ICOT Technical Report: TR-667

TR-667

KL1による並列ATMS

中島　誠 (JIPDEC)
太田　好彦、井上　克巳

July, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

KL1 による並列 *ATMS*

中島 誠† 太田 好彦‡ 井上 克巳‡

†(財) 日本情報処理開発協会

〒105 東京都港区芝公園 3-5-8 機械振興会館内

e-mail:j-nakash@icot.or.jp

‡(財) 新世代コンピュータ技術開発機構

〒108 東京都港区三田 1-4-28 三田国際ビル

e-mail:ohta@icot.or.jp, inoue@icot.or.jp

目次

第 1 章

はじめに

仮説推論は，常に正しい知識である事実の集合と，常に正しいとは限らない知識である仮説の集合とに基づき，事実の集合と矛盾しない仮説部分集合を加えて結論を導く推論形態であり，知識システムを構築するための重要な要素技術である [Inoue 88]. しかしながら，推論過程で加える仮説部分集合の無矛盾性を検査せねばならず，推論速度の点で問題がある.

従来提案されている ATMS (Assumption-based Truth Maintenance System) [deKleer 86] は，命題論理の事実集合と仮説集合を基に，データ (命題) の真偽を管理するシステムである. ATMS は新たなデータが付け加わった時に既存のデータとの整合性を保ち，矛盾が発見された場合に仮説集合を見直してその無矛盾性を管理することを主な機能とし，これをデータ構造および，アルゴリズムの工夫により効率的に行なうことを実現している. しかしながら，不完全な知識ベースを用いる問題解決には指数オーダの計算量が必要であると考えられることから，より一層の高速化が望まれている. [Dixon 88] では，コネクションマシン上で ATMS の並列化の試みが行なわれているが，入力される知識 (データ) の数に対して，指数オーダのプロセッサが必要になる. また，疎結合並列計算機上での高速化についてもいくつかの試みが行なわれている [Harada 89] [和田 90].

これら従前の試みの中にあって本システムの特徴は，入力されるデータの数だけのプロセス[1]を生成する並列アルゴリズムを採用している点にある. この方式により，入力データの数に対して生成するプロセス数の爆発が防げる. さらに，これらの生成されたプロセスを複数のプロセッサへ割り当てることが，データの分配の問題に帰着されるため，比較的簡単に行なえると考えられる.

以下 2では，ATMS 並列化の実現方法とその性能について述べ，3では並列 ATMS システムの仕様および利用例を示す. また KL1 によるソースコードを 4に示す.

[1]本稿では，プロセスという用語を解くべき問題中で並列に処理されるべきサブタスクの単位とする. これは KL1 プログラミングでいうところのプロセスよりも広い意味で用いている.

第 2 章

並列 ATMS

本章では, 基本的 ATMS の概要を述べ, つづいて, それをもとに並列化の実現方法を示し, さらに具体例を用いた計測値をあげて, 考察を加える.

2.1 ATMS

ここでは基本的な ATMS の概要を示す. ATMS の並列化の具体的な実現方法については章を改めて論じる.

ATMS[deKleer 86] は, 仮説集合と命題論理のホーン節 (理由付け (*Justification*) と呼ぶ) の集合を入力とし, アトム (データ (*Datum*) と呼ぶ) の真偽の状態 (あるいは信念の状態) を管理し, 出力する.

理由付けとして入力されるホーン節は次に示すいずれかの形式である.

$$a_1, \cdots, a_n \Rightarrow b.$$
$$a_1, \cdots, a_n \Rightarrow \bot.$$

ここに, $a_i (i = 1, \cdots, n; n \geq 0)$ 及び b は命題論理のアトム, 記号 $\bot$ は偽 (false) を表し, 記号 $\Rightarrow$ の左の各アトムを前件, 右を後件と呼ぶ.

ATMS 内部では各データは以下に示すデータ構造 (これをノードと呼ぶ) で表現され, その信念の状態が管理されている[1].

γ_{Datum} : < データ, ラベル, 理由付け >.

現時点までに入力された仮説の集合の部分集合を環境と呼ぶ. また, 現時点まで入力された全ての理由付けの集合を J とすると, J とある環境から $\bot$(偽;false) が導かれるときその環境を矛盾環境 (*Nogood*) と呼び, J とある環境から $\bot$ が導けないときその環境を無矛盾 (consistent) な環境と呼ぶ. あるノードが信じられている (IN 状態と呼ぶ) とは, ある無矛盾な環境 E があって J と E からノードのデータが導かれることである. IN 状態のノードは, 後から追加される知識によってそのノードが信じられていない状態 (OUT 状態と呼ぶ) になる可能性を持っている. ノードのラベルはそのノードが信じられている極小の環境の集合である. ATMS は, 入力された理由付けをその後件に対応するノード (後件ノードと呼ぶ) の理由付けの項に, 前件に対応するノード (前件ノードと呼ぶ) の集合を付加して記憶している.

ATMS に新たな理由付けが与えられるとそれを基に, ラベルの更新が行なわれる. 以下は a_i (i=1,. . .,n:n≥0) を前件とする理由付けが与えられたときのラベルの更新手順を表したものである. また, ラベル計算の簡単な例を付録 A に示す.

[ラベル更新手順]

1. 後件ノードのラベルを L_{old}, 前件ノード a_i のラベルを L_i と示す. L_i に含まれる各環境の集合に関して直積をとり, L' とする.

$$L' = \{x | x = \bigcup_i E_i, E_i \in L_i\}$$

L' は, 環境をビットベクタ[2]で表現しておくとビット演算 or により計算できる.

