ICOT Technical Memorandum: TM-0587

TM-0587

ESPによる仮説推論機構
－ASTRON－

藤原遠(NTT)，井上克己

August, 1988

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

ICOT Technical Memorandum

TM－587

ＥＳＰによる仮説推論機構
－ＡＳＴＲＯＮ－

ＥＳＰによるＡＴＭＳ　－第２版－

A Hypothetical Reasoning System in ESP -ASTRON-

藤原　　達　　Toshi Fujiwara(*)

井上　克巳　　Katsumi Inoue

ＩＣＯＴ

(*)ＮＴＴデータ通信㈱

昭和 ６３年 ７月 ３１日

概要

仮説推論は従来の古典論理の枠組みだけでは扱えなかった不完全な知識を扱うための一方式である。仮説を基に推論を行うためには真理維持機構が必要である。その１方式としてＡＴＭＳが提唱されているがＡＴＭＳは、それ自身は問題解決を行うものではない。そのＡＴＭＳに、実際に問題領域に依存する問題解決機構を、いかに組み合わせて非単調な推論を行わせるかが、実際に仮説推論を応用していくにあたっての課題となる。本稿では、そのうち仮説の組合せに着目して、一貫性の維持を行っているＡＴＭＳと、ルール形式の記述を許した問題解決器をＥＳＰ上で実現した仮説推論システム「ＡＳＴＲＯＮ」について報告する。

目次

1．はじめに

仮説推論[井上編 88] は従来の古典論理の枠組みだけでは扱えなかった不完全な知識を扱うための一方式である。真偽が不明な知識はとりあえず真(仮説)として推論を進め後に、矛盾する状況が起こった時には、その矛盾の基になる仮説を修正するという、一種の非単調推論を実現する枠組みである。すなわち、問題解決の過程で競合する知識や、常に成立するとは限らない知識を取り扱う場合に必要となるものであり、それらの知識を仮説と見立ててそれに基づいた処理を行うことにより進めていく推論の形態をいう。

この仮説推論は実際に人間が行っている推論に近く、高次推論機能の多くを実現するための一つの鍵でもある。また、推論の結果得られた無矛盾な仮説の集合を解とすることにより、従来の診断や設計といった問題の解決に対し、プロダクション・システムとは異なるアプローチが考えられている。

仮説を基に推論を行うためには、作業記憶内を動的に管理するなんらかの真理維持機構が必要となる。この真理維持機構としては、ＴＭＳ[Doyle 79]やＡＴＭＳ[de Kleer 86a] などが提唱されている。本稿では、そのうち仮説の組合せに基づいて、一貫性の維持を行っているＡＴＭＳに着目し、これとルール・ベースの問題解決器をＥＳＰで実現した仮説推論システム「ＡＳＴＲＯＮ」[藤原 他 88]について報告する。

2．ＡＴＭＳと問題解決器

対象毎に最適な問題解決器を組み込めるようにするためには、知識管理機構(ＡＴＭＳ)と問題解決機構は明確に独立されるべきである。ＡＴＭＳは、多重コンテキストに基づき、知識の一貫性を管理する汎用の機構であり、それ自身は問題解決を行うものではない。そのＡＴＭＳに、実際に問題領域に依存する問題解決機構を、いかに組み合わせて非単調な推論を行わせるかが、仮説推論を応用していくにあたっての課題となる。

ＡＳＴＲＯＮではルールベースの問題解決器を考慮しており、図１に示すように、問題解決器が事実、仮説及びそれらに対する理由付けをＡＴＭＳに与えると、ＡＴＭＳが効率的に全てのコンテキストを決定する。

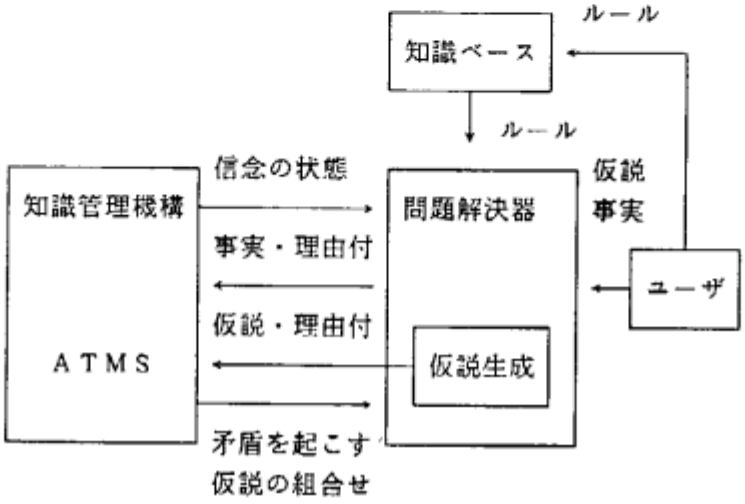


図１．　仮説推論システム ＡＳＴＲＯＮ の構成

問題解決器はあるコンテキストが矛盾を起こすかどうか、またあるデータがどのコンテキストに含まれるかをＡＴＭＳに確認しながら、推論を進めている。

3．仮説推論アルゴリズム

ＡＳＴＲＯＮの動作アルゴリズムをＡＴＭＳ・問題解決器にわけて説明する。

3.1 ＡＴＭＳ[飯島，井上 88]

ＡＴＭＳ[de Kleer 86a]は、仮説推論で扱う仮説、矛盾、理由付けを以下の環境、コンテキスト、ラベル等で管理している。

3.1.1 ＡＴＭＳの特徴

ＡＴＭＳの特徴をまとめると以下のようになる。

(1)仮説の組合せに基づき、多重コンテキストで同時に管理する。
(2)データは複数のコンテキストに含まれていてよい。
(3)各コンテキストが無矛盾であれば、全体は無矛盾である必要はない。
(4)コンテキストの参照が容易である。
(5)探索空間の別の点に容易に移動できる。
(6)探索戦略はＡＴＭＳの外にある。

3.1.2 構成要素とその役割

(1)ノード (node)：事実・仮説・導出データ等を表現するデータ構造。

(2)理由づけ (justification)：ノードの支持関係を示す。
ｘ1、ｘ2、・・・・ｘn　－＞　n
ｘｉ：antecedent node
n　：consequent node

(3)環境 (environment)：仮説(assumption)からなる集合。

(4)コンテキスト (context)：無矛盾な環境の仮説とそれら仮説から推論できるノードの集合。

(5)ラベル：環境からなる集合で、データが究極的に依存する仮説を示す。ラベルは以下の４つの性質を保存する。
(a)consistent　：ラベルに含まれる各環境が無矛盾である。
(b)sound　：コンテキストに含むべきでないデータは含まない。
(c)complete　：各コンテキストは、含むべきデータはすべて含む。
(d)minimal　：ラベルの各環境が他の環境のsurpersetでない。

(6)Ｎｏｇｏｏｄ：矛盾を起こす環境。

3.1.3 ノードのデータ構造及びその種類

（1）データ　x　を持つノードを次のように表す。

＜x、label、justification＞

x	：データ
label	：ラベル
justification	：理由付け

（2）ノードには、次の4つの種類がある。

(1)前提(premise)　：常に成立（理由付けに成立）
　＜p、｛｛｝｝、｛｛｝｝＞
(2)仮説(assuption)　：自分自身を環境に持つ
　＜A、｛｛A｝｝、｛｛A｝｝＞
(3)仮定ノード(assumed node)　：assumptionを理由付けに持つ
　＜a、｛｛A｝｝、｛｛A｝｝＞
(4)導出ノード(derived node)　：それ以外の上記から導かれるノード
　＜n、｛E1、E2、・・・En｝、｛J1、J2、・・・Jn｝＞

ＡＴＭＳは、ノードと理由付けからなる、ネットワークで知識を管理し、

(a)データに対応するノードの生成
(b)ノードの理由付けの付加
(c)ノードのラベル計算

を行っている。

3.1.4 ＥＳＰを用いたＡＳＴＲＯＮの特徴

ＡＴＭＳの多重世界においては、新たな事実の発見等に伴い、理由付けの追加、矛盾の生成といったイベントが次々に発生する。この動きをより適格に表現するため、ＡＳＴＲＯＮではＥＳＰの特徴であるオブジェクト指向をとり入れ、仮説、理由付け、矛盾といった各要素をオブジェクトとして表している。各構成要素のアトリビュート情報を、オブジェクトのスロットに持つとともに、知識一貫性維持のアルゴリズムを、オブジェクト相互のメッセージ交換により実現している。[飯島,井上 88]

