TM-0467

ESPによるATMS

－第1版－

飯島勝美，井上克巳

March, 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

TM

ＥＳＰによるＡＴＭＳ

－ 第１版 －

１９８８年２月

飯島　勝美（ＮＴＴ）

井上　克己

目次

１．ＡＴＭＳの概要及びそのメカニズム

[de Kleer 86a]

１．１　真理維持機構（ＴＭＳ）とは

予め与えられた仮説や仮定をもとに推論を行う「仮説推論」のような非単調推論を行うためには、データベースの無矛盾性を管理する何らかの真理維持機構（Truth Maintenance System）が必要となる。

この真理維持機構の役割は、次の３点にまとめられる。

①データベースが無矛盾であることを保証する。
②問題解決器が非単調推論を可能とする。
③問題解決器が推論した結果を管理する。（再計算の回避）

１．２　Doyle のＴＭＳ と de Kleer のＡＴＭＳ

Doyle のＴＭＳは、あるデータを信じたり、用いたりしたときの理由を記録するという方法で（１）に上げるような役割を果している。つまり、理由づけの集合から信念を決定したり、新しく追加された理由づけと整合するように信念の集合を更新するという方法でデータベースの一貫性を保持している。またＴＭＳは、いわゆる chronological backtracking が持つ矛盾の再発見や推論の再発見などの冗長な計算を回避して、矛盾の原因になっている選択までバックトラックする（dependency-directed backtracking）探索制御を持つ。

しかし、Doyle のＴＭＳは次のような欠点を持っている。

①複数の解を比較することは困難である。
②状態を変更することは困難である。
③矛盾する２つの仮説が存在する場合に、２つの仮説を独立に推論することはできない。
④推論の進行とともに理由づけが変更され、仮説とされていたデータが変化する。
⑤解を見つけるために多くの資源を消費することがある。
⑥過去に導いたデータを再び導く可能性がある。

上記のような欠点の多くは、ＴＭＳが多重コンテキストを持つことができないことに起因する。

一方、de kleer のＡＴＭＳ（An Assumption-based TMS）は、ＴＭＳと同様にデータの依存関係に基づいてコンテキストの一貫性を管理するものであるが、仮説の組合せを考慮することにより多重コンテキストの問題に対処していることから、現在注目されている真理維持機構のである。

ＡＴＭＳは、多重コンテキストを扱えるため上記に上げた Doyle のＴＭＳの多くの欠点を改善している。

ここで、Doyle のＴＭＳと de Kleer のＡＴＭＳの解の探索の効率を比較すると次のようになる。

①ＴＭＳは１つの解の探索に向いているため、複数（多くの）解の探索はコストパフォーマンスが悪い。
②ＡＴＭＳは、複数の解（全解）を同時に求めるような探索に向いているため、少しの解の探索はコストパフォーマンスが悪い。

１．３　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、データは複数のコンテキストに含まれる。
②多重コンテキストで同時に問題解決する。
③各仮説世界が無矛盾であれば、データベース全体は無矛盾である必要はない。
④コンテキストの参照は容易である。
⑤探索空間の別の点に移動することは容易である。
⑥探索戦略はＡＴＭＳの外にある。

１．４　問題解決器とＡＴＭＳの関係

問題解決器とＡＴＭＳの関係は、次図が示すように、問題解決器がデータ（仮説を含む）及び理由づけをＡＴＭＳに与えると、ＡＴＭＳが効率的に可能な全てのコンテキストを決定することである。ＡＴＭＳはコンテキストを決定するために、

①データに対応するノードの生成
②ノードのラベル計算
③ノードへの理由づけの付加

を行っている。一方、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に教わりながら推論を進めていく。問題解決器が、コンテキストの内容を直接参照することは希である。

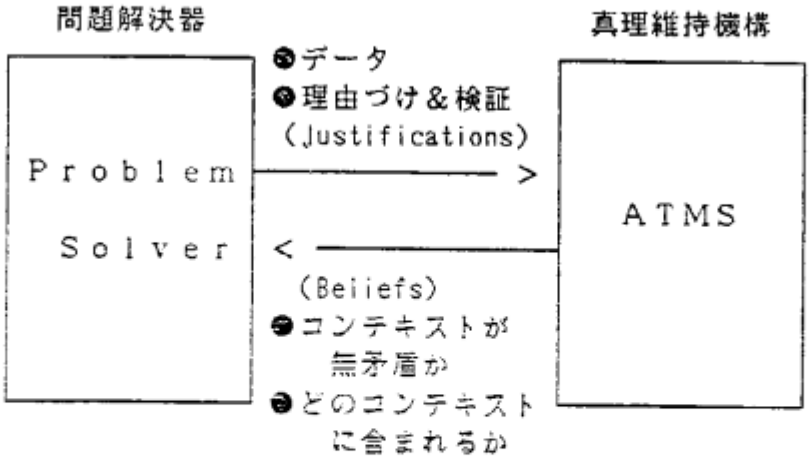


図１．１　問題解決器とＡＴＭＳの関係

１．５　用語

（１）ノード　：問題解決器のデータに対応
(node)　仮説（assumption）もノードの一種
（２）理由づけ　：x_1, x_2, ・・・ ＝＞ n
(justification)　x_i：antecedent node
n ：consequent node
（３）環境　：仮説（assumption）からなる集合
(environment)
（４）コンテキスト　：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
(context)　の集合
（５）ラベル　：環境からなる集合で、各ノードが持つ
(label)　（データが究極的に従属する仮説を示す）
（６）ラベルの状態
① consistent　：ラベルに含まれる各環境が無矛盾である
② sound　：コンテキストは含むべきでないデータを含まない
（consequent node がラベルの各環境から導かれる）
③ complete　：含むべきデータは全て含む
④ minimal　：ラベルの各環境が他の環境のsupersetでない
（７）ｎｏｇｏｏｄ　：矛盾を起こす環境

