｒｏｎｍｅｎｔ：［０］のｓｕｂ　ｅｎｖｉｒｏｎｍｅｎｔを以下に示す。この場合は存在しない。

    ％Ｅｎｖｉｒｏｎｍｅｎｔ：［０］をラベルに持つノードを以下に示す。

＜Ｎｏｄｅ＞：ｒｏｏｔ
＜Ｌａｂｅｌ＞：［０］
＜Ｊｕｓｔｉｆｉｃａｔｉｏｎｓ＞
＊＊＊　Ｅｎｖｉｒｏｎｍｅｎｔ：［１］
＜Ｓｕｐｅｒ＞
［３］
＜Ｓｕｂ＞
［０］
＜Ｎｏｄｅ＞：出席可能（Ａ，Ｂ，Ｃ）
＜Ｌａｂｅｌ＞：［１］
＜Ｊｕｓｔｉｆｉｃａｔｉｏｎｓ＞
［［１］］
      ・
      ・
      ・
```

図１３　　ネットワークの構造（図９ルール・セット）

```
！－Ｒｕｎｎｉｎｇ
＊＊＊　Ｃｈｅｃｋ　Ｒｕｌｅ　：＃出席可能
＊　Ｔｏｋｅｎ　：
＜Ｎｏｄｅ＞：出席候補（ａ，本人）＾日（月曜日）
＜Ｌａｂｅｌ＞：［０］
＜Ｊｕｓｔｉｆｉｃａｔｉｏｎｓ＞
［日（月曜日）出席候補（ａ，本人）］
＊＊＊　Ｆｉｒｅｄ
```

％上記Ｔｏｋｅｎは、出席候補（ａ，本人）と日（月曜日）というノードをａｎｄ結合した中間ノードである。このように、２入力ノードによってノードのａｎｄ結合（＾で結合される）がとられ、中間ラベル計算が行われるのが本推論エンジンの特徴である。これは、そのＴｏｋｅｎが＃出席可能というテスト・ノードに流れてきてテストを実行した結果、テストが成功し発火（Ｆｉｒｅｄ）したことを示す。テストが失敗すると”＊＊＊　Ｆｉｒｅｄ”なるメッセージは表示されない。

```
＊＊＊　Ｃｈｅｃｋ　Ｒｕｌｅ　：＃月曜日予約済み
＊　Ｔｏｋｅｎ　：
＜Ｎｏｄｅ＞：空き会議室（２１２，月曜日）
＜Ｌａｂｅｌ＞：［１６］
＜Ｊｕｓｔｉｆｉｃａｔｉｏｎｓ＞
［［ｔｅｍｐｏｒａｒｙ，空き会議室（２１２，月曜日）］会議室（２１２）＾日（月曜日）］
＊＊＊　Ｆｉｒｅｄ
```

％空き会議室なるルールによって、空き会議室（２１２，月曜日）なる動的仮説［ｔｅｍｐｏｒａｒｙ，空き会議室（２１２，月曜日）］が生成されている。会議室（２１２）＾日（月曜日）は１個の中間ノードである。［［ｔｅｍｐｏｒａｒｙ，空き会議室（２１２，月曜日）］会議室（２１２）＾日（月曜日）］は、［ｔｅｍｐｏｒａｒｙ，空き会議室（２１２，月曜日）］と会議室（２１２）＾日（月曜日）のａｎｄ結合である。

```
！－Ｅｎｄ
```

図１４　推論トレース（図９のルール・セット）

```
？－Ｐａｔｔｅｒｎ：開催（＿，＿，＿，＿）．
＊＊＊　　ＩＮ　Ｐａｔｔｅｒｎ　　＊＊＊
＊開催（火曜日，２１２，ａ（代理人），ｂ（代理人））
＊開催（火曜日，２１２，ａ（本人），ｂ（代理人））
？－Ｐａｔｔｅｒｎ：開催（＿，＿，＿，＿）．
＊＊＊　　ＯＵＴ　Ｐａｔｔｅｒｎ　　＊＊＊
```

図１５　推論結果（ｐａｔｔｅｒｎ指定、図９のルール・セット）

４．使用条件

以下に示すファイルをＣａｔａｌｏｇｕｅし、以下に示すクラス・オブジェクトを生成し、生成した全クラス・オブジェクトをＳａｖｅする。