3.2 問題解決器

問題解決器は仮説推論システムにおいて、知識ベース中の知識と真理維持機構から得られる、事実・仮説といった前提あるいは推論の途中結果を用い、推論制御機構のコントロールの元に、実際の問題解決を行う機構である。

3.2.1 問題解決器に要求される条件[de Kleer86c]

（1）適切なjustificationを生成すること

問題解決器は、常に「正しい」justification（理由付け）をＡＴＭＳに与えなければならない。この場合、何が「正しい」かを表現するのは困難であるが、ここでは問題解決器の設計者の意図にあっていることを「正しい」　とすることにする。justificationの生成に際して注意すべきことは、antecedent として指定するノードを必要十分にするということである。さもなければcontext内の知識の一貫性が保証されなくなってしまうからである。

（2）明確な問題解決戦略を持つこと

外部に別の推論制御機構を持つという形にせよ、解決戦略として、問題領域毎にはっきりした戦略を持つべきである。その際、考慮すべき点は、

(a)問題解決器はＡＴＭＳにどのくらいの情報を与えるべきか。
(b)探索をどのような順序で行うか。

等があげられる。

3.2.2 ＡＳＴＲＯＮにおける問題解決器

本来、問題解決器は問題領域に依存したものであるが、ＡＳＴＲＯＮは特定の対象問題を意識せず、仮説を扱える汎用的なルール型問題解決器を用意している。
3.2.1(1)で述べた適切なjustificationの生成をみたすため、ＡＳＴＲＯＮのルール型エンジンでは無駄なルールの発火を避け、必要最低限のjustificationを生成するようにしている。また、3.2.1(2)の明確な問題解決戦略の保持については、6.(1)探索の効率化に述べる。

３．２．３ 問題解決器の構成要素

ＡＳＴＲＯＮにおける問題解決器(３．２．４のversion2のアルゴリズムによる)は以下の構成をとっている。矢印は、データのやり取りを表現している。

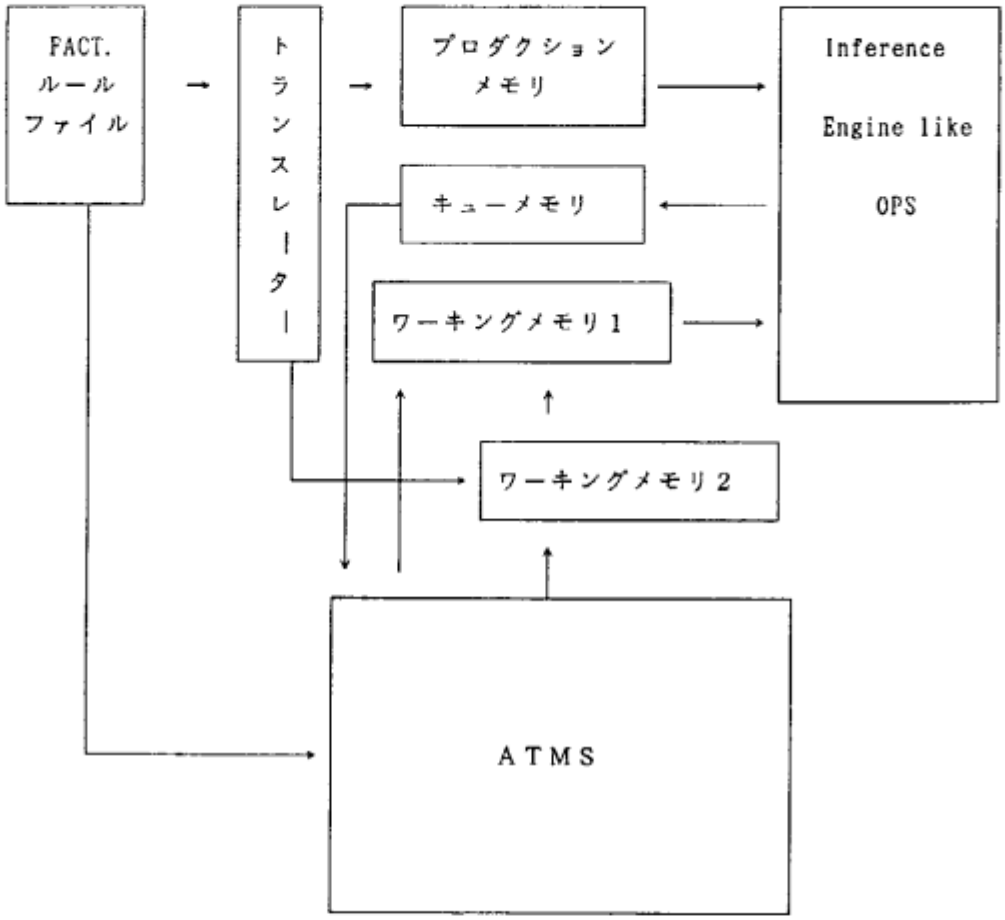


図２．ＡＳＴＲＯＮの問題解決器の構成

(1)assertion　ユーザーから与えられたり、ルールの実行に従い、動的に生成されるデータ構造。３．１．３(2)で述べた４つのデータの内、assumption以外の３つが持つ。ユーザ寄りのデータ表現で、ＡＴＭＳが管理するnodeと１対１に対応する。ルールのマッチングの対象となる、具体的なデータをスロットの値として保持している。
ノードと内部の表現の対応は以下の通り。

内部の表現 ノード	assertion	node	仮説の宣言	前提の宣言
premise	O	O		O
assumption			O	
assumed-node	O	O	(O)	
derived-node	O	O		

(2)ルール　ルールは、全てのコンテキストにおいて、それまでに得られた事実や仮説とマッチングを行う条件部と、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。
ルールは、
＜ルール名＞：＜［条件１、条件２・・条件ｎ］＞
－＞　＜実行部＞
の形式で記述され、入力されると同時に、プロダクション・メモリ内に、格納される。
ルールのマッチングをとる際には、単純なパターンマッチだけではなく、対象となる事実や仮説が信じられているかどうかのチェックも行っており、信じられている（これをINと呼ぶ）時にのみ、マッチングが成功する。
又、ルール発火の結果、理由付けが、
＜条件部＞　－＞　＜実行部＞
の形式で、ATMSに与えられる。

(3)ルール検索キー
これはデータを分類してモジュール化すると共に、ルールとの結び付けを行うために、データ(事実・仮説)とルールに共通に持たせたキーである。このルール検索キーにより、ルールの発火チェック時には、自分に関係のあるキーのデータのみチェックすればよく、マッチンングの効率が高くなっている。

3.2.4 推論アルゴリズム

仮説や事実と、ルールの条件部のマッチングを行う際、ＥＳＰのユニフィケーション機能を利用している。
ＡＳＴＲＯＮでは、推論のアルゴリズムとして、以下の２つを考えている。

＜Version1（ワーキングメモリが１つ）＞

〈1〉 ルールを，その条件部第一条件の先頭要素をキー値とするハッシュの形で，プロダクション・メモリ(PM)に展開する。
〈2〉 新たに、事実・仮説が、あるアクションの実行やユーザの入力により、言明されるとその先頭要素をキー値として持つルールをPMから取り出す。
〈3〉 〈2〉で抽出した各ルールに対し、ワーキングメモリ内に格納されている、すでに言明された事実・ 仮説とのユニィフィケーションをとり、他に条件が無ければそのルールのアクション部をキューイングする。
他に条件があれば、それら残りの条件部に対してユニフィケーションを保存したものを、新たな条件部とするルールを生成し、PMに〈1〉と同様に格納する。
〈4〉 キューメモリを順に実行して実行結果そのものをATMSに送ると，その発火したルールの全ての条件部，実行部を用いて〈条件部〉→〈実行部〉の理由付けをATMSに与える。

＜Version2（ワーキング・メモリが２つ）＞

〈1〉 ルールを，その条件部の各条件を先頭に持つルールをつくり、その条件の先頭要素をキー値とするハッシュの形で，プロダクション・メモリ(PM)に展開する。
〈2〉 新たに、事実・仮説があるアクションの実行やユーザの入力により、言明されると、それら事実・仮説をワーキング・メモリ１(WM1)に格納する。
〈3〉 WM1 から１つ事実・仮説を取り出す。WM1が空なら終了。
〈4〉 〈3〉で取り出した事実・仮説と同じ先頭要素をキー値として持つルールをPMから取り出す。空ならば〈3〉へ。
〈5〉 〈4〉で抽出したルールの１つめの条件と、〈3〉でwm1 から取り出した事実・仮説をマッチさせマッチすれば〈5〉に移り、しなければ、〈4〉へもどる。
〈6〉 〈4〉で取り出したルールの２つめ以降の条件と、ワーキングメモリ２(wm2)内に格納されている事実・仮説とのマッチさせ、すべてマッチすれば、ルールのアクション部をキューメモリに格納する。マッチしなければ、〈4〉へ戻る。
〈7〉 キューメモリを順に実行して実行結果そのものをATMSに送ると，その発火したルールの全ての条件部，実行部を用いて〈条件部〉→〈実行部〉の理由付けをATMSに与える。ルールの発火により、新たな事実・仮説が生成されると、wm1に格納し、その後〈3〉へ。