１．６　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	：	理由づけ

（２）ノードには、次の４つの種類がある。

① premise　　：常に成立（理由づけなしに成立）
＜p、｛｛｝｝、｛（）｝＞

② assumption　：自分自身のみ環境に持つ
＜A，｛｛A｝｝，｛（A）｝＞

③ assumed node：assumptionを理由づけに持つ
＜a，｛｛A｝｝，｛（A）｝＞

④ derived node：それ以外の上記から導かれるノード
＜n，｛E_1，E_2，・・・｝，｛J_1，J_2，・・・｝＞

１．７　ラベル更新アルゴリズム

問題解決器がＡＴＭＳに理由づけ（justification）を与えると、ＡＴＭＳはそのノード（consequent node）のラベルを再計算する。なお、そのノードのラベルを更新することによって影響を受けるノードは、再帰的に順次更新する。

次のようなノードｎに対する理由づけが追加されたとき、

justification ：　x_1，x_2，x_3，・・・　＝＞　n

影響を受けるノードのラベルは、次のようなアルゴリズムにより計算できる。

ラベル更新アルゴリズム

[1] 次式により、ノードのcomplete labelを求める。

$$\bigcup_{x} \{x \mid x = \bigcup_{i} x_i \quad \text{where } x_i \in L_{ik}\}$$

L_{ik}：k番目のjustificationのi番目のantecedent nodeのラベル

[2] [1]で求めたcompleteなラベルをsound かつ minimal にする。

矛盾する環境及び他を包含する環境を除く。

[3] [2]で求めたラベルの条件により次の処理を行う。

<IF> 計算前のラベルと等しい？

<THEN> そのノードに関する処理を終了する。（RETURN）

<ELSE IF> 矛盾する？

<THEN> ●矛盾する環境をnogoodテーブルへ追加する。
●全ノードのラベルに対して矛盾する環境とそれを含む環境を除く。

<ELSE> ●このノードをantecedentに含む全justificationのconsequent nodeのラベルを再計算する。
（consequent nodeについて[1]から実行する。）

２．ＡＴＭＳの課題及び動向

２．１　ＡＴＭＳの課題

（１）仮説の組合せ爆発の問題

ＡＴＭＳは、ＴＭＳとは異なり多重コンテキストを取り扱うことができるため十分に評価できる。また、問題解決器とは独立していてＡＴＭＳの果たす役割は明確でわかりやすい。

しかし、その反面ＡＴＭＳ自身は問題解決のための戦略を持たないために生成された仮説を単純に組み合わせるという、いわゆる組合せ爆発の問題を内在している。現在、ＡＴＭＳで仮説の組合せ爆発の問題を回避するためには、矛盾を引き起こす仮説の組合せ（ｎｏｇｏｏｄ）を十分に宣言する必要がある。

（２）計算コスト大

ＡＴＭＳの主たる役割は、あるノードの理由づけが与えられたとき、そのノードに対するラベルを計算することである。また、１つのラベルの変更によって影響するノードについても同様の計算を施すことである。

そして、複数の環境間の包含関係のチェックあるいはラベルの構成要素である個々の環境が矛盾しないかといったチェックは、各ラベルを計算する毎に行う必要があり多大な計算コストを必要とする。

技術的な改善策としては、ビット・ベクタあるいはハッシュ・テーブル等の利用が考えられるが、この程度のオーダの改善では不十分である。

（３）メモリ消費大

データを記述するためのＡＴＭＳの内部表現は、３章で示されるように、多くのメモリを必要とする。必要な部分のみの記述あるいは仮説世界のfocusingなどを考える必要がある。当然、ＡＴＭＳの内部表現量は、計算コストに関わる問題である。

（４）探索制御が困難

問題解決器（あるいはユーザ）がＡＴＭＳの探索を制御することは困難であり、また解の探索過程を直接見ることができないためデバックが難しい。

（５）時制的推論は困難

ＡＴＭＳと同様な機能を含む、商用ツールＡＲＴ[Williams 85]は、環境（仮説の組合せ）を生成するとき、仮説が生成された時間的順序を考慮している。そのため、時間的な順序を考慮しなければならない、例えば「農夫のジレンマ」などの問題を容易にとけるが、仮説の組合せしか考慮していないＡＴＭＳではこの種の問題を解くのは困難である。

（６）表現能力の拡張

basic ＡＴＭＳでは扱える論理式として次のようなＨｏｒｎ節

[justification] $a_1 \wedge a_2 \wedge \cdots \wedge a_n \rightarrow b.$

[premise]　　b.
[nogood]　　$\neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_i$

のみを許しているが、否定や論理和を取り入れることにより表現能力を拡張することができる。[5]では否定と肯定節（positive clause）を入れており、[32]では任意の節に拡張している。しかし、ラベルの計算をcomplete かつ minimal に保つためには、導出（resolution）が必要となり、計算量がbasic ＡＴＭＳ に比べて増大する。したがって、表現能力の拡張と計算量の増大というトレード・オフが生じる。

効率的な論理ベースのＡＴＭＳは今後の重要な研究テーマの１つである。

（７）ＡＴＭＳの並列化[de Kleer 86c]

ここでconsumerをノードに付加された問題解決のための手続きとする。consumerは実行順序、ノードの状態を考慮してスケジュールしなければならないが、それを１つのプロセスとみなして並列化することにより制御の負担を減少することができる。

consumerアーキテクチャ以外にＡＴＭＳ自体の並列化も有効性が認められる。

具体的には、個々のプロセッサに１つの無矛盾なコンテキストを割り当てればよい。但し、各プロセッサ間で情報の共有ができるようにする。なぜなら、矛盾発見や環境生成などの機能に関しては、非ローカルなものとなるためにＡＴＭＳはデータベース全体のアクセスが可能でなければならないからである。