[1] データ X が真であるという仮説を表すノードを特に仮説ノード (assumption node) と呼ぶ.

[2] 仮説集合 H がその要素として n 個の仮説 $A_1, A_2, \cdots, A_n$ を含む時, この n 個の仮説を n 個のビット列 $b_1, b_2, \cdots, b_n$ と対応させた表現. 各ビットは対応する仮説がある時 1, 無い時 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.

2. $L_{old} \bigcup L'$ から矛盾環境を取り除き，さらにその集合の極小の要素 (他の要素を包含しない要素) の集合を L_{new} とする．

3. もし L_{old}=L_{new} ならばこのノードに関するラベル計算は終了．

4. もし後件が偽であるなら，L_{new} に含まれるすべての環境について，矛盾環境として登録し，自分以外のノードのすべてのラベルからそれを包含する環境を除去し終了する．

5. もし後件が偽でないなら L_{new} を後件ノードの新しいラベルとする．さらに，このノードを前件ノードとする理由付けに関して 1. から実行する．

2.2 並列 ATMS の実現方法

本章では ATMS の並列化の実現方法について論じる．

2.2.1 システム構成

以下に本システムの基本構成図を示す．

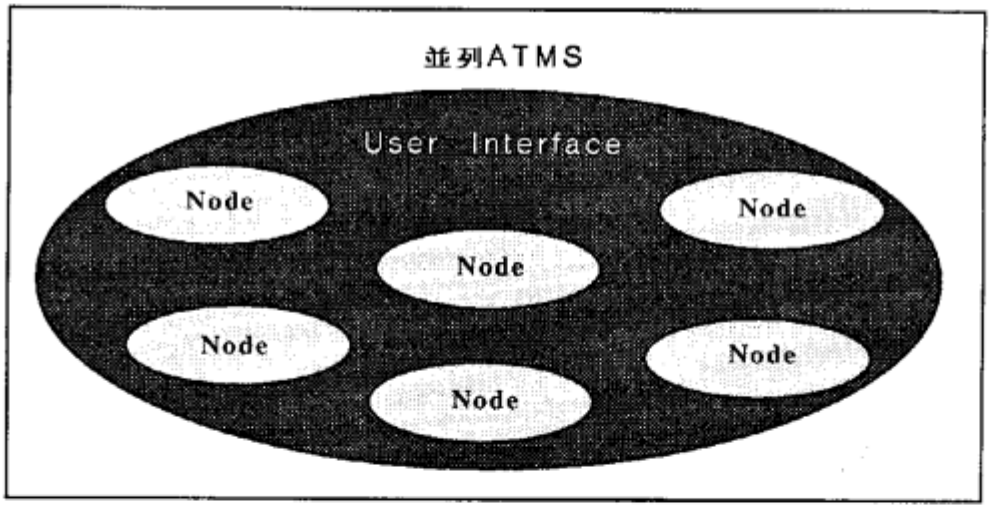

図 2.1: 並列 ATMS 基本構成図

1. ユーザ・インタフェース (user interface)

   各ノードプロセスに対するストリームの管理を行ない，外部からのメッセージを解析して，ノードプロセスに対して対応するメッセージを送る．ノードプロセスの管理の有無によってユーザインタフェースは 2 つのレベルに分けることができる (詳しくは 3.1,3.2を参照のこと)．

2. ノード・プロセス (node)

   ATMS ノードを表す．ノードプロセスは外部からのメッセージに基づいて生成され，ATMS ノードに関する以下のような処理を行なう．

   - 前件ノードからのラベルの収集
   - 理由付けの追加にともなうラベルの更新

- 理由付けを介した後件ノードへのラベルの伝播
- 矛盾環境の削除

各ノードプロセスはノードの信念の状態を保持するだけでなく，外部あるいは内部の他ノードからのメッセージを受けて自らその状態を更新していく．そのために必要な情報は，理由付けによって関連づけられた他ノードとの間で情報の送受を行なうことで収集する．本システムの特徴としては，理由付けの追加に関して必要なラベルを収集するようなマネージャは存在せず，上記のように各ノードが各々必要な処理を行なう点にある．これは，ラベルの収集を行なうマネージャへの通信の集中が並列実行の妨げとなることを配慮した結果である．以下，並列 ATMS のデータ構造および機能について述べる．

2.2.2 データ構造

並列 ATMS では，外部から与えられたデータに対応して以下のデータ構造を有するプロセスを生成し，これによってノードを表現する:

γ_{Datum}: <データ, ラベル, *Justifications*, *Consequents*, *Attention*>.

Justifications: このデータを後件とする理由付けの前件ノードへメッセージを送信する出力ストリーム (これを $J-$ ストリームと呼ぶ) の集まり．

Consequents: このデータを前件に持つ理由付けの後件ノードへメッセージを送信する出力ストリーム (これを $C-$ ストリームと呼ぶ) の集まり．

Attention: 偽に対応するノードから送られる矛盾環境の削除命令を受信する入力ストリーム (これを $A-$ ストリームと呼ぶ)．

データおよびラベルは 2 章において定義されたものと同じである．$J-$ ストリームは理由付けに基づいてノードのラベルを計算する際に，その理由付けの前件ノードからラベルを収集するメッセージを送信するためのストリームである．さらに，$C-$ ストリームはラベルの更新が起こった場合に，後件ノードに対して，そのラベルの更新命令を伝える．*Justifications*, *Consequents* は，ノードを表すプロセスへのストリームの集まりとなっている．