```
ana.esp"..#analyst;ap.esp"..#apricot0;atms.esp"..#atms;con.esp"..#consistent
env.esp"..#env;ie.esp"..#ie;in.esp"..#inconsistent;int.esp"..#interface
int_s.esp"..#rete_interface;matms.esp"..#matms;net.esp"..#rete_net
node.esp"..#node;print.esp"..#print;ratms.esp"..#rete_atms
renv.esp"..#rete_environme;union.esp"..#union;rete_print.esp"..#rete_print
trans.esp"..#trans;rin.esp"..#rete_inconsistent;sol.esp"..#solution
runion.esp"..#rete_union;sch.esp"..#scheduler
```

使用するｕｓｅｒディレクトリ直下のｌｏｇｉｎ．ｃｏｍファイルを以下のように修正する。ｌｏｇｏｕｔ後、再びそのｕｓｅｒディレクトリにｌｏｇｉｎし、ＳＹＳＴＥＭ ＭＥＮＵでＡＰＲＩＣＯＴ／０を選択する。

```
menu  :-
      items_list(
          [  [  {"debugger", debugger}  ,
                        .
                        .
                        .
                {"APRICOT/0", apricot0}]  ]    ).
```

同一パッケージ内に、上記ＡＰＲＩＣＯＴ／０システム・クラス名，ルール・セット名およびルールＩＤ名をライブラリ・レベルで複数存在させてはいけない。また、ＡＰＲＩＣＯＴ／０システム・クラス・オブジェクトの占有する領域は、約１３０［ＫＢｙｔｅｓ］であり、ソース・コード行は、約３［ＫＳｔｅｐｓ］である。

５．まとめ

以上、ＡＰＲＩＣＯＴ／０の仕様、方式、使用例について述べた。

特に、ＡＰＲＩＣＯＴ／０は、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が特徴的であり、以下のような効果がある。

・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・サイズを小さくすることができる。

・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。

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ならびに滝主任研究員に深く感謝致します。

参考文献

［藤原　他　８８－１］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）

［藤原　他　８８－２］藤原　遠，井上　克巳：”ＥＳＰによる仮説推論機構ＡＳＴＲＯＮ（ＥＳＰによるＡＴＭＳ　－第２版－）”，ＩＣＯＴ　Ｔｅｃｈｎｉｃａｌ　Ｍｅｍｏｒａｎｄｕｍ　Ｎｏ．　ＴＭ－５８７，　ＩＣＯＴ，（１９８８）

［飯島　他　８８］飯島　勝美，井上　克巳：”ＥＳＰによるＡＴＭＳ　－第１版－”，ＩＣＯＴ　Ｔｅｃｈｎｉｃａｌ　Ｍｅｍｏｒａｎｄｕｍ　Ｎｏ．　ＴＭ－４６７，ＩＣＯＴ，（１９８８）

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

［ｄｅ　Ｋｌｅｅｒ　８６－１］ｄｅ　Ｋｌｅｅｒ，　Ｊ．：”Ａｎ　Ａｓｓｕｍｐｔｉｏｎ－ｂａｓｅｄ　ＴＭＳ”，Ａｒｔｉｆｉｃｉａｌ　Ｉｎｔｅｌｌｉｇｅｎｃｅ　２８，　ｐｐ．１２７－１６２（１９８６）

［ｄｅ　Ｋｌｅｅｒ　８６－２］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（１１），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　Ｖｏｌ　Ｊ７１－Ｄ　Ｎｏ．６　ｐｐ．１１００　１１０８　（１９８８）

［ＩＣＯＴ　８８－１］新世代コンピュータ技術開発機構：”小型化ＰＳＩ　ＥＳＰ説明書”、ＩＣＯＴ（１９８８）

［ＩＣＯＴ　８８－２］新世代コンピュータ技術開発機構：”小型化ＰＳＩ／ＳＩＭＰＯＳプログラミング説明書（Ｉ）－基本編－（ＳＩＭＰＯＳ　４．０版）”、ＩＣＯＴ（１９８８）

［ＩＣＯＴ　８８－３］新世代コンピュータ技術開発機構：”小型化ＰＳＩ／ＳＩＭＰＯＳプログラミング説明書（ＩＩ）－入出力編－（ＳＩＭＰＯＳ　４．０版）”、ＩＣＯＴ（１９８８）