２．２　ＡＴＭＳの動向

（１）不確実な知識とＡＴＭＳとの融合

ＡＴＭＳは、知識ベースにおける不完全な知識の取扱いに対するアプローチであり、一方、不確実な知識の取扱いは、ファジィ理論やDempster-Shafer理論等によってアプローチされてきた。これまで独立に行われてきた両アプローチを融合する研究として、中川らの研究[中川 88]がある。

（２）問題解決器との融合

ＡＴＭＳは、問題解決器とは独立して知識ベースの無矛盾性だけを管理するシステムである。推論時間の高速化を図るために、問題解決器の中にＡＴＭＳ機能を取り込むアプローチがある。飯島ら[飯島 87]は、プロダクション・システムとＡＴＭＳを分離せず、拡張したrete treeに証明木を記憶させてこの証明木の上で推論するメカニズムを提案している。

（３）探索戦略の導入

ＡＴＭＳは、問題解決のための戦略を持たないため、生成された仮説はすべて同じ比重で取り扱う。ｎ個の仮説が生成されたとすると、2^n個の仮説の組合せがを考えなければならない。当然、ｎが増大すると仮説の組合せ爆発が起こる。

井上[井上 87]は、増大した仮説空間から探索木の枝刈を効率的に行うアルゴリズムを提案している。また、de Kleer[de Kleer 86b]は、枝刈を早期におこなうための効率的なbacktrackingのアルゴリズムを提案している。

（４）他パラダイムへの応用

仮説推論は、問題解決過程で競合する知識や不完全な知識を取り扱う場合に、候補となる知識を仮説と見立てて処理を進めて行く推論形態であり、仮説推論を行うためには、ＡＴＭＳのような知識ベースの真理維持機構が必要となる。

したがって、ＡＴＭＳは仮説推論を要素技術とするような複合技術への導入が考えられる。次に、ＡＴＭＳの機能を応用できる分野の例をいくつか上げる。

①定性推論
定性制約式におけるパラメータの設定や可能な状態遷移の同定など

②分散協調問題
各問題解決器における知識ベースの管理やリザルト・シェアなどの複数の問題解決器間の協調活動など

③設計型問題
設計過程は、多段の仮説の生成、選択、検証から構成されていると捉えることができる。真理維持機構などの要素技術は必須である。

④知識獲得
獲得した知識の解釈可能性の限定など

３．ＡＴＭＳのインプリメント

本インプリメントは、マシン：ＰＳＩⅡ、言語：ＥＳＰで行った。

３．１　試作したＡＴＭＳの特徴

（１）本システムは、de KleerのBasic algorithms[de Kleer 86a](4.7章)を忠実に反映している。

（２）本システムは、ＡＴＭＳのモデル表現にあたりオブジェクト指向を取り入れている。オブジェクト指向は、自律的に活動できる各要素（オブジェクト）がオブジェクト間をメッセージを交換をしながら分散協調的に目的の機能を果たすパラダイムと捉えることができる。この考えに基づき、本システムのクラスを次のように構成した。なお、各クラスに任された機能は、クラス内に処理を埋め込むことにより実現されているため、各クラスがどんな処理を行っているかはクラスのメッセージの呼び出し形式（メソッド）を参照すれば理解できる。（３．３参照）

①node クラス
　ノード・オブジェクトの生成及び管理
②justification クラス
　理由づけ・オブジェクトの生成及び管理
③assumption クラス
　仮説・オブジェクトの生成及び管理
④environment クラス
　環境・オブジェクトの生成及び管理
⑤nogood クラス
　矛盾する環境（nogood）の管理
⑥interface クラス
　ユーザ（問題解決器）に対するＡＴＭＳとのインタフェースの提供
⑦solver クラス
　ＡＴＭＳを用いた簡単な問題解決器
⑧init クラス
　ＡＴＭＳ全体の初期化
⑨print クラス
　ＡＴＭＳ内部表現の表示
⑩basic クラス
　各クラスで用いる共通的な手続きの管理
⑪user クラス
　ユーザ（問題解決器）がＡＴＭＳへ与える命令（メソッド）の管理
　（interface クラスが提供するメソッドより簡易）

３．２　クラスのスロット構成

（１）node クラス

クラス・スロット		
1	contra_node	矛盾を示すノード・オブジェクトがどれか記憶する。
2	nodes	生成済みnode・インスタンスを記憶する。
3	node_queue	ラベル更新を待つノード列を記憶する。
4	counter	生成インスタンス・オブジェクト数を記憶する。
5	name_to_obj	ATMSが命名した名前とオブジェクトの対応を記憶する

インスタンス・スロット		
1	name	本オブジェクト（ノード）の名前を記憶する。
2	index	何番目に生成されたオブジェクトか記憶する。
3	datum	問題解決器が与えたデータを記憶する。
4	status	本ノードが信じられるか否かを示す。（IN　or　OUT）
5	label	本ノードのラベルを記憶する。
6	c_justs	本ノードを導く理由づけ(justifications)を記憶する
7	a_justs	本ノードをantecedents部にもつ理由づけを記憶する
8	rules	このノードに付加するルールを記憶する。
9	plist	このノードに付加する属性と値を記憶する。

（２）justification クラス

クラス・スロット		
1	justifications	生成済みjustification・インスタンスを記憶する。
2	counter	生成インスタンス・オブジェクト数を記憶する。
3	name_to_obj	ATMSが命名した名前とオブジェクトの対応を記憶する