また，偽に対応するノードは，極小の矛盾環境 (他の矛盾環境を包含しない矛盾環境，以下単に矛盾環境と呼ぶ) の集合 (*Nogood* データ・ベースと呼ぶ) をそのラベルとして保持し，自身以外の全てのノードへ $A-$ ストリームを通じて矛盾環境を伝達する．他のノードの *Attention* はその矛盾環境を受信するためのストリームである．

本並列 ATMS では，基本的にノードのみをプロセスとして表現しているので入力されたデータの数のプロセスが生成される．

2.2.3 並列化機能

本並列 ATMS では以下のような処理を並列に実行することが可能である．

1. **外部から ATMS に与えられる複数の命令の並列実行**

   ここに，命令とは，以下の 3 項目である．

   - 仮説の追加
   - 理由付けの追加
   - データの信念の状態の問合せ

   命令の並列実行は，各々のデータを管理するプロセスがそれぞれ独立して実行されることによって実現された機能である．以下に示すのは本並列 ATMS において実際に用いる仮説ノードの生成命令の例である．

create-a(a,As)
create-a(b,Bs)
create-a(c,Cs)

As,Bs,Cs はそれぞれデータ a,b,c が真であるという仮説ノードを表すプロセスへのストリームとなっている．各々の仮説ノードは互いに独立しており，同時生成が可能である．また，それぞれのノードに対する理由付けの追加，信念の状態の問い合わせは，各ストリーム (As,Bs,Cs) に対して対応する命令を送ることで実行される．したがって，複数のノードのそれぞれに対する理由付けの追加も並列に実行できる．さらに仮説の追加と理由付けの追加が混在するような場合も並列に実行できる．ただしデータの問い合わせについては，それ以前に ATMS に与えられた命令が全て処理された後行なわれる．これにより，問い合わせを行なった時点でのラベルの正当性が保証される．

2. 理由付けに伴うラベル更新の並列実行

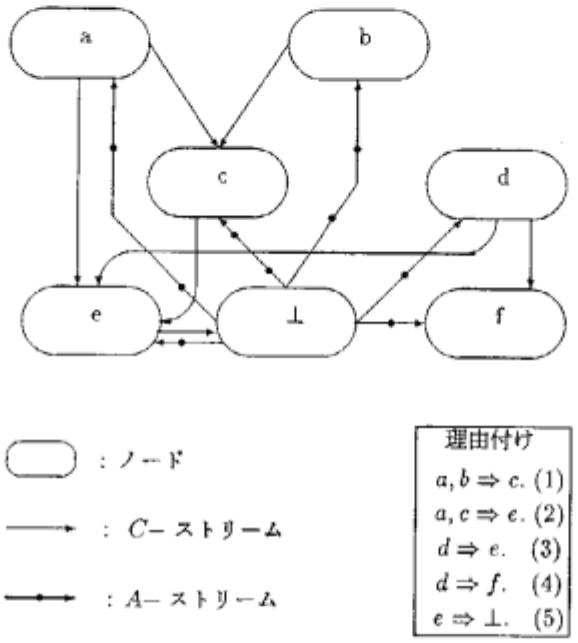

図 2.2: 本並列 $ATMS$ によって生成されるプロセス・ネットワークの例

ここで，ラベル更新とは具体的に以下の処理を行なうことである．

(a) 各前件ノードのラベルの収集
与えられた理由付けの前件ノードから，それらのラベルを集める．

(b) 後件ノードのラベルの更新
(a) で集めたラベルを基に後件ノードのラベルを更新する．

(c) 更新ラベルの伝播
このノードを理由付けの前件に持つ全てのノードに対して，更新されたラベルを伝播させる．

(d) 矛盾する環境の削除
もし偽に対応するノードのラベルが更新された場合には，新たに追加された矛盾環境を自分以外の全てのノードのラベルから削除する．

このうち，並列に実行できるのは (a) と (c) および (d) である．図 2.2に示した理由付けとそれによって構築されるプロセスのネットワークを例にラベル更新の並列実行を考える．

図 2.2のプロセスがそれぞれ異なったプロセッサに割り当てられた時，理由付け (1,3,4) のようにお互いの後件ノードをそれぞれの前件に持たない場合は，各々の理由付けに基づくラベル計算は独立して並列に実行することができる．また，理由付け (1,2) と (2,3,5) とによって生じるラベルの伝播処理もパイプライン処理によって並列に実行することが可能となっている．

以下に並列ATMSにおけるラベルの更新手順を示す.

[並列ラベル更新手順]
ATMSにある理由付け $a_1,\cdots,a_i,\cdots,a_n \Rightarrow x\ (1 \leq i \leq n; n \geq 0)$ が与えられた時, その理由付けの後件ノードを表すプロセス (γ_x と示す) において以下の手続きを実行する.

1. γ_x は, その前件ノード γ_{a_i} への $J-$ ストリームを有しているので, γ_{a_i} に対してそのラベル L_i の問い合わせを発する. これを受信した γ_{a_i} は自身のラベルを返す. γ_x は L_i を得ないと次の処理ができないので, この問い合わせ処理のプライオリティは他の処理よりも高くしている[3]. これにより, 個々に独立したノードから, ほとんど同時にラベルの収集を行なうことができる.

2. γ_x は $L' = \{e|\ e = \cup_{i=1}^{n} E_i,\ E_i \in L_i\}$ を計算する. ここに, 環境のビットベクタ表現を用いているため, 集合演算 $\cup$ はビット演算 or でよい. また, γ_x はすべての前件ノード のラベルが揃わなくても, 少なくとも 2 つのラベルが送られてくれば計算を開始することができる.