ただし、現在インプリメントされているのはVersion2のみである。Version2の問題点はjustificationを与える時、antecedent の順序がルールで与えた順序と変わってしまうことである。対象によっては、これは問題となる。また、version2はversion1と比較して、同じ条件を何度もチェックすることがあり、処理の効率が悪い。
それゆえ、version1のアルゴリズムのインプリメントの必要がある。

4．特徴的な機能

4．1 論理における推論ルール

プロダクション・ルールは Modus Ponens（三段論法）により推論の実行が行われるもので、ヒューリスティックなルールの記述に従来用いられてきた。この枠組みだけで、もう少し論理的な構造のもの（例：回路の構造）を扱おうとすると、論理の完全性が保証されない。

この１つの解決手段として、ＡＳＴＲＯＮではルールに論理的推論ルールの記述を許し、And Elimination, Or Elimination, Modus Tollens 等を扱うことができるようにしている。

例として、以下の推論ルールが記述できる。

```
<Modus Ponens>      rule1:  implies(P,Q) & P          -> Q

<Modus Tollens>     rule2:  not(Q) & implies(P,Q)     -> not(P)

<Equivalence>       rule3:  iff(P,Q)                  -> implies(P,Q)
                                                        ,implies(Q,P)

<And Elimination>   rule4:  and(P1,P2,P3..)           -> P1,P2,P3..

<Or  Elimination1>  rule5:  or(P,Q) & implies(P,R) & implies(Q,R)
                                                      -> R

<Or  Elimination2>  rule6:  or(P,Q) & not(P)          -> Q

<Or  Elimination3>  rule7:  or(P,Q) & not(Q)          -> P

<Contradiction>     rule8:  not(P) & P                -> contradiction
```

4.2 仮説の動的生成とDefault推論[Reiter 80]

人間は頭の中で問題解決を行う際、状況に応じて、様々な仮定をしながら推論を行っている。この過程は予め考えられる仮説を全部列挙してから推論するのではなく、必要に応じて仮説を生成し消去する過程と捉えることができる。こういった過程を実現するためにＡＳＴＲＯＮでは、事実から事実を導くだけでなく、事実から仮説を動的に作り出す機能を提供している。これにより、”必要に応じて”という推論が実現される。

一般に、世界に関して不完全な知識しか持っていない場合には、「明かに背反する事実のない限り、あることを結論する」という見切りをつけないと、推論を続けていくことができない。このように、例外を許しつつ通常の状況を現す規則のことを、デフォルトと呼んでいる。

ＡＳＴＲＯＮでは、このデフォルトの実現に、仮説の動的生成機能を応用し、Reiterのnormal defaultルール

$$a(x) : M\ B(x) / B(x)$$

の記述が可能である。内部的には、Ｂを仮説(assumption)とし、ｂを仮説を理由付けに持つ仮定ノード(assumed node)として表現する。Ｂという仮説の存在が矛盾を起こさないなら、ｂという仮定ノードを生成し、$a(x) \wedge B(x) \rightarrow b(x)$ という理由付けとともにＡＴＭＳへ手渡す。これにより、矛盾を生じない限り、Ｂをデフォルトの仮説として推論をしていくことができる。

なお、参考として、ＡＳＴＲＯＮを用いた問題解決の例を、補遺として末尾に掲載している。

5．他の仮説推論機構との比較

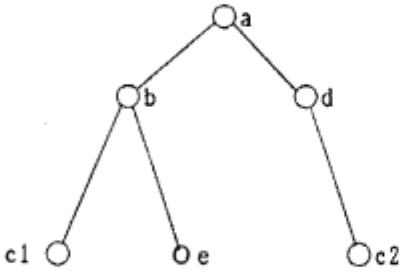
5．1 ＡＲＴのＶｉｅｗｐｏｉｎｔ[Williams 85]

（概要）　ＡＲＴのビューポイント機能は宣言的知識表現であるファクトのデータベースを多重化し、自由に連結することを可能にしている。これにより、単一のデータベースで、仮説の連鎖や時間経過による変化を表現できる。

（特徴）　ＡＲＴでは、２種類のビューポイントをサポートしている。
１つは、時制型ビューポイントで、時間の経過に沿ったファクトの状態変化を記述できる。もう１つは、仮説型ビューポイントで、いくつかの選択肢を仮説として推論を進めて行くものである。

（適用例）　１つ１つの宣言的知識を仮定とするのではなく、一連の順序性を持った宣言的知識の組合せが、仮説の単位となる問題に向いている。例えば、農夫のジレンマ，作業の順序性に拘束条件が依存する問題（「この作業は先にやった方が，かかる時間が少なくてすむ」といった条件がある場合）は、特に時制型Viewpointの適用が向いている。

（問題点）　要素集合の包含関係に着目せずコンテキストとして管理しているため「Ｃが矛盾を生じる」ことが分かっても，そのＣを導くにいたったルートを取り除くだけで，Ｃそのものが，別のルートから導かれてもそれを許してしまう。
これはA→B→C とB→A→C は違うとみなす順序性のある問題は有効だが，普通の組合せ問題に関しては無駄なpropagation（伝播）をしていることになる。つまりＣの下にコンテキストがさらに延びて、どこかＣを原因とする矛盾が生じても、Ｃそのものを否定するのではなく、そのコンテキストのみを否定するため、新たにＣの下にコンテキストを作ってしまう可能性がある。（Ｃそのもが矛盾を起こすため、Ｃの下にいくらコンテキストを作っても、無駄なものを作っていることになる。）



例えばc1が矛盾をおこしたとすると，
(a, b, c1)というcontextをnogoodする。
そのため(a, d, c2) というのも許してしまう。(問題によっては、この方が適していることもある)

図３．ART Viewpointの考え方

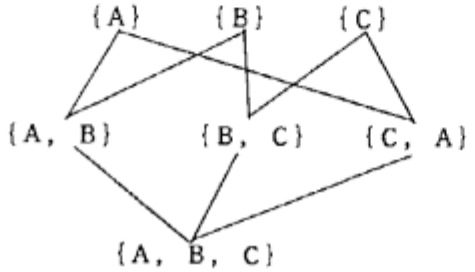
論理的な依存関係は、View Point では陽には扱っておらず、むしろＡＲＴでは論理従属(Logical Dipendencies) で扱っている。

5.2 de KleerのATMS[de Kleer 86a]

（特徴） 一貫性の管理が効率的。個々の宣言的知識を仮説として扱える。ただし宣言的知識の組合せを考える場合、順序性を持たせることはできない。

（適用例） 電子回路の故障診断（故障と思われる箇所を1つの仮説として扱う）その他、知識の順序性のある組み合わせではなく、個々の宣言的知識そのものが、矛盾を生じるような問題。あるいは、宣言的知識の組合せ（順序性なし）が矛盾を起こすような問題。

（問題点） 要素間の依存関係，要素集合の包含関係を管理しているが，あくまで仮説の組合せとして環境をもっているだけで，自分が導かれた経路記憶しているわけではない。そのため，「C->Aという順序が矛盾をおこす」という宣言は「いついかなる時も{C，A}の存在を許さない」という意味になる。従って順序性のある問題への適用は工夫を要する。



C->A という順序が矛盾を起こすと、{A,B,C}という{C,A}のsuper set も矛盾を起こすとみなされてしまう。そのためA->B->C, A->C->B 等の6通りが、すべて矛盾を起こすと扱われる。

図4.de Kleer ATMS のコンテキストの考え方

（応用） 診断 ： 原因と思われる部分に仮説をたてて推論を行えば、順序性は関係ない場合に適用は可。

設計 ： 設計の条件が，設計順序によって変化しなければ適用可。

計画 ： スケジューリングでは、時間的な関係が避けられないため、困難。設計の場合も同様だが，前後関係のない、フェーズ毎に作業を階層化し，その階層毎にATMSを適用するなどの利用が考えられる。