インスタンス・スロット		
1	name	本オブジェクト（ノード）の名前を記憶する。
2	index	何番目に生成されたオブジェクトか記憶する。
3	type	本理由づけのタイプ（ユーザ指定）を記憶する。
4	consequence	本理由づけのconsequenceを記憶する。
5	antecedents	本理由づけのantecedentsを記憶する。

（３）assumption クラス

クラス・スロット		
1	assumptions	生成済みassumption・インスタンスを記憶する。
2	counter	生成インスタンス・オブジェクト数を記憶する。
3	name_to_obj	ATMSが命名した名前とオブジェクトの対応を記憶する

インスタンス・スロット		
1	name	本オブジェクト（ノード）の名前を記憶する。
2	index	何番目に生成されたオブジェクトか記憶する。
3	node	本仮説の内容を持つノード・オブジェクトを記憶する
4	environments	本仮説を含む環境を記憶する。

（４）environment クラス

クラス・スロット		
1	empty_env	premise nodeが持つ空の環境オブジェクトを記憶する
2	env_table	仮説数をインデックスとして、環境をテーブル形式で記憶する。
3	environments	生成済みenvironment・インスタンスを記憶する。
4	indices	env_tableのインデックスを記憶する。
5	counter	生成インスタンス・オブジェクト数を記憶する。
6	name_to_obj	ATMSが命名した名前とオブジェクトの対応を記憶する
7	env_work	環境生成作業用スロット

インスタンス・スロット		
1	name	本オブジェクト（ノード）の名前を記憶する。
2	index	何番目に生成されたオブジェクトか記憶する。
3	ass_size	本環境が持つ仮説の数を記憶する。
4	assumptions	本環境が持つ仮説を記憶する。
5	nodes	本環境をラベルに持つノードを記憶する。
6	contradictory	本環境が矛盾するか否かを示すフラグを記憶する。

（５）nogood クラス

クラス・スロット		
1	nogood_table	仮説数をインデックスとして、矛盾する環境をテーブル形式で記憶する。
2	indices	nogood_tableのインデックスを記憶する。

※nogoodクラスにインスタンス・スロットは存在しない。

※ 以下のクラスには、クラス・スロット及びインスタンス・スロットは存在しない。
(6) interface クラス　(7) solver クラス　(8) init クラス
(9) print クラス　(10) basic クラス　(11) user クラス

3．3 各クラス内の手続き（メソッド）

（1）node クラス

クラス・メソッド		
1	:make_node	ノード・オブジェクトを生成する。

インスタンス・メソッド		
1	:node_updating	ノード（ラベル）を更新する。
2	:label_updating	ラベルを更新する。 （:node_updatingの下位メソッド）

（2）justification クラス

クラス・メソッド		
1	:make_justification	理由づけ（justification）・オブジェクトを生成する。

インスタンス・メソッド		
1	:get_envs_from_just	理由づけから環境を求める。

（３）assumption クラス

クラス・メソッド		
1	:make_assumption	仮説（assumption）・オブジェクトを生成する。
2	:get_env_from_asss	仮説から環境を生成する。
3	:assumption_p	オブジェクトが仮説か否か判断する。

※assumption クラスにインスタンス・メソッドは存在しない。

（４）environment クラス

クラス・メソッド		
1	:make_environment	環境（environment）・オブジェクトを生成する。
2	:subsumed_check	環境（複数）がある環境を含むか否かを判断する。
3	:env_subsumed_p	ある環境が他の環境を含むか否かを判断する。
4	:get_new_envs	いくつかのの環境から組み合わせた環境を求める。
5	:union_envs	２つの環境を合成する。
7	:push_env	環境に仮説を追加して新しい環境を作る。
8	:push_env_table	環境をenv_tableに登録する。
9	:lookup_env	ある環境が存在するか探す。
10	:env_contradictory_p	環境が矛盾するか否かをチェックする。
11	:remove_env_from_labels	各ノードのラベルからある環境を除く。

※environment クラスにインスタンス・メソッドは存在しない。

（５）nogood クラス

クラス・メソッド		
1	:nogood_updating	矛盾を導く理由づけからnogoodを更新する
2	:push_nogood_table	矛盾する環境をnogood_tableへ登録する。

※nogood クラスにインスタンス・メソッドは存在しない。

（６）interface クラス

クラス・メソッド		
1	:lookup_assumption	あるノードを指示する仮説があれば、その仮説のオブジェクトを求める。
2	:assume_node	あるノードを指示する仮説を立てる。
3	:premise_node	あるノードがpremiseであることを宣言する。
4	:nogood_nodes	矛盾するノードの組合せを宣言する。

※interface クラスにインスタンス・メソッドは存在しない。

（７）solver クラス

クラス・メソッド		
1	:breadth_search	横型探索で解を求める。
2	:depth_search	縦型探索で解を求める。

※solver クラスにインスタンス・メソッドは存在しない。

（８）init クラス

クラス・メソッド		
1	:init	ＡＴＭＳ管理を初期化する。

※init クラスにインスタンス・メソッドは存在しない。

（9）print クラス

クラス・メソッド		
1	:printf	ATMS出力ｳｨﾝﾄﾞｳにﾌｫｰﾏｯﾄ付き出力をする。
2	:status	指定 NODEの状態(IN or OUT)を表示する。
3	:statuses	全 NODE の状態(IN or OUT)を表示する。
4	:node	指定 NODE を表示する。
5	:nodes	全 NODE を表示する。
6	:justification	指定 JUSTIFICATION を表示する。
7	:justifications	全 JUSTIFICATION を表示する。
8	:assumption	指定 ASSUMPTION を表示する。
9	:assumptions	全 ASSUMPTION を表示する。
10	:environment	指定 ENVIRONMENT を表示する。
11	:enviornments	全 ENVIRONMENT を表示する。
12	:node_justs	指定 NODE の JUSTIFICATIONS を表示する
13	:env_with_index	指定 INDEX を持つ ENVIRONMENT を表示する。
14	:env_datum	指定 ENVIRONMENT が持つ ASSUMPTIONS のデータを表示する。
15	:env_datums	全 ENVIRONMENT に関して ENVIRONMENT が持つデータを表示する。
16	:table	ENVIRONMENT ﾃｰﾌﾞﾙ と NOGOOD ﾃｰﾌﾞﾙ の内容を表示する。
17	:context	指定 NODE がラベルを持つ場合には、そのJUSTIFICATION（理由づけ）をたどる。