3. 現在の自身のラベルの内容を L として, $x \neq \bot$ なら, $L \cup L'$ から矛盾環境を取り除いた集合の極小の要素の集合を求めて L'' とする. $x = \bot$ なら, 単に $L \cup L'$ の極小の要素の集合を求めて L'' とする.

4. $L = L''$ ならばこの処理を終わる. そうでなければ L'' を自身のラベルとする.

5. $x \neq \bot$ であるならば, この x を理由付けの前件に持つ全てのノードに対して $C-$ ストリームを介してラベルの更新命令を送る. それを受けたノードは, 自身の *Justifications* に格納してある理由付けに基づいて 1. より計算を始める. $x = \bot$ ならば, 新しく追加された矛盾環境を $A-$ ストリームに送信する. $A-$ ストリームを介して矛盾環境を受信したプロセスは, 自分自身のラベルの要素で矛盾環境を含むものを全て削除する.

後件が相異なる複数の理由付けが与えられた時には, 各々並列にそれぞれの後件ノードに対応するプロセスにおいて, 上記の手順を並列に実行することができる.

2.3 性能評価

本節では, 並列ATMSの性能評価を行なう.

2.3.1 実測結果

並列ATMSの実行性能を評価するためにチェス盤上に N 個のクィーンが衝突しないように配置するクィーン問題を取り上げた. 図2.3は $N = 8$ の場合のATMSに与える仮説集合, 矛盾環境および8個のクィーン (以下Qと示す) の配置に関する理由付けである.

ここで, $q(i,j)(1, \leq i,j \leq 8)$ は, チェス盤上 i 行 j 列にクィーンを置くという仮説を意味する. また矛盾環境 {q(i,j), q(m,n)} $(1 \leq i,j,m,n \leq 8)$ は以下の条件を満たすものとした.

- $i = m$ あるいは $j = n$
- $|i-j| = |m-n|$

まず, 8×8 の盤上の 1 つの桝目に 1 つの Q をおくことを仮説とする (*Assumptions*). 次に二つの Q が互いにとり合う位置にある組合せを矛盾環境とする (*Nogoods*). また, 6 個の Q に 1 から 6 行まで行ごとに理由付けを導入する (*6 Columns Justifications*). さらに, この 6 個の Q と 7,8 行における残り 2 つの Q の配置に対応する理由付けを追加する (*8 Queens Justifications*). 8 クィーンの解は *queen1* から *queen64* までのラベルの和集合で示される.

ここで生成されるノードは, $q(i,j)\ (1 \leq i,j \leq 8)$, *pos* $k\ (1 \leq k \leq 6)$, *queen* $l\ (l \leq l \leq 64)$, および $\bot$ であるから, その数は,

$$8 \times 8 + 6 + 64 + 1 = 135$$

である. このノードを以下の条件で各プロセッサに静的に割り当てる. M は使用するプロセッサ数を表し, PE は各ノードを分散させるプロセッサ番号を示す.

---

[3]KL1 実装上は, 節の優先関係指定 (alternatively) によって実現している.

$$
\begin{aligned}
&PE_{q(i,j)} = (i-1)*8+j \mod M \quad (1 \le i,j \le 8)\\
&PE_{posk} = k \mod M \quad (1 \le k \le 6)\\
&PE_{queenl} = l \mod M \quad (1 \le l \le 64)
\end{aligned}
$$

$$
(Assumptions)\left\{
\begin{array}{lllll}
q(1,1), & q(1,2), & q(1,3), & \cdots & q(1,8),\\
q(2,1), & q(2,2), & \cdots & \cdots & q(2,8),\\
q(3,1), & q(3,2), & \cdots & \cdots & \cdots\\
\cdots & \cdots & \cdots & \cdots & \cdots\\
q(8,1), & q(8,2), & q(8,3), & \cdots & q(8,8).
\end{array}
\right.
$$

$$
(Nogoods)\left\{ \{q(1,1),q(1,2)\}, \quad \{q(1,1),q(2,1)\}, \quad \cdots \quad \{q(i,j),q(m,n)\}, \quad \cdots \quad \{q(8,7),q(8,8)\}. \right.
$$

$$
(6\ Columns\ Justifications)\left\{
\begin{array}{llll}
q(1,1) \Rightarrow pos1. & q(1,2) \Rightarrow pos1. & \cdots & q(1,8) \Rightarrow pos1.\\
q(2,1) \Rightarrow pos2. & q(2,2) \Rightarrow pos2. & \cdots & q(2,8) \Rightarrow pos2.\\
 & & \cdots & \\
q(6,1) \Rightarrow pos6. & q(6,2) \Rightarrow pos6. & \cdots & q(6,8) \Rightarrow pos6.
\end{array}
\right.
$$

$$
(8\ Queens\ Justificatins)\left\{
\begin{array}{lllllllll}
pos1, & pos2, & pos3, & pos4, & pos5, & pos6, & q(7,1), & q(8,1) & \Rightarrow\ queen1.\\
pos1, & pos2, & pos3, & pos4, & pos5, & pos6, & q(7,1), & q(8,2) & \Rightarrow\ queen2.\\
 & & & & & \cdots & & & \\
pos1, & pos2, & pos3, & pos4, & pos5, & pos6, & q(7,8), & q(8,8) & \Rightarrow\ queen64.
\end{array}
\right.
$$

図 2.3: 8 クィーン問題理由付け例

M の値を 1 から 64 まで段階的に変更して, *8-queen* および, *10-queen* の実行を行なった結果を図 2.4に示す. この結果によると, *8-queen* および *10-queen* を 1 台のみのプロセッサで実行した場合に対して, 8 台の時の実行速度はそれぞれ約 6.5 倍, 7.8 倍であり, 64 台の時は同様に約 23 倍, 40 倍早くなることがわかる.