ASTRONのATMSはde Kleer のメカニズムを採用している。

6．今後の課題

ＡＳＴＲＯＮは、仮説推論を問題解決に適用する基本的な枠組みを提供したものである。今後の課題としては以下のものがあげられる。

6．1 探索の効率化［Inoue 88a］

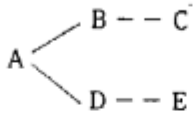
３．２．２の推論アルゴリズムで述べたルールの実行部のキューは、キューされた順に機械的に実行しているだけである。これだけだと、先にキューされていたものの実行によって、ラベルの状態が変化し、それ以降の実行が無駄になる場合が考えられる。
それ故、キューからの実行部取り出しに、スケジュール機能を追加し、

(1)実行によって、影響を受ける仮説の最も少ないものから実行する。
(2)他の実行部の発火によって、ラベルの状態が変化し、発火の必要のなくなったものは、キューから除去する。

等の処理を行うことで、探索の効率化を図る必要がある。
また、ラベル更新がなるべく少なくてすみ、かつ、nogoodの最も簡単な形をなるべく早く見つけ、探索の刈り込みを行うアルゴリズムをＡＴＭＳそのものに組み込むことも考えられる。

6．2 時制的推論の扱い

５．他の仮説推論機構との比較 でも述べたが、ＡＴＭＳでは、多重コンテキストは、無矛盾な環境の仮説とそれら仮説から導かれるノードの集合であり、１つの仮説は複数のコンテキストに含まれるが、仮説そのものは、それらコンテキストのなかで共通である。例えば、



｛A｝が矛盾を起こせば、｛A，B，C｝と｛A，D，E｝の両方が各ノードのラベルから取り除かれ、それ以降、Aを含むコンテキストは延びない。これがＡＲＴのViewpoint等と比べて効率のよい点であるが、反面、｛A，B，C｝が矛盾を起こすと、｛A，C，B｝もラベルから取り除かれてしまい、

A－＞B－＞C

A－＞C－＞B

といった順序性の区別ができない。このためＡＲＴのViewpointのような順序のある解の探索ができず、ＡＴＭＳの適用できる分野を狭めている。
そえ故、順序のあるものも扱え、かつ、ＡＴＭＳのような効率の良いアルゴリズムの開発が必要である。

6．3 制約式の扱い [de Kleer 86c]

問題解決器の実現には、様々なアプローチが考えられるが、積極的に対象問題の仕様や要求条件を扱えるとともに、より効率的な解の探索ができるよう、constraint（制約）の扱えるconsumerアーキテクチャ（C．A．）の導入も考えられる。consumerの概念は以下の通り。

(a) consumerとは，
制約式の形で与えられた条件から、プリコンパイルによって，ノードに付加された、対象に関する制約条件の宣言。

(b) consumerの作成アルゴリズム

① まず，ユーザーは，変数間の関係式で制約式として入力する．
R(x,y,z):x+y-z=0 ;xとyとzには左式の関係がある

② ①で得られた制約式をヒューリスティックな知識をもとに，変形する．（プリコンパイル）

③ ②で得られた式をconsumer用データとして格納するとともに，条件部にあたる変数ノードに，そのconsumerをattachする．

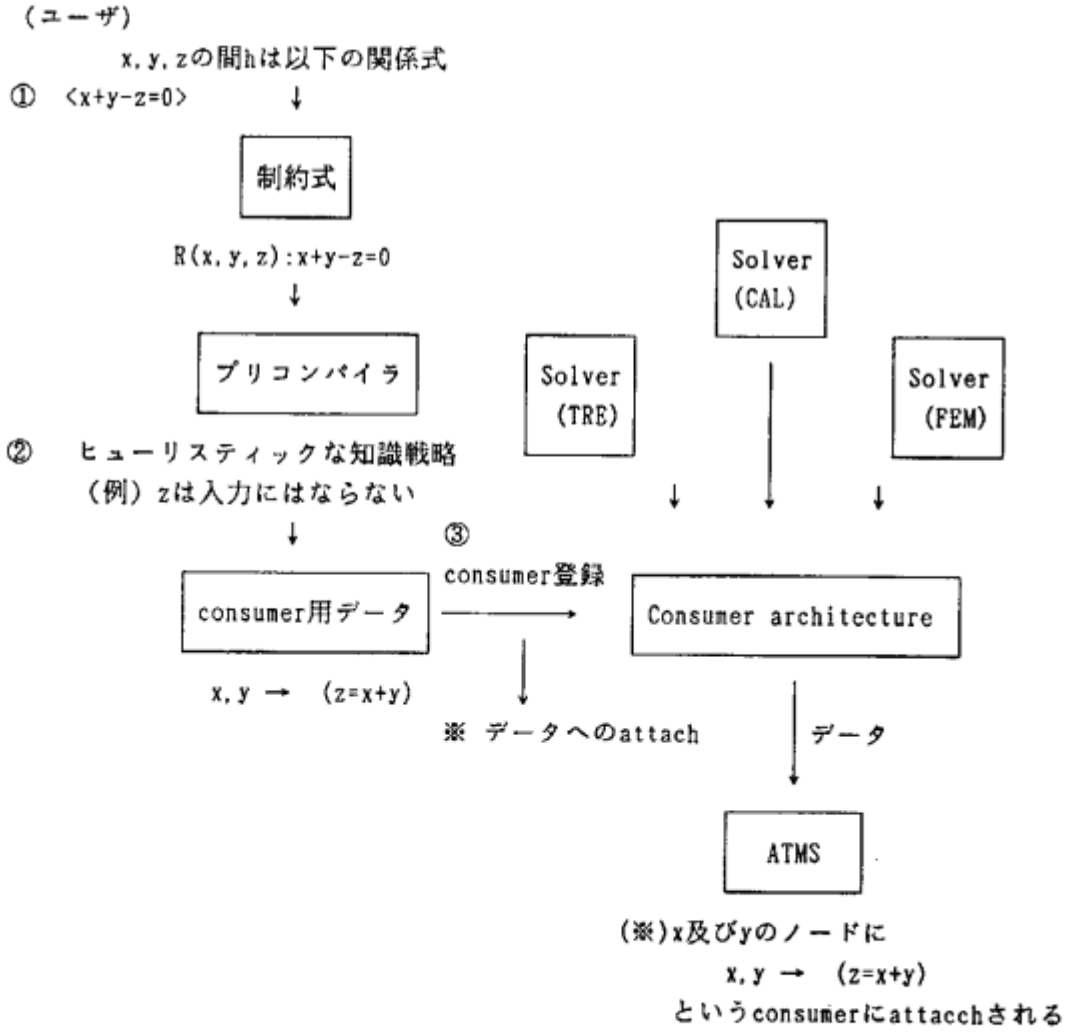


図5．consumer アーキテクチャの概念図

(c) consumerの起動アルゴリズム

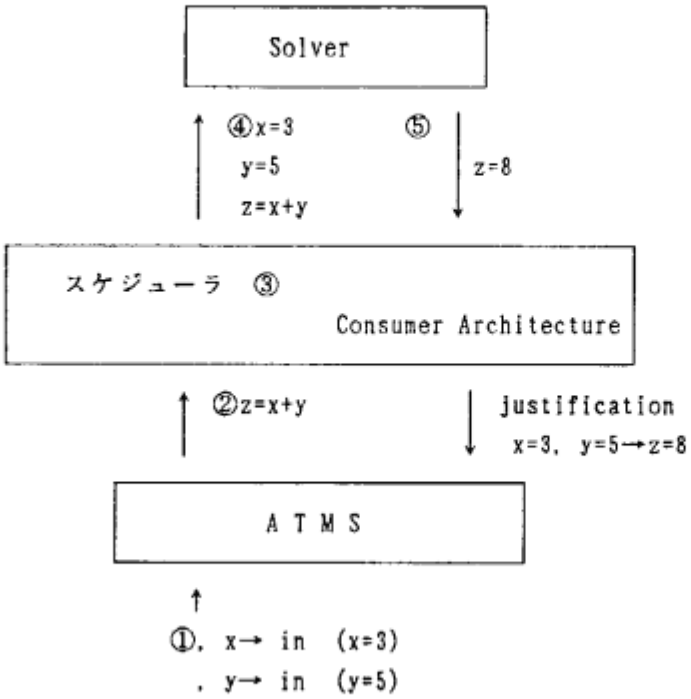


図6.consumer アーキテクチャの動作原理

① なんらかの理由でx,yがinになる。
② x,yに付加されているconsumer
x,y I→ (z=x+y)
が起動され（invoke)C.A.に送られる。
③ ある戦略にしたがって，スケジューラに登録(Queue)する。
④ スケジュール．キューの中から一つconsumerを選びsolverへ渡す。
⑤ ④の情報をもとにsolverは解をもとめC.A.に返す。
⑥ C.A.は
x=3,y=5 → z=8
というJustificationをATMSへ登録する。