※print クラスにインスタンス・メソッドは存在しない。

（１０）basic クラス

クラス・メソッド		
1	:adding	指定クラスの指定スロットに値を追加する（リスト　プールへの追加）
2	:adding_with_index	指定クラスの指定スロットにインデックスを付けて値を追加する。（インデックス付きプールへの追加）
3	:setting	指定クラスの指定スロットに値（リスト構造）をセットする。（リスト　プールへの追加）
4	:setting2	指定クラスの指定スロットに値をセットする。（値をヒープベクタへ変換する。）
5	:getting	指定クラスの指定スロットの値を得る。（:adding,setting によりセットした値を取得）
6	:getting2	指定クラスの指定スロットの値を得る。（:setting2 によりセットした値を取得）
7	:getting_with_index	指定クラスの指定スロットに指定インデックスを持つ値を取得する。（:adding_with_indexによりセットした値を取得する。）
8	:remove_with_index	指定クラスの指定スロットに指定インデックスを持つ値を削除する。
9	:incf	指定クラスの指定スロットの値を１増加させる。
10	:length	指定リストの長さを求める。
11	:append	２つのリストのappendを求める。
12	:member	指定要素が指定リストに入っているか。
13	:add_last	指定要素を指定リストの最高尾に追加する

14	:delete	指定要素を指定リストから削除する。
15	:pushnew	指定要素が指定リストになければ追加する
16	:ascending_ordered_pushnew	指定要素を昇順にソートされたリストに追加する。
17	:reverse	指定リストの要素の順番を逆にする。
18	:mapcan	リストを各要素とするリストの全要素からその各要素を取り出しリストをつくる。
19	:rest	指定リストから指定要素以降のリストを求める。
20	:ordered_insert	指定要素をｲﾝﾃﾞｯｸｽの昇順によりソートされたリストに追加する。
21	:get_obj	オブジェクト名（ユーザが命名）からオブジェクトを求める。
22	:gensym	新しいアトムを生成する。

※ basic クラスのメソッド（クラスメソッド及びインスタンスメソッド）は、他の全クラスに継承される。

※ 以下にあげる basic クラスのインスタンスメソッドは、同一名をもつクラスメソッドと同様な機能を持つ。

① :adding ② :adding_with_index ③ :setting ④ :setting2
⑤ :getting ⑥ :getting2 ⑦ :getting_with_index
⑧ :remove_with_index ⑨ :incf

（１１）user クラス

クラス・メソッド		
1	:assert_fact	前提（premise）データを宣言する。
2	:assert_assumption	仮説（assumption）データを宣言する。
3	:assert_justification	理由づけ（justification）を宣言する。
4	:nogood	矛盾するデータの組合せを宣言する。
5	:datum_to_node	指定データに対応するNODEオブジェクトを

		取得する。
6	:datum_to_node_name	指定データに対応するNODEオブジェクトの名前を取得する。
7	:get_solution1	ｵﾙﾀﾅﾃﾌﾞな集合（リスト）を要素とするリストを与え可能な組合せ（環境）を求める（横型探索）
8	:get_solution2	ｵﾙﾀﾅﾃﾌﾞな集合（リスト）を要素とするリストを与え可能な組合せ（環境）を求める（縦型探索）
9	:status	指定データの状態（IN or OUT）を求める
10	:implied_p	指定データが指定環境に含まれるかチェックする。
11	:consistency_p	指定データが指定環境と矛盾しないかチェックする。
12	:get_justifications	指定データの理由づけを求める。
13	:get_assumption	指定データが仮説であれば仮説名を求める
14	:init	ATMSを初期化する。

※user クラスにインスタンス・メソッドは存在しない。

３．４　試作ＡＴＭＳの改善

２．１のＡＴＭＳの課題で上げた各項目が、本インプリメントの課題でもある。ここでは、インプリメントの技法上の問題に限定する。

（１）処理速度の向上

本インプリメントでは、環境（enviornment）が持つ仮説（assumptions）を直接リストとして保持している。そのため、２つの環境の組合せや包含関係のチェックなどは全てリスト操作による。したがって、計算速度は、組み合わせる環境の数及び各環境が持つ仮説の数に大きく依存する。

そこで、論理演算程度で環境の組合せや包含関係をチェックできるビット・ベクタによる高速化を考える。

［ビット・ベクタ計算による高速化］

仮説空間がｎ個（有限個）の仮説p_1，p_2，p_3，，，p_nを持つとき、このｎ個の仮説をｎ個のビット列b_1，b_2，b_3，，，b_nと対応させる。各ビットは、対応する仮説が有るとき１、ないとき０の値をとる。

ここで、E_1，E_2は、それぞれｎ個の仮説の中から任意個選んで作った環境とする。また、E_1及びE_2に対応するビット列をB_1及びB_2とすると、次のように簡単な論理演算で環境の結合せ及び包含関係のチェックが計算できる。