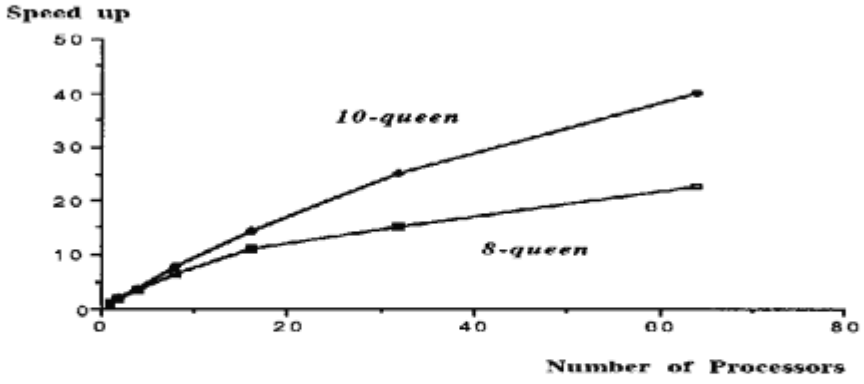

図 2.4: クィーン測定結果

2.3.2 考察

図 2.3の *8 Queens Justifications* に示した 64 個の理由付けでは, *pos1* から *pos6* までの前件ノードがすべて同じであり, *queen1* から *queen64* までのノードで重複したラベル計算をしている. これは以下の理由による.

8クィーンの解法では,8個のQについて1行から8行まで行ごとに理由付けを導入して(*pos1* から *pos8* の作成), *pos1* から *pos8* を前件に持つ後件ノード γ_{queen} を作成する方法がまず考えられる[4]. しかしながら, この方法ではノード γ_{queen} に負荷が集中することになり, マルチプロセッサによる並列実行の有効性を発揮できない. このことから明白なようにノードをプロセスとして並列実行を行なうことの限界は, ラベル計算が一つのノードへ集中する場合にある. これを回避するためには *8 Queens Justifications* のように理由付けの方法に工夫を加えて, 負荷を分散させる必要がある. その結果, 一つのノードへの理由付けを *queen1* から *queen64* に対する64の理由付けに分散して並列に処理することができ, 図2.4に示したような台数効果が得られる. ここで, 理由付けの分解による負荷の分散は, 使用できる計算機資源を考慮した適度な分散を行なわねばならないことを付記しておく. すなわち, 8個のクィーンの全ての配置を理由付けとして導入して[5], 負荷の分散を行なうことが考えられるが, この方法では 8^8 の理由付けが必要となり, メモリの不足を招いて解を求めることができなかった.

[Dixon 88] でインプリメントされたATMSは, 13クィーン問題で逐次処理の70倍の高速化がなされたと報告している. しかしながら, この基本的なアルゴリズムは, 無矛盾な環境下で信じられているデータを各プロセッサが保存するものである. したがって, 入力された仮説の数の指数オーダのプロセッサが必要になる. そこで, 極大の無矛盾な環境をプロセッサに割り当てる方法も提案されているが, これもまた, 最悪の場合において指数オーダのプロセッサが必要になる.

Multi-PSI上で稼働するATMSは, [Harada 89] と [和田 90] に述べられている. 前者のシステムでは, 5プロセッサのときに2倍の高速化が観測されているが, 1つの理由付けに伴うラベル計算の並列アルゴリズムを提案しており, 各前件データのラベルの要素数の相乗積オーダのプロセスが生成されてしまう可能性があると考えられる. また, 後者では, 各ATMSノードをプロセスで表現し, 複数の理由付けの集まりに伴うラベル計算タスクを1プロセッサに割り当てている. これによって, 16プロセッサのときに4倍の高速化がなされている. しかしながら, 事実と矛盾する環境の集合を保存している *Nogood* データ・ベースがネットワークの端にあるプロセス構成なので, ラベルの無矛盾性を維持する計算に時間がかかるように思われる.

[Rothberg 89] および [奥乃 90] は, 共有メモリ型並列マシン上でインプリメントされたATMSについて述べている. [Rothberg 89] のATMSでは, 14PE上での8クィーン問題解決において約6.5倍の高速化が観測されている. また, [奥乃 90] のATMSでは, 4PEで3.5倍の高速化が観測されている. しかしながら, 共有メモリ型マルチ・プロセッサでは, PE数をあまり増やせない小規模並列のアーキテクチャなので, それ以上の高速化はあまり期待できないと考えられる.

2.4 まとめ

並列ATMSの特徴は, ノードをプロセスによって表すことで, その生成数を現実的な数に押えることができる点にあり, 同時にATMSの処理の基本となるラベル更新の処理をそれぞれのノードが独自に行なうことで, ノードを一括管理する方法に比べて並列化の効果を期待できることにある. しかしそのためには, 一つのノードへ負荷が集中することを避けるような理由付けの戦略が不可欠となる.

また, ATMSは推論エンジンと連結することで, 効率的な仮説推論システムを構築できることが分かっているが [太田 91], 本並列ATMSも, 適当な推論エンジンとの連結によって仮説推論システムを構築することができる. この時, 推論エンジンとの連結で注目することは, 大量の理由付けが連続してATMSに入力されることが予想されることである. したがって, 推論エンジンの側では, 探索空間の枝刈りのために, 少なくとも各理由付けの後件のラベルに無矛盾な環境があるかないかを高速に返す機能が要求される. 開発された並列ATMSは複数の理由付けに伴うラベル計算を並列に計算する機能を備えたもので, 仮説推論システムのコンポーネントとして用いる上で有利な特徴を有しているといえる.