(d) 仮説推論とルール型推論との枠組みの比較

consumerアーキテクチャまで備えた仮説推論システムと、従来のルール型システムとを比較すると以下のようになる。

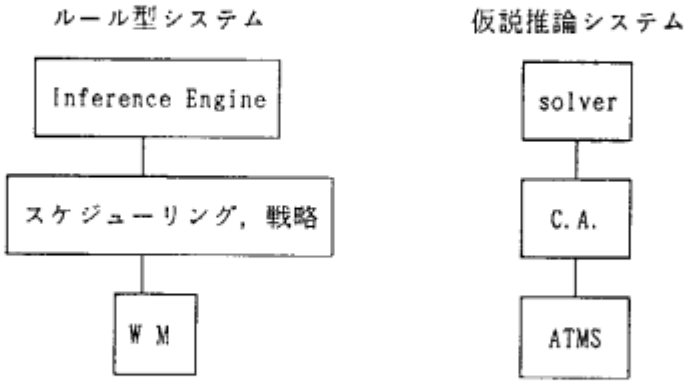


図7．ルール型システムと仮説推論システムの比較

consumer機能を持った仮説推論システムの特徴としては、

. ATMS等でデータの依存関係が記述できる。
. 知識表現はルールだけでなく制約式も与えられる。
. 深い知識（物理式，実験式，構成要素）の表現が扱える。

等があげられる。

(e) ＡＳＴＲＯＮへのインプリメント

(a)-(d) で述べたのは、consumerアーキテクチャの、一般的な枠組みである。制約式として、様々なものを許すためには、制約式のプリコンパイラを、制約式の形式毎に用意しなければならない等、解決すべき課題は多い。ここでは、制約式のプリコンパイルは、すでにされているものとして、consumerアーキテクチャの実現方法を述べる。

（現在の方式）

現在ＡＳＴＲＯＮに事実・仮説を理由付けをつけて付加するためには、直接それらを、ユーザが書いて入れるか、ルールの形で記述し、推論をとおして追加して行くかの２つの方法しかない。
たとえば、

・tweetyはペンギン　－＞　tweetyは飛べない

ということを宣言しようとすると

＜ユーザが直接入力＞

```
assert(penguin(tweety)).
assert(not(fly(tweety))).
justify(not(fly(tweety)),[penguin(tweety)]).
        %[penguin(tweety)] -> not(fly(tweety))の意味
```

とするか、あるいは、

＜ルールによる記述＞

```
assert(penguin(tweety)).
if [penguin(X)] then not(fly(X)).
```

とする、いずれかの方法をとっていた。
しかし、これだけでは、等式や不等式のような制約式は取り扱えない。

（制約式の扱い）

ここでは、等式や不等式などから、直接、ノード・理由付けを作り出す方式を述べる。

(i)制約式の宣言

```
y = f （x1、x2、x3）  ・・(A)
        f （x1、x2、x3）＝a＊x1＊x1＋b＊x2＋x3

y > g （x1、x2、x3）  ・・(B)
        g （x1、x2、x3）＝c＊x1＋x2＊x3

y < h （x1、x2、x3）  ・・(C)
        h （x1、x2、x3）＝x1＊x2＊x3
```

という制約式が与えられたとする。（これ以前のプリコンパイル処理はすでに行われているものとする。

(ii)ルール形式への展開

(i)で与えられた制約式をルールの形式に展開する。

(A)より

```
 if [equal(x1,X),equal(x2,Y),equal(x3,Z)]
 then justify(equal(y,a*X*X+b*Y+Z),[equal(x1,X),
                        equal(x2,Y),equal(x3,Z)]).   ・・(A')
```

(B)より

```
 if [equal(x1,X),equal(x2,Y),equal(x3,Z)]
 then justify(more_than(y,c*X*X+Y*Z),[equal(x1,X),
                        equal(x2,Y),equal(x3,Z)]).   ・・(B')
```

(C)より

```
 if [equal(x1,X),equal(x2,Y),equal(x3,Z)]
    then justify(less_than(y,X*Y*Z),[equal(x1,X),
                        equal(x2,Y),equal(x3,Z)]).   ・・(C')
```

ここで、ルール形式に展開するのは、現在のＡＳＴＲＯＮのProblem Solverが、ルール形式用に作られているからであり、いずれはconstraintから直接justificationを導けるよう改造することが考えられる。
ちなみにルール形式のなかで用いている equal(x1,X) は、変数x1の値を、X にバインドされた値とするという意味である。

(iii)contradictionの宣言

また(A'),(B'),(C')以外に

```
if [less_than(x,X),more_than(x,X)] -> contradiction
if [less_than(y,X),equal(y,X)] -> contradiction
                    .
                    .
```

といった、equal,more_than,less_thanに関する論理的な関係をルールで記述しておく。

(iv)条件部への不等式の記述

これまでは、制約式として不等号を与える枠組みについて述べてきたが、さらにこの機能を拡張して、ルールの条件部でも、制約式から導かれた不等号関係をチェックできるようにする方法を述べる。このようなルールが必要となる理由は、例えばless_than(x,5)という事実・仮説が、ユーザあるいはルールから導かれると、less_than(x,10)を条件部に持つルールが発火しないと、論理的に不完全となるからである。

そこで、less_thanを条件部に含むルール、

```
if less_than(x,10) ・・・
```

があり、ワーキング・メモリに

```
less_than(x,X)   : X<10
```

があれば、マッチングが成功するようにマッチングのアルゴリズムを以下のように修正する必要がある。

inferenceクラスのローカル述語
　rule_match_check5
で、Ｐｍ＝Ｗｍ１しかチェックしていないところを、
　{Ｐｍの３番目の要素}　＞　{Ｗｍ１の３番目の要素}
が成り立てば、failしないようにすればよい。
あるいは、ルールの条件部に test部 を設け prolog 実行コマンドを書けるようにするという方法も考えられる。

(v)キュー実行時のラベルチェック

キューの実行に関しては、以下の機能を追加する。

・キューイングされた実行部のjustificationの中から、antecedentsのラベルが一番小さい順に選び出す。

・取り出したjustification中に含まれた計算式を展開するために、macro_expanderにより、interpretを行う。

・キューの実行によって、antecedentsがＯＵＴになったjustificationは発火させない。（無駄なjustificationの生成を防ぐ）

これにより、等式や不等式によって表現された制約式を扱える仮説推論の枠組みが実現される。その結果、制約とヒューリスティック・ルール、論理的推論ルールを同じ枠組みで扱えるのが特徴である。従って、制約式の使い方などに関するヒューリスティックスがあれば、自然に表現できるため知識コンパイラとして使用できる。

7．仮説推論システムの実現　－ＡＳＴＲＯＮ－

7.1　ＡＳＴＲＯＮの特徴

（１）真理維持機構として、de Kleer によるＡＴＭＳのBasic Algorithms[de Kleer 86a]を忠実に反映している。

（２）問題解決器との組合せにより、論理における推論ルールが記述できる。

（３）問題解決器との組合せにより、normal defaultルールが記述できる.
[Reiter 80]

7.2 クラスの構成

ATMS部については[飯島,井上 88]参照のこと。

（1）Assertionクラス

＜クラス・スロット＞

1	assertions	生成済みのassertionインスタンスを記憶する。
2	counter	生成したインスタンスの数を記憶する。
3	name_to_obj	systemが命名した名前と、オブジェクトとの対応を記憶する

＜クラス・メソッド＞

1	make_assertions	assertionインスタンスの生成を行う

＜インスタンス・スロット＞

1	name	本オブジェクト名を記憶する
2	index	何番目に生成されたオブジェクトかを記憶する。
3	tms_node	本assertionに対応するＡＴＭＳ内のtms_nodeを記憶する。
4	fact	ユーザ・あるいはルールの発火によって導かれた事実・仮説そのもののデータを保存する。
5	key	factの先頭要素。wm1やPMと共通の検索キー。 fly(bird(tweety))なら、flyがキーになる。

（２）inferenceクラス

＜クラス・スロット＞

1	production_memory	入力されたルールを、その条件部第一条件の先頭要素をキーとするハッシュの形で格納する。
2	working_memory1	ユーザあるいはルールの発火により以前に導かれた事実仮説を記憶する。
3	working_memory2	ユーザあるいはルールの発火により新たに導かれた事実仮説を記憶する。
4	q_memory	発火可能なルールの実行部を格納する。
5	macro_expander	キューされていたルールの実行部をinterpretする。