(a) 環境E_1とE_2の組み合わせ

B_1 or B_2

(b) 包含関係のチェック（E_1がE_2を含む）

B_1 and B_2 ＝ B_2

（２）ｎｏｇｏｏｄの非単調性の実現

本システムでは矛盾を起こす環境は、nogoodテーブル に登録される。現在、nogoodの宣言は断定であり、将来取り消すことはできない。したがって、nogoodテーブル は単調増加する。

nogoodの宣言の取り消し（非単調性）を可能とするためには、次のことを可能としなければならない。

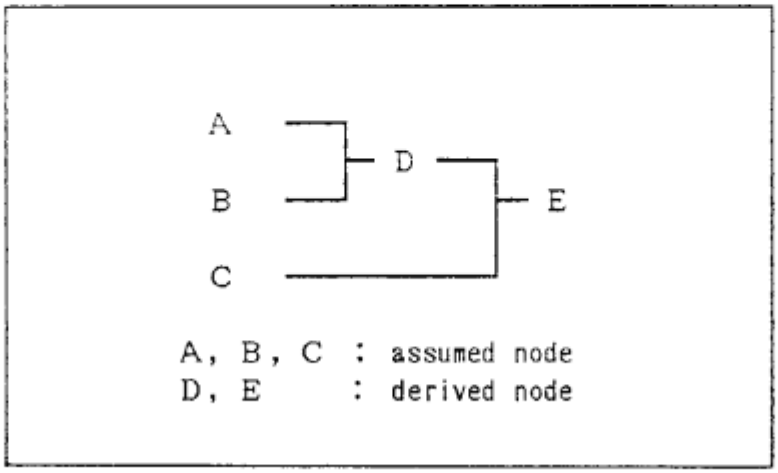
①nogoodの宣言によってラベルから取り消された、環境を復活させる。その際に、nogoodで宣言された環境だけでなく、その環境を包含する上位集合も復活させる必要がある。（理由づけ(justification)をたどりながらラベルを再度計算し直す。）

②nogoodテーブルは矛盾する環境をminimalで管理しているために、矛盾を起こす環境を nogoodテーブル へ登録するときにその環境を含む上位集合はテーブルから除かれる。したがって、nogoodの宣言を取り消す場合には、宣言によって取り除かれた上位集合を復活させる必要がある。(a)とは異なり再計算によって求めることはできないので、予め環境を消去するとき記憶しておく必要がある。

４．ＡＴＭＳを用いた問題解決例

４．１ 簡単なラベル更新例

次の図で示されるようなノード間の関係を生成するまでに、ＡＴＭＳの内部表現がどのように変化していくか順に示す。



（１）データ a，b，c，d，e に対応するノード A，B，C，D，E を生成する。

```
:make_node(#node,a,A).
:make_node(#node,b,B).
        :
        :
:make_node(#node,e,E).
```

ノードＡのＡＴＭＳ内部表現

```
node_name  : node2
index      : 2
datum      : a
status     : out
label      : []
c_justs    : []
a_justs    : []
rules      : nil
plist      : []
```

（２）ノード Ａ，Ｂ，Ｃ が仮説であることを宣言する。

```
:assume_node(#interface,A,ja).
:assume_node(#interface,B,jb).
:assume_node(#interface,C,jc).
```

ノードＡの内部表現

```
node_name  : node2
index      : 2
datum      : a
status     : in
label      : [env2]
c_justs    : [just1]
a_justs    : []
rules      : nil
plist      : [ass1]
```

環境ｅｎｖ２の内部表現

```
environment   : env2
index         : 2
ass_size      : 1
assumptions   : [ass1]
nodes         : [node2]
contradictory : nil
```

仮説ａｓｓ１の内部表現

```
assumption   : ass1
index        : 1
node         : node2
environments : [env2]
```

理由づけｊｕｓｔ１の内部表現

```
justification : just1
index         : 1
type          : assume-node
antecedents   : [ass1]
consequence   : node2
```

（３）①ノードＤ（データｄ）は、ノードＡ（データａ）及びノードＢ（データｂ）から得られたことを宣言する。

```
:make_justification(#justification,given,D,[A,B],jd).
```

②ノードＥ（データｅ）は、ノードＣ（データｃ）及びノードＤ（データｄ）から得られたことを宣言する。

```
:make_justification(#justification,given,E,[C,D],je).
```

ノードＤの内部表現

```
node_name  : node5
index      : 5
datum      : d
status     : in
label      : [env5]
c_justs    : [just4]
a_justs    : [just5]
rules      : nil
plist      : []
```

ノードＥの内部表現

```
node_name  : node6
index      : 6
datum      : e
status     : in
label      : [env6]
c_justs    : [just5]
a_justs    : []
rules      : nil
plist      : []
```

環境ｅｎｖ５の内部表現

```
environment   : env5
index         : 5
ass_size      : 2
assumptions   : [ass1,ass2]
nodes         : [node5]
contradictory : nil
```

環境ｅｎｖ６の内部表現

```
environment   : env6
index         : 6
ass_size      : 3
assumptions   : [ass1,ass2,ass3]
nodes         : [node6]
contradictory : nil
```

理由づけｊｕｓｔ４の内部表現

```
justification : just4
index         : 4
type          : derived-node
antecedents   : [node2,node3]
consequence   : node5
```

理由づけｊｕｓｔ５の内部表現

```
justification : just5
index         : 5
type          : derived-node
antecedents   : [node4,node5]
consequence   : node6
```

（４）ノードＢとＣの組合せは矛盾であることを宣言する。

```
:nogood_nodes(#interface,[B,C],jnogood).
```

ノードＤの内部表現	
node_name	: node5
index	: 5
datum	: d
status	: in
label	: [env5]
c_justs	: [just4]
a_justs	: [just5]
rules	: nil
plist	: []