---

[4] 図2.3の6 *Columns Justifications* に以下を追加し,

$$q(7,1) \Rightarrow pos7.\quad q(7,2) \Rightarrow pos7.\quad \cdots\quad q(7,8) \Rightarrow pos7.$$
$$q(8,1) \Rightarrow pos8.\quad q(8,2) \Rightarrow pos8.\quad \cdots\quad q(8,8) \Rightarrow pos8.$$

8 *Queens Justifications* の代わりに次の理由付けを用いる.

$$pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8 \Rightarrow queen.$$

[5] 次のような理由付けになる.

$q(1,x_1),\cdots,q(i,x_i),\cdots,q(8,x_8) \Rightarrow queen(x_1,\cdots,x_i,\cdots,x_8).\ (1 \le x_i \le 8, 1 \le i \le 8)$

第 3 章

並列 ATMS 仕様

本システムでは ATMS ノードを一つのプロセスとして実現していることから，ノードの操作を行なうためには，そのプロセスへのストリームを管理する必要がある．並列 ATMS は，このストリーム管理の方法の違いによって，2 つのレベルのユーザインタフェースを提供している．1 つはノードプロセスへのストリームの管理をシステムが自動的に行なうことで，ユーザの負担を軽くしたよりユーザフレンドリなインタフェース (以下，上層インタフェースと呼ぶ) である．もう一つは，効率的な実行を目指し，ノードプロセスの管理をユーザ自身が行なう，システムよりのインタフェース (以下，下層インタフェースと呼ぶ) である．ここでは，以上の 2 つのインタフェースについてその仕様を示し，次にその利用例を示す．

3.1 上層インタフェース仕様

以下に上層インタフェースの生成法と，そのメッセージプロトコル[1]を示す．

生成

```
atms:go(Stream*)
```

```
atms:go(Stream*,PEsize)
```

```
atms:go(Stream*,PEsize,~Time)
```

```
atms:goF(InputFileString,PEsize,~Time)
```

並列 ATMS を生成し，そこへの上層インタフェースを介したストリームを **Stream** に返す．**PEsize** は，利用するプロセッサ総数を指定する．また，**Time** には実行時間が返される．入力メッセージをファイルから与える場合は，**InputFileString** にそのファイル名をストリングで指定する．

プロトコル

```
assumption(Datum)
```

```
assumption(Datum,PE)
```

Datum をデータとするノード γ_{Datum} がすでに存在する場合，そのノードが前提あるいは仮説ノードであった時には，なにも行なわずそれ以外の時は γ_{Datum} を仮説ノードとする．また，もし，γ_{Datum} が存在しない場合は仮説ノードとして新たに生成する．**PE** が指定された場合は，ノードプロセスをその指定されたプロセッサ上に割り当てる．**PE** が指定されていない場合はシステムが利用可能なプロセッサに順番に割り当てる．また **PE** が利用可能範囲を越えている場合は失敗する．以下同様．

---

[1] 引数の入出力関係を示すモード表記は以下の原則に従う．

- モード表示は対象となる項と，その 1 つ外側の項を具体化する者が同じかどうかを示す．
- 両者が同じ場合には特別の表示を行なわない．両者が異なる場合，対象となる項の前に "^" を付加してその旨を示す．
- 変数に PIMOS の組み込み述語 merge/2 の入力ストリームがユニファイされる場合，対象となる変数の後に "*" をつける．

`premise(Datum)`

`premise(Datum,PE)`

Datum をデータとするノード γ_{Datum} がすでに存在する場合, そのノードが前提ノードであった時には, なにも行なわずそれ以外の時は γ_{Datum} を前提ノードとする. また, もし, γ_{Datum} が存在しない場合は前提ノードとして新たに生成する.

`jassume(Datum,Antecedents)`

`jassume(Datum,Antecedents,PE)`

すでに存在するノード γ_{Datum} が前提あるいは仮説ノードであったときには何も行なわず, それ以外の時は γ_{Datum} を **Antecedents** を理由付けの前件に持つ仮説ノードとする.
γ_{Datum} が存在しない時は, **Antecedents** を理由付けの前件に持つ仮説ノードとして生成する.
ここで **Antecedents** は, 前件ノードのデータのリスト $Datum_1, Datum_2, ..., Datum_n$ である. また, γ_{Datum_1}, γ_{Datum_2}, ..., γ_{Datum_n} のいずれかが存在しない場合は, そのノードを新たに生成した後, 上記の処理を行なう.

`aj(Datum,Antecedents)`

ノード γ_{Datum} の理由付けに **Antecedents** を前件とする理由付けを追加する. ここで **Antecedents** は, 前件ノードのデータのリスト $Datum_1$, $Datum_2$, ..., $Datum_n$ のリストである.
γ_{Datum} あるいは, γ_{Datum_1},γ_{Datum_2},...γ_{Datum_n} のいずれかが存在しない場合は, そのノードの生成を行なった後, 上記処理を行なう.

`ac(Antecedents)`

矛盾環境に **Antecedents** のラベルを追加する (矛盾ノードの理由付けに **Antecedents** を前件に持つ理由付けを追加する). ここで **Antecedents** は, 前件ノードのデータ $Datum_1$, $Datum_2$, ...,$Datum_n$ のリストである.
γ_{Datum_1},γ_{Datum_2},...γ_{Datum_n} のいずれかが存在しない場合は, そのノードの生成を行なった後, 上記処理を行なう.