＜クラス・メソッド＞

1	run_rules	working_moemory1,2とpuroduction_memoryをマッチさせ発火可能なルールを選び、その実行部をq_memoryに格納する。
2	rule_match	working_moemory1,2とpuroduction_memoryのマッチを行う。
3	fire_action	キューされていたルールの実行部をファイアさせる。
4	fire_action2	fire_actionのサブルーチン

＜ローカル・メソッド＞

1	rule_match_check	production_memoryからworking_memory2と同じキーを持つルールを取り出す。
2	rule_match_check2	production_memoryからルールを１つづつ取り出す。
3	rule_match_check3	１つのルールの条件部をworking_memory1,2と順にマッチさせ、すべてマッチすれば、そのルールのアクション部をq_memoryにキューイングする。
4	rule_match_check4	production_memoryの各条件の先頭要素と同じものをキーに持つファクトをworking_memory1から探し出す。

5	rule_match_check5	working_memory1とproduction_memoryの各条件とのマッチング
6	status_check	事実・仮説のステータスチェック。 （返却値はin or out）
7	fact_to_node	事実・仮説の内容から、それに対応するnode名を取り出す。
8	fact_to_node1	fact_to_nodeのサブルーチン

（３）interface_atreクラス

＜クラス・メソッド＞

1	assertion	新たな事実・仮説が入ってきた場合それに対応するassertion,tms-nodeを作成する。もしすでにその事実・仮説が存在すれば、新たには作成しない。 キーとするハッシュの形で格納する。
2	assert_all	事実・仮説のリストから、assertionを一度に実行する。
3	referent	assertionのメインルーチン 仮説を記憶する。
4	assume	仮説の生成。assumptionとassumdedノードを作成する。
5	justify	引き数で渡されたantecedentsとconsequent間の理由付けを行う。antecedents、consequentに関する事実・仮説がデータベース内に存在しなければ,referentを使って新たに作成する。
6	justify_assume	justificationの作成に関しては、justificationと同じ。但し、consequentが仮定ノード(assumed node)で、antecedentsにconsequentの元になる仮説(assumption)を含んでいる。Reiterのnormal Default によるデフォルト推論を行う際に用いる。
7	nogood	引き数で渡された仮説の組合せをcontradictionとして、登録する。

＜ローカル・メソッド＞

1	insert	入力された事実・仮説に対応するnodeがなければ、新たに、tms-node,assertionを作成する。
2	fact_to_assertion	事実・仮説からそれに対応するassertionを探し出す。
3	get_fact_list	ノードのリストからそれに対応する事実・仮説のリストを探し出す。ただし、justify_assumeの際できるtemporary nodeは除く。
4	make_tms_nodes	入力されたファクト・リストに対応する、assertion,tms-nodeリストを作成する。

（４）translatorクラス

＜クラス・メソッド＞

1	set_p_memory	入力されたルールから、ルールの各条件部を先頭に持つルールを複製し、それぞれの条件の先頭要素をキーにして、プロダクションメモリに格納する。

＜ローカル・メソッド＞

1	make_rule_key	入力されたルールの第一条件の先頭要素からそのルールのキーを作る
2	make_rule_key1	make_rule_keyのサブルーチン。入力ルールから条件部を１つづつとりだす。
3	make_rule_key2	make_rule_key1のサブルーチン。条件部の先頭要素をキーとする。
4	store_rule	ルールをプロダクションメモリに格納する。
5	sort_list	store_ruleのサブルーチン。 引き数で渡されたキーを持つ条件部が先頭にくるようにルールの条件部をソートする。
6	premise_search	sort_listのサブルーチン。 キーを先頭要素に持つ条件部をサーチする。
7	premise_search2	premise_searchのサブルーチン。 条件が、変数のみ（fly(X)のようなのではなく X の様な場合）であれば、スキップする。それ以外なら、先頭要素をキーとする。
8	premise_delete	条件部のソートの結果以前の位置にあった条件を取り除く。
9	bind_check	sort_listのサブルーチン。 条件部がバインドされているかをチェックする。
１０	bind_check2	bind_checkのサブルーチン。 条件部の２個目以降（キー以外）がすべてバインドされていなければ、それはプロダクションメモリにいれない。
１１	key_to_rule_names	引き数で渡されたキーを持つルールをすべて取り出す。

12　get_rule_name　　ルールからルール名を取り出す。

（5）tre_initクラス

＜クラス・メソッド＞

1　tre_init　　問題解決器のイニシャライズを行う。

7．3 現状の問題点

（1）知識表現

現在、ＡＳＴＲＯＮでは、ルール型の知識表現を用いており、ルールの条件部・実行部のシンタックスが、prolog,ESP like な表現となっている。
このため特に実行部で justification を与える場合

＜例＞ if [penguin(X)] then justify(not(fly(X)),[penguin(x)]).

のように、条件部の[penguin(X)]を実行部でも書かねばならず、ユーザへの負担となっている。
この点を解決するため、パーシング機能を強化し、必要最低限の記述ですむよう改善する必要がある。

（2）ルール・エンジン

３．２．４ 推論アルゴリズム で述べたように当初、２種類の推論アルゴリズムを設計していたが、実際にインプリメントされたのはVersion2のみである。Version2は先述のとおり無駄な条件チェックを行うため、Version1と比較して効率の上でやや問題がある。条件の多いルールでは、特に、その影響がでるためversion1のインプリメントが必要である。

7．4 課題

（1）論理における推論ルールの強化1
－ choose 述語のインプリメント[de Kleer 86b]

4．1論理における推論ルール において、And EliminationやModus Ponens といった論理の枠組みをＡＳＴＲＯＮが用意していることを述べた。
現在のままでも、論理の健全性は満足しているが、幾つかの点で機能拡張が考えられる。その1つがchoose述語である。
ＡＳＴＲＯＮでは、orに関して以下の3つのルールが用意されている。

```
<Or  Elimination1> or(P,Q) & implies(P,R) & implies(Q,R) -> R

<Or  Elimination2> or(P,Q) & not(P)                       -> Q

<Or  Elimination3> or(P,Q) & not(Q)                       -> P
```

しかし、これらはあらかじめ宣言されている事実として or(P,Q) を扱うものであり、ルールの条件部で用いることができるだけで、P または Q を基にして推論を進めるという意味で or を宣言するためには、別の述語を用意する必要がある。この or を論理的に意味があり、かつ、各要素を仮説として宣言するのが choose述語である。choose述語は、

```
                    assume(P)
choose(P,Q)   ->    assume(Q)
                    assert(or(P,Q))
```

を行うもので、ユーザは P または Q のどちらかは必ず成り立つが、どちらが成り立つかは分からない、といった場合に利用することができる。
このchoose述語を追加することにより、or を用いた表現を動的に作りだすことも可能となる。

（２）論理における推論ルールの強化２ － justify_choose 述語のインプリメント

一般に Horn Clause ではpositive literal は高々１個しか含むことができない。それを任意個の positive literal が

a1∧a2∧..∧an -> c1∨c2∨..∨cn

の形で扱えるようにするのが justify_choose 述語である。
記述としては、

if [a1,a2..an] then justify_choose([c1,c2..cn]).

というルール形式で表現する。
そして実行部 発火時に、justify_choose 述語により、choose 述語と同様に

```
                                    assume(c1)
                                    assume(c2)
justify_choose([c1,c2..cn])  ->       :
                                    assume(cn)
                                    assume(or(c1,c2..cn))
```

を行う。
この際、[a1,a2..an]のラベルが {{A}} であったとすると、

ci(1 ≦i ≦ n) のラベルは{{A,Ci}}
or(c1,c2..cn) のラベルは{{A}}

として各Ciを assume する。
このような justify_choose 述語を追加することにより、Horn Clauseを越えた論理の記述も可能になると考える。
なお、ここで述べたchoose,justify_choose の各述語は [de Kleer 86b] とは少し異なるアプローチを用いている。

8．ＡＳＴＲＯＮ使用の手引

8.1 ＡＳＴＲＯＮファイル一覧

（１）ＡＴＭＳ

(1)user.esp	(2)print.esp	(3)node.esp
(4)justification.esp	(5)assumption.esp	(6)environment.esp
(7)nogood.esp	(8)interface.esp	(9)solver.esp
(10)basic.esp	(11)init.esp	(12)librarian.com
(13)atms_load.com	(14)save.com	

（２）問題解決器

(1)util.esp	(2)win.esp	(3)tre_init.esp
(4)translator.esp	(5)inference.esp	(6)assertion.esp
(7)interface_atre.esp		

（３）例題

(1)test1	(2)test2	(3)test3
(4)atre_test1	(5)atre_test2	(6)atre_test3
(7)atre_test4	(8)atre_test5	