ノードＥの内部表現	
node_name	: node6
index	: 6
datum	: e
status	: out
label	: []
c_justs	: [just5]
a_justs	: []
rules	: nil
plist	: []

理由づけｊｕｓｔ６（jnogood）の内部表現	
justification	: just6
index	: 6
type	: nogood
antecedents	: [node3,node4]
consequence	: node1

ノードｎｏｄｅ１の内部表現	
node_name	: node1
index	: 1
datum	: contradiction
status	: out
label	: []
c_justs	: [just6]
a_justs	: []
rules	: nil
plist	: []

４．２ Ｎ－ＱＵＥＥＮ問題

N-QUEEN問題は、ＮＸＮのチェスの盤上にＮ個のクイーンがお互いに取り合わないように配置するゲームである。

ここでは、ＡＴＭＳを使って問題解決する手法と解を得るまでに生成した各クラスのオブジェクト数を示す。なお、N-QUEENの問題解決の戦略は、盤の対称性を利用するなどいくつか考えられるが、ここではＡＴＭＳを用いた問題解決の一例を示すことに焦点を当てて以下の手順にしたがって解く。

（１）ＡＴＭＳを用いた問題解決手順

［手順１］盤上の各目にクイーンを置く仮説を立てる。
（盤のｉ行ｊ列目にクイーンを置くという仮説をa_{ij}で表す。）

１行目　：　a_{11}, a_{12}, a_{13}, ・・・a_{1n}
２行目　：　a_{21}, a_{22}, a_{23}, ・・・a_{2n}
３行目　：　a_{31}, a_{32}, a_{33}, ・・・a_{3n}
：
：
ｎ行目　：　a_{n1}, a_{n2}, a_{n3}, ・・・a_{nn}

［手順２］お互いに干渉するクイーンの位置関係（ｎｏｇｏｏｄ）を宣言する。

ｎｏｇｏｏｄ　：　（a_{ij}, a_{kl}）

ただし、　ｉ＝ｋ
または　ｊ＝ｌ
または　｜ｉ－ｋ｜＝｜ｊ－ｌ｜
の関係が成り立つとき。

［手順３］nogoodに触れないように仮説（置く位置）を１行目から１つ選択、２行目から１つ選択し新しい環境（environment）を作る。できた環境が矛盾（干渉している）がなければ、その環境に順次ｎ行目まで仮説を追加していく。ｎ行目まで仮説を追加した環境に矛盾がなければ、これが１つの解となる。
全解をもとめる方法として、上のようにある行の仮説を１つ選択したら次の行へと仮説を１つ１つｎ行まで組み合わせる方法（縦型探索）と、まず１行目と２行目の組合せ（環境）を全て求め、それらと３行目の仮説の組合わせるというように１行１行全仮説を組合せる方法（横型探索）がある。しかし、縦型も横型も基本的に大差はない。

（２）実行結果

８－ＱＵＥＥＮの実行結果を以下に示す。

	縦型探索／横型探索
仮説数 (assumption)	６４
nogood 数	７２８
生成環境数 (environment)	１６４８３
無矛盾な 環境数（解）	９２

５．本ＡＴＭＳの使用てびき

５．１　ＡＴＭＳ ファイル一覧

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 load.com　⑭ save.com

５．２　ＡＴＭＳ 立ち上げ手順

ＳＩＭＰＯＳの基本的操作は省略する。

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 file-load を選択する。

⑥ マウスで do it を選択する。

２．debugger 内でのＡＴＭＳ操作

① システム・メニューで debugger をオープンする。

② ATMS を初期化する。　（ :init(#user). を実行する。）

５．３　ＡＴＭＳのユーザ・インタフェースについて

通常、ユーザ（問題解決器）は 、図１．１が示すようにデータ及び理由づけをＡＴＭＳに与える。したがって、データに対応するノードの生成やJUSTIFICATION（ＡＴＭＳが生成するオブジェクト）の生成などは、ＡＴＭＳ内で管理すべきもので直接ユーザが操作する必要はない。このような視点に立って、本ＡＴＭＳのインタフェースを構築している。

実際にユーザ（問題解決器）が ＡＴＭＳに指示を与えるためには、次のような user クラス のメソッドを用いればよい。（詳細は、第３章を参照のこと）

① :assert_fact	② :assert_assumption	③ :assert_justification
④ :nogood	⑤ :datum_to_node	⑥ :datum_to_node_name
⑦ :get_solution1	⑧ :get_solution2	⑨ :status
⑩ :implied_p	⑪ :consistency_p	⑫ :get_justifications
⑬ :get_assumption	⑭ :init	

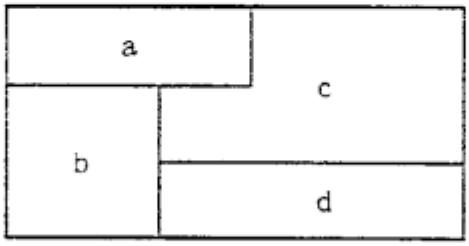
またＡＴＭＳが内部で管理している情報を見るためには、print クラス の各メソッドを用いればよい。実際にプリントのメソッドを起動するには、オブジェクト（NODE, JUSTIFICATION,ASSUMPTION,ENVIRONMENT）を指定する必要がある。オブジェクトの指定の仕方としては、次のいずれでもよい。

(1) ＡＴＭＳに与えたデータ名（NODE インスタンスの場合）
　(ex.　assert_assumption(#user,a). を実行した　a)
(2) ＡＴＭＳが生成オブジェクトに命名した名前
　(ex.　node3)
(3) オブジェクト自身
　(ex.　$node　をbindされた変数　Node)