`show-label(Datum,^Label)`

ノード γ_{Datum} のラベルを **Label** に返す.

`show-node(Datum,^Node_data)`

ノード γ_{Datum} のノード情報 (データ, ラベルおよびそれが表す環境) を **Node_data** に返す.

`show-nogood(^Nogood)`

矛盾環境のリストを **Nogood** に返す.

3.2 下層インタフェース仕様

以下に下層インタフェースの生成法と, そのメッセージプロトコルを示す.
生成

`atms:patms_lif(Stream*)`

`atms:patms_lif(Stream*,PEsize)`

`atms:patms_lif(Stream*,PEsize,^Time)`

`atms:patms_lifF(InputFileString,PEsize,^Time)`

並列ATMSを生成し，そこへの下層インタフェースを介したストリームをStreamに返す．PEsizeは，利用するプロセッサ総数を指定する．また，Timeには実行時間が返される．入力メッセージをファイルから与える場合は，InputFileStringにそのファイル名をストリングで指定する．

プロトコル

`create-n(Datum, ^Node)`

`create-n(Datum, PE,^Node)`

Datumをデータとするノードγ_{Datum}を生成し，これへのストリームをNodeに返す．PEが指定された場合は，ノードプロセスをその指定されたプロセッサ上に割り当てる．PEが指定されない場合は，利用可能なプロセッサに自動的に順次割り当てられる．PEが利用範囲を越えた場合失敗する．以下同様．

`create-a(Datum,^Node)`

`create-a(Datum,PE,^Node)`

Datumをデータとする仮説ノードγ_{Datum}を生成し，これへのストリームをNodeに返す．

`create-p(Datum,^Node)`

`create-p(Datum,PE,^Node)`

Datumをデータとする事実ノードγ_{Datum}を生成し，これへのストリームをNodeに返す．

`create-ja(Datum,Antecedents,^Node,^NewAntecedents)`

`create-ja(Datum,Antecedents,PE,^Node,^NewAntecedents)`

Antecedentsを理由付けの前件に持つ仮説ノードγ_{Datum}を生成し，これへのストリームをNodeに返す．またAntecedentsを新たにNewAntecedentsにする．ここでAntecedentsは，前件ノードのリスト$Node_1, Node_2, ..., Node_n$である．

`revision-a(Node,^NewNode)`

γ_{Datum}が前提あるいは仮説ノードであった時にはなにも行なわず，それ以外の時はγ_{Datum}を仮説ノードとする．

`revision-p(Node,Datum,^NewNode)`

γ_{Datum}がすでに前提ノードであった時にはなにも行なわず，それ以外の時はγ_{Datum}を前提ノードとする．

`revision-ja(Node,Datum,Antecedents,^NewNode,^NewAntecedents)`

γ_{Datum}が前提あるいは仮説ノードであった時には何も行なわず，それ以外の時はγ_{Datum}をAntecedentsを理由付けの前件に持つ仮説ノードとする．ここでAntecedentsは，前件ノードのリスト$Node_1, Node_2, \cdots, Node_n$である．

`aj(Node,Antecedents,^NewNode,^NewAntecedents)`

ノードγ_{Datum}の理由付けにAntecedentsを前件とする理由付けを追加し，新たにNewNodeとする．またAntecedentsを新たにNewAntecedentsにする．ここでAntecedentsは，前件ノードのリスト$Node_1, Node_2, ..., Node_n$である．

`ac(ContNode,Antecedents,^NewContNode,^NewAntecedents)`

矛盾環境に Antecedents のラベルを追加 (矛盾ノード γ_{fail} の理由付けに Antecedents を前件に持つ理由付けを追加) し, 新たに NewContNode とする. また Antecedents を新たに NewAntecedents にする. ここで Antecedents は, 矛盾するノードの組合せのリスト $Node_1$, $Node_2$, $\cdots$, $Node_n$ である. また ContNode が矛盾ノード (アトム fail をデータとするノード) でなければ失敗する.

`show-label(Node,^Label,^NewNode)`
ノード γ_{Datum} のラベルを Label に返す.

`show-node(Node,^Node_data,^NewNode)`
ノード γ_{Datum} のノード情報 (データ, ラベルおよび, 信じられている環境) を Node_data に返す.

`show-nogood(Node,^Nogood)`
矛盾環境のリストを Nogood に返す.

`terminate(Node)`
ノード γ_{Datum} のプロセスを終了する. すべてのノードプロセスはこのメッセージによって終了されねばならない.

3.3 使用例

本章では例を用いて並列 ATMS の利用法を示す.

[例題] 女性 (a,b,c,d) および男性 (e,f,g,h) から女性・男性のペアを全員について知りたい. この時以下に示す仲が良いペアを考慮する. 何通りの方法があるか.

中の良いペア: (a,e), (a,g) (b,e), (b,h),
(c,h), (c,g), (d,f), (d,h)