8.2 ＡＳＴＲＯＮ立ち上げ手順

（１）必要ファイルのロード

(1) システム・メニューで librarian をオープンする。
(2) マウスで Execute を選択する。
(3) マウスで menu を選択する。
(4) マウスで user defined を選択する。
(5) マウスで atms_load を選択する。
(6) マウスで do it を選択する。
(7) マウスで atre_load を選択する。
(8) マウスで do it を選択する。

（２）debugger によるＡＳＴＲＯＮの起動

(1) システム・メニューで debugger をオープンする。
(2) ＡＳＴＲＯＮの初期化を行う。（:init(#init).:tre_init(#tre_init)）

8.3 ＡＳＴＲＯＮのインタフェース

ＡＳＴＲＯＮでは、以下のユーザ・インタフェースを提供している。

（１）ルール

ルールのシンタックスは以下の通り。

＜rule名＞：if ［＜条件１＞｜＜条件２＞｜・・・］
then ＜実行部＞

＜条件ｎ＞：１つめの条件の、先頭要素は変数でないこと
・・[X(bird),name(tweety)]等は、許されない。
：変数は大文字で記述する
：必ずしも変数を含んでいなくてもよい
：変数を含んだ例・・・name(X)
：変数を含まない例・・bird(name(tweety))

＜実行部＞：基本的には、ＥＳＰのtermはなんでもかける。
：条件部でユニファイした値は、継承される。
：通常は

理由付け	・・justify
仮説から導かれる理由付け	・・justify_assume
矛盾の宣言	・・nogood
事実の宣言	・・assertion
仮説の宣言	・・assume

等を用いる。

（２）事実・仮説の直接入力

ルールの実行部と同じ様に

事実の宣言	・・assertion
仮説の宣言	・・assume

を直接、ユーザが実行すればよい。

（３）出力用インタフェース

ＡＴＭＳにたいする直接のインタフェースは[飯島,井上 88] ５.３ＡＴＭＳのユーザ・インタフェースを参照のこと。改善点として、以下の関数を追加している。

指定assertionの参照
:assertion(#print,<assertion オブジェクト名>)
assertionの一覧表示
:assertions(#print)

特定のnode、assertion、justification、assumption、environmentを参

照するコマンド。

```
:check_node(#node,node名)
:check_assertion(#assertion,assertion名)
:check_justification(#justification,justification名)
:check_assumption(#assumption,assumption名)
:check_environment(#environmnet,environment名)
```

内部情報ウィンドウのｐｍａｃｓ化

```
:pmac(#print)または
:gett(#print!print_obj,Xn).
```

9．おわりに

本稿では、知識の一貫性を管理するＡＴＭＳと、ルールベースの問題解決器からなる仮説推論機構 ＡＳＴＲＯＮ について述べてきた。その特徴として、仮説を動的に生成しながら、問題解決を行うという、より人間的な推論の枠組みを提供していることがあげられる。今後、制約式の取り組み等により、設計問題や計画問題への応用も考えている。[Inoue 88b],[井上 他 88]

謝辞

本研究の機会を与えて下さった、（財）新世代コンピュータ技術開発機構 淵 一博 所長 第５研究室 藤井 裕一室長、ならびにＮＴＴデータ通信（株）開発本部 岩下安男 ＡＩ技術担当部長に深く感謝します。

また、的確なアドバイスを下さった 第５研究室 滝 寛和 主任研究員、ＰＳＩ及びＥＳＰプログラミングに関してご指導いただいた 椿 和弘 研究員、ＡＴＭＳ部を作成された 日本電信電話（株）飯島 勝美氏に感謝致します。

最後に、様々な面で御助力頂きましたＩＣＯＴ第５研究室の皆様に、心から感謝致します。

参考文献

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

[de Kleer 86b] de Kleer, J., "Extending the ATMS", Artificial Intelligence 28 (1986), pp. 163-196

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

[Doyle 79] Doyle, J., "A Truth Maintenance System", Artificial Intelligence 12 (1979), pp. 231-272

[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－」, 第３７回 情報処理学会全国大会, 1988

[飯島, 井上 88] 飯島 勝美, 井上 克巳 「ＥＳＰによるＡＴＭＳ －第１版－」, ICOT TM-467, ICOT, 1988

[Inoue 88a] Inoue, K., "Pruning Search Trees in Assumption-based Reasoning", Proceeding of AVIGNON'88 (1988) PP. 133-151

[Inoue 88b] Inoue, K., "Problem Solving with Hypothetical Reasoning", FGCS-88 (1988) to appear

[井上 編 88] 井上 克巳 (編) 「仮説推論に対する期待とイメージ」, 昭和６２年度KSS WG HYR-SWG報告書, ICOT TM-487, ICOT, 1988

[井上 他 88] 井上 克巳, 永井 保夫, 藤井 裕一, 今村 聡, 小島 俊雄, 「工作機械の設計手法の解析－旋盤の回転機能部品の設計」, ICOT TM-494, ICOT, 1988

[Reiter 80] Reiter, R., "A Logic for Default Reasoning", Artificial Intelligence 13 (1980), pp. 81-132

[Williams 85] Williams, C., "Managing Search in a knowledge-based system", unpublished, 1985

A.補遺　ＡＳＴＲＯＮを用いた問題解決の例

ここでは，本文で紹介しきれなかった，ＡＳＴＲＯＮを実際に用いた推論の例を３つ示す。ＡＳＴＲＯＮの動作を詳しく理解するために利用していただきたい。

A.1　Ｔｗｅｅｔｙ－Ｐｅｎｇｕｉｎ　の例

これは、ＡＳＴＲＯＮを用いたデフォルト推論の例である。
鳥なので飛べると思っていたＴｗｅｅｔｙは、実はペンギンだったので飛べなかったというストーリを、表現している。

（１）ノードの関係

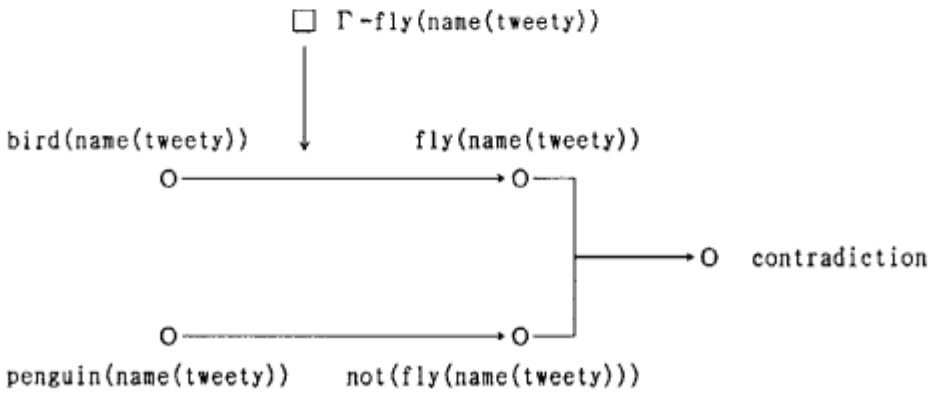


図９.Tweety-Penguin のノード関係図

（２）ルール

```
rule1: if [not(X),X] then nogood([not(X),X])
                              %論理矛盾

rule2: if [bird(name(X))] then justify_assume(fly(name(tweety))).
                                  %通常、鳥は飛ぶ

rule3: if [penguin(name(tweety))] then justify(not(fly(name(tweety)))).
                              %ペンギンは飛べない
```

A.2 ハックルベリー・フィンの例

これは、ＡＳＴＲＯＮを用いた一般的な推論の例である。
平日、映画をみているハックルベリー・フィンをみたトム・ソーヤが、一度はあいつは不良じゃないかと疑ったものの、そんな奴ではないと思いだして、結局学校が休みだったというストーリを、表現している。