５．４　ＡＴＭＳの例題プログラム

例題１

次のような領域（ａ、ｂ、ｃ、ｄ）に色（赤、青、緑）を塗りたい。ただし、隣合わせの領域に同じ色を塗ってはいけない。何通りの色の塗り方があるか？



プログラム

```
example1(Class,Solutions) :-

  :assert_assumption(#user,a(red)),
  :assert_assumption(#user,a(blue)),
  :assert_assumption(#user,a(green)),
  :assert_assumption(#user,b(red)),
  :assert_assumption(#user,b(blue)),
  :assert_assumption(#user,b(green)),
  :assert_assumption(#user,c(red)),
  :assert_assumption(#user,c(blue)),
  :assert_assumption(#user,c(green)),
  :assert_assumption(#user,d(red)),
  :assert_assumption(#user,d(blue)),
  :assert_assumption(#user,d(green)),
  :nogood(#user,[a(red),b(red)]),
  :nogood(#user,[a(blue),b(blue)]),
  :nogood(#user,[a(green),b(green)]),
  :nogood(#user,[a(red),c(red)]),
  :nogood(#user,[a(blue),c(blue)]),
  :nogood(#user,[a(green),c(green)]),
  :nogood(#user,[b(red),c(red)]),
  :nogood(#user,[b(blue),c(blue)]),
  :nogood(#user,[b(green),c(green)]),
```

```
:nogood(#user,[b(red),d(red)]),
:nogood(#user,[b(blue),d(blue)]),
:nogood(#user,[b(green),d(green)]),
:nogood(#user,[c(red),d(red])),
:nogood(#user,[c(blue),d(blue)]),
:nogood(#user,[c(green),d(green)]),

:get_solutioni(#user,[[a(red),a(blue),a(green)],
                      [b(red),b(blue),b(green)],
                      [c(red),c(blue),c(green)],
                      [d(red),d(blue),d(green)]],
               Solutions) ;
```

実　行　例

```
?- :example1(Class,Solutions).

 Solutions= [env33,env39,env48,env53,env62,env68]

?- :env_datum(#print,env33).
```

ＡＴＭＳ内部情報

```
--------------------------------------
  ENV が含む ASSUMPTION-DATUM の表示
--------------------------------------
  environment     : env33
  contradictory   : nil
  datum           : a(red)
  datum           : b(blue)
  datum           : c(green)
  datum           : d(red)
```

例題２

男子（ａ夫、ｂ夫、ｃ夫、ｄ夫）及び女子（ｅ子、ｆ子、ｇ子、ｈ子）からカップル（全員）をつくりたい。何通りの作り方があるか？
ただし、次の仲がよいペアを考慮すること。
（ａ夫、ｅ子）、（ａ夫、ｇ子）、（ｂ夫、ｅ子）、（ｂ夫、ｈ子）、
（ｃ夫、ｇ子）、（ｃ夫、ｈ子）、（ｄ夫、ｆ子）、（ｄ夫、ｈ子）

プログラム

```
example2(Class,Solutions) :-

   :assert_assumption(#user,pair(a,e)),
   :assert_assumption(#user,pair(a,g)),
   :assert_assumption(#user,pair(b,e)),
   :assert_assumption(#user,pair(b,h)),
   :assert_assumption(#user,pair(c,h)),
   :assert_assumption(#user,pair(c,g)),
   :assert_assumption(#user,pair(d,f)),
   :assert_assumption(#user,pair(d,h)),

   :nogood(#user,[pair(a,e),pair(b,e)]),
   :nogood(#user,[pair(b,h),pair(c,h)]),
   :nogood(#user,[pair(a,g),pair(c,g)]),
   :nogood(#user,[pair(b,h),pair(d,h)]),
   :nogood(#user,[pair(c,h),pair(d,h)]),

   :get_solution1(#user,[[pair(a,e),pair(a,g)],
                         [pair(b,e),pair(b,h)],
                         [pair(c,g),pair(c,h)],
                         [pair(d,f),pair(d,h)]],
                  Solutions);
```

実　行　例

```
?- :example2(Class,Solutions).

 Solutions= [env17,env23]

?- :env_datum(#print,env17).
```

ＡＴＭＳ内部情報

```
-------------------------------------
  ENV が含む ASSUMPTION-DATUM の表示
-------------------------------------
  environment     : env17
  contradictory   : nil
  datum           : pair(a,e)
  datum           : pair(b,h)
  datum           : pair(c,g)
  datum           : pair(d,f)
```

［ 謝　辞 ］

本研究の機会を与えて下さったＩＣＯＴ研究所 淵一博所長、第５研究室 藤井裕一室長ならびにＮＴＴデータ通信事業本部 岩下安男 ＡＩ担当部長に深く感謝致します。
またＰＳＩ－Ⅱ及びＥＳＰプログラミングに関して御指導頂いた第５研究室 椿　和弘氏に感謝いたします。最後に本研究に御助力頂きました滝 寛和 氏をはじめとするＩＣＯＴ第５研究室の皆様に感謝致します。

［参考文献］

[de Kleer 86a] de Kleer, J.,An Assumption-based TMS,
Artificial Intelligence 28 (1986) 163-196.

[de Kleer 86b] de Kleer, J. and Williams, B.C.,
Back to Backtracking : Controlling the ATMS,
Proc. AAAI-86 (1986) 910-917.

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

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

[Williams 85] Williams, C.,Managing Search in a Knowledge-based System,
Inference Corporation, unpublished (1985).

[飯島 87] 飯島他，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資料，SIG-FAI-8701-2 (1987).

[井上 87] 井上，仮説推論による探索木の枝刈りについて，
日本ソフトウェア科学会第４回大会（1987）131-134.

[中川 88] 董，中川，不確実な知識におけるＡＴＭＳ，
人工知能学会誌 3 (1988) 86-93.