（１）ノードの関係

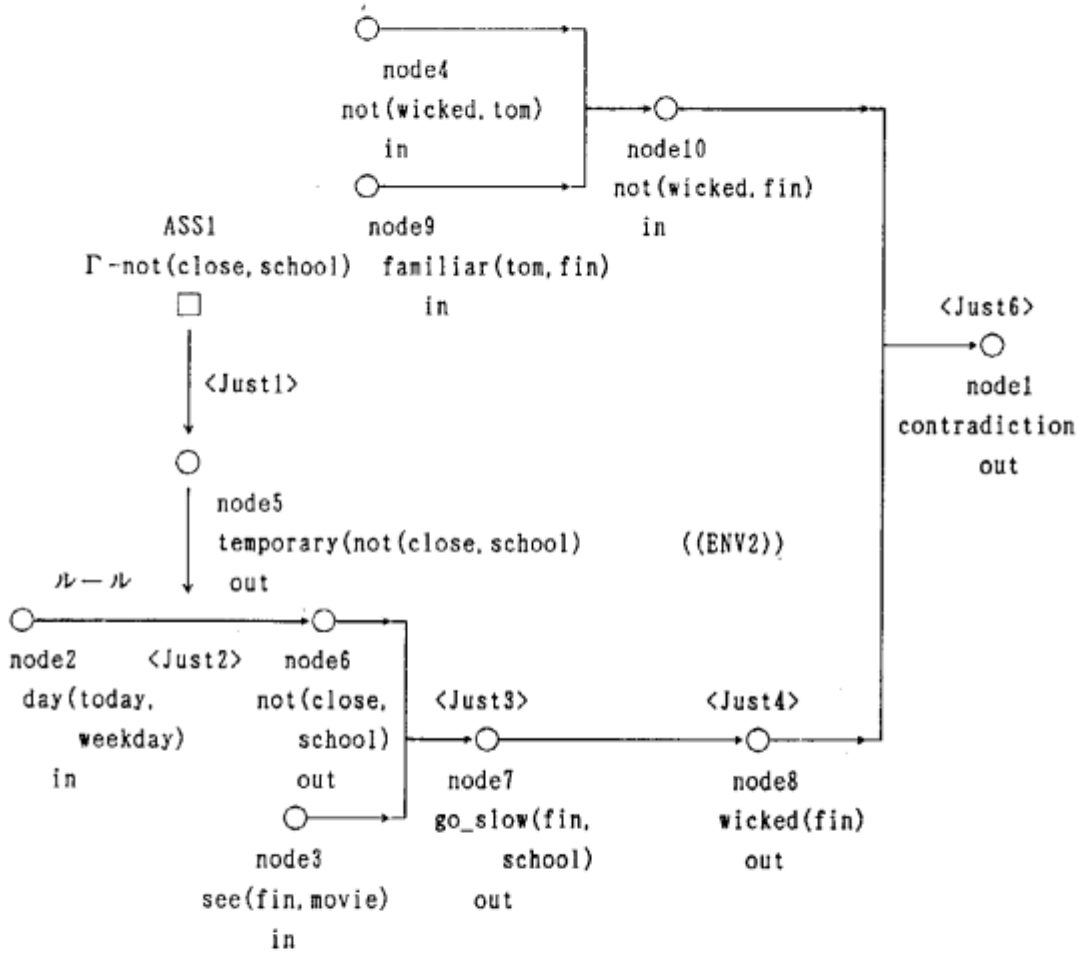


図１０．ハックルベリー・フィン のノード関係図

（２）ルール

```
rule1: if [not(X),X] then nogood([not(X),X])
                              %論理矛盾

rule2: if [day(X,weekday)] then justify_assume(not(close,school)).
                                  %通常、平日は学校は休みではない

rule3: if [not(close,school),see(X,movuie)]
       then justify(go_slow(X.,school)).
                              %学校が休みでないのに、映画を見てるのは不良だ

rule4: if [go_slow(X.,school)] then justify(wicked X).
                                  %通常、平日は学校は休みではない

rule5: if [familiar (Man1,Man2),not(wicked,Man1)]
       then justify(not(wicked,Man2)).
                                  %不良でない人の知合いは不良でない
```

A．3 論理における推論ルールの例

これは、ＡＳＴＲＯＮを用いた論理における推論の例である。
４．２でのべた、And Elimination や Modus Tolensといった論理ルールを用いて推論を進めていく。

（１）ノードの関係

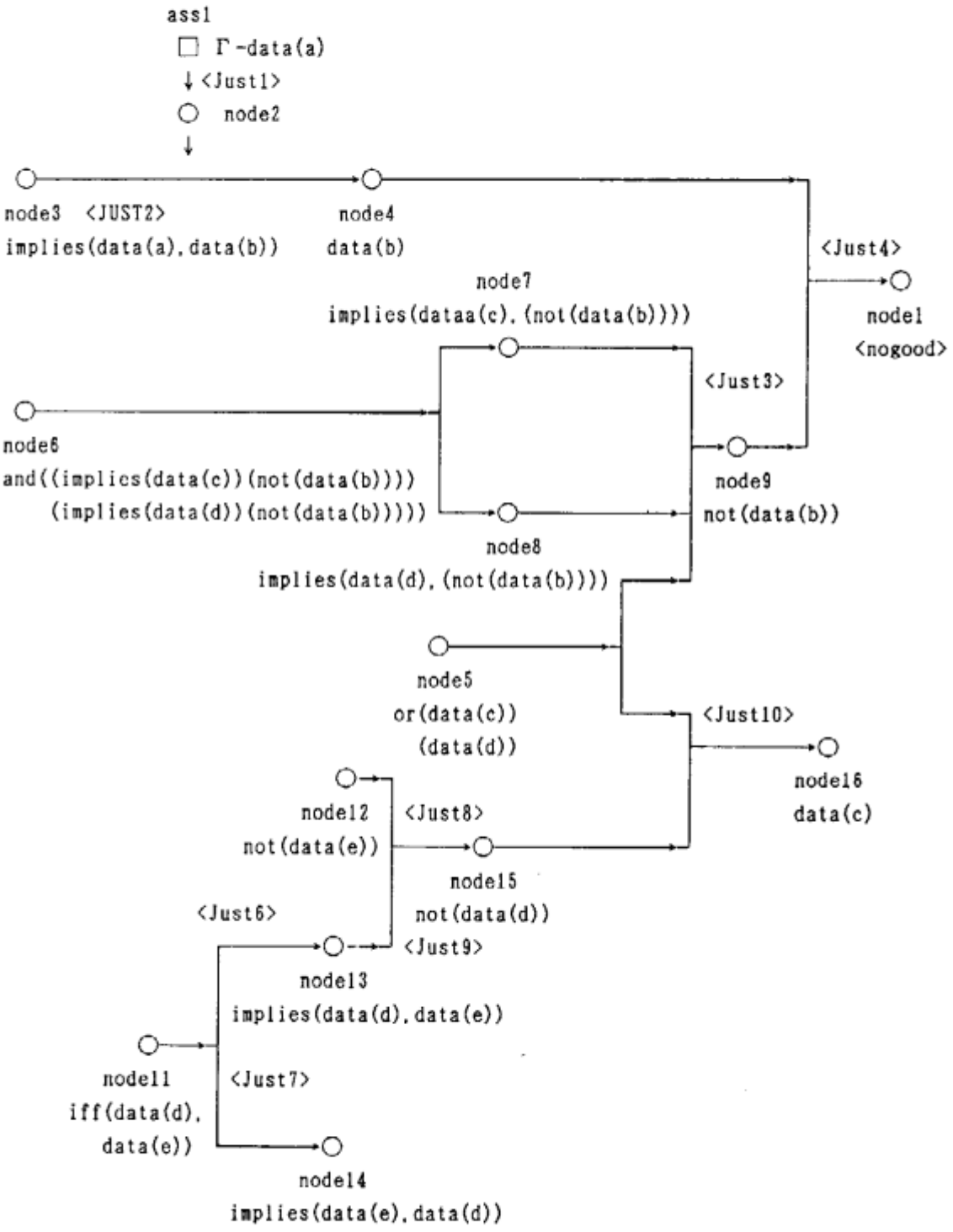


図１１．論理における推論ルールのノード関係図

（２）ルール

```
<Modus Ponens>
   rule1:  if[implies(P,Q), P] then justify(Q,[implies(P,Q), P])

<Modus Tollens>
   rule2:  if[not(Q),implies(P,Q)]
           then justify(not(P),[not(Q),implies(P,Q)])

<Equivalence>
   rule3:  if[iff(P,Q)] then justify(implies(P,Q),[iff(P,Q)]),
                             justify(implies(Q,P),[iff(P,Q)])

<And Elimination>
   rule4:  if[and(P1,P2,P3..] then assert(P1),assert(P2),assert(P3)..

<Or  Elimination1>
   rule5:  if[or(P,Q),implies(P,R),implies(Q,R)]
           then justify(R,[or(P,Q),implies(P,R),implies(Q,R)])

<Or  Elimination2>
   rule6:  if[or(P,Q),not(P)] then justify(Q,[or(P,Q),not(P)])

<Or  Elimination3>
   rule7:  if[or(P,Q),not(Q)] then justify(P,[or(P,Q),not(Q)])

<Contradiction>
   rule8:  if[not(P),P] then contradiction
```