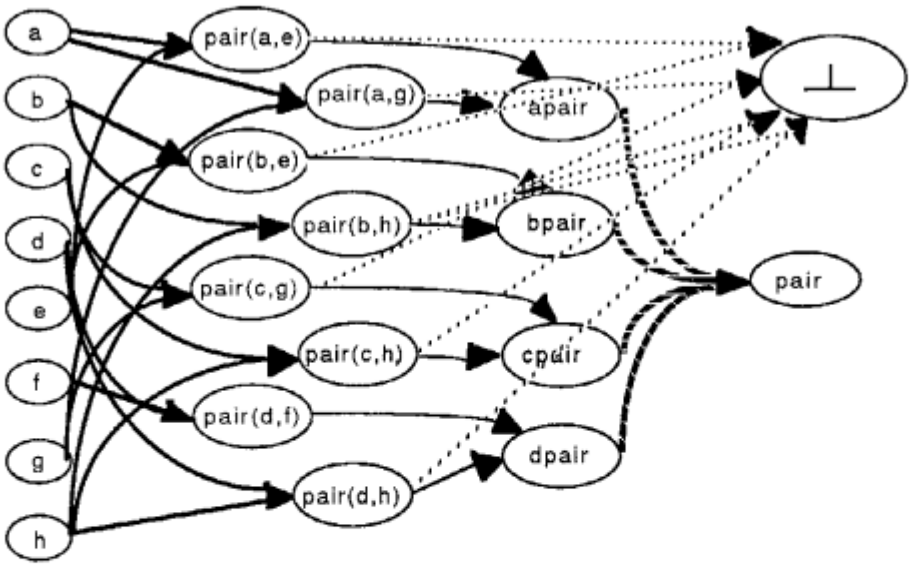

図 3.1: ペア問題理由付け図

[解法例] まず, 仲の良い女性と男性のペアを仮説とする (図 3.1太実線). さらに, 女性 4 人について, 可能性のあるペアを理由付けとして与える (図 3.1細実線). そして, 女性 4 人分の可能なペアを一つの理由付けとして考える (図 3.1破線). また, 異なる女性と同一の男性とのペアの組みあわせは矛盾環境とした (図 3.1点線). これを並列 ATMS に対する上層インタフェースへのメッセージで表現したものを図 3.2に示す.

```
assumption(pair(a,e)).   assumption(pair(a,g)).
assumption(pair(b,e)).   assumption(pair(b,h)).
assumption(pair(c,h)).   assumption(pair(c,g)).
assumption(pair(d,f)).   assumption(pair(d,h)).
        pair(a,e),pair(b,e) ⇒⊥.
        pair(b,h),pair(c,h) ⇒⊥.
        pair(a,g),pair(c,g) ⇒⊥.
        pair(c,h),pair(d,h) ⇒⊥.
        pair(b,h),pair(d,h) ⇒⊥.
pair(a,e) ⇒ apair.       pair(a,g) ⇒ apair.
pair(b,e) ⇒ bpair.       pair(b,h) ⇒ bpair.
pair(c,h) ⇒ cpair.       pair(c,g) ⇒ cpair.
pair(d,h) ⇒ dpair.       pair(d,f) ⇒ dpair.
        apair,bpair,cpair,dpair ⇒ pair.
```

図 3.2: ペア問題理由付け

```
atms:go(M,8), % 使用するプロセッサ数は 8.
M = [
        assumption(pair(a,e)),
        assumption(pair(a,g)),
        assumption(pair(b,e)),
        assumption(pair(b,h)),
        assumption(pair(c,h)),
        assumption(pair(c,g)),
        assumption(pair(d,f)),
        assumption(pair(d,h)),
        ac([pair(a,e),pair(b,e)]),
        ac([pair(b,h),pair(c,h)]),
        ac([pair(a,g),pair(c,g)]),
        ac([pair(c,h),pair(d,h)]),
        ac([pair(b,h),pair(d,h)]),
        aj(apair,[pair(a,e)]),aj(apair,[pair(a,g)]),
        aj(bpair,[pair(b,e)]),aj(bpair,[pair(b,h)]),
        aj(cpair,[pair(c,h)]),aj(cpair,[pair(c,g)]),
        aj(dpair,[pair(d,h)]),aj(dpair,[pair(d,f)]),
        aj(pair,[apair,bpair,cpair,dpair]),
        show-node(pair,PairNode) ].
```

図 3.3: ペア問題メッセージ

ここで, 述語 assumption(X) の形で表されたものは, そのデータ X が真であるという仮説を表している. 図 3.2に示した理由付けを上層インタフェースを介して並列 ATMS に入力する場合のメッセージは図 3.3のようになる (下層インタフェースを用いた場合のメッセージは付録 B を参照のこと). 解答は, ノード *pair* の状態 (PairNode) から 2 通りである. 図 3.3に示したメッセージを与えることで, 内部で無矛盾な環境が生成される. この環境の元で成立するペアを解として求めるためには, ノードの信念の状態を問い合わせればよい. 具体的には, 図 3.3の show-node(pair, PairNode) は, その問い合わせメッセージであり, PairNode にはノード *pair* の信念の状態が以下のように返される.

PairNode = node(pair, {32#{105},32#{86}}, [[pair(c,g), pair(d,f), pair(b,h), pair(a,e)], [pair(d,f), pair(c,h), pair(b,e), pair(a,g)]])

本システムでは ATMS の内部で信じられているノードを一括して表示する機能はサポートされていない. したがって show-node などのノード情報の表示用のメッセージを用いて, ユーザ自身, あるいはそのコンポーネントとして用いる推論エンジンの側で提供する必要がある.

3.4 使用条件およびインストール方法

以下に並列 ATMS のインストール方法と使用条件を示す.

3.4.1 使用条件

本システムは以下の環境においてその動作を保証する.

表 3.1: 並列 ATMS 実行環境

H/W	SIMPOS	PIMOS	CSP	KL1 ファーム	KL0 ファーム
Multi-PSI あるいは PSI-II 上の Pseudo Multi-PSI	6.00 版以降	2.00 版以降	3.00 版以降	1.18 版以降	1.80 版以降

3.4.2 インストール方法

並列 ATMS のインストール方法は以下の手順で行なう.

1. ソースファイルのコピー リリースは基本的にソースファイルで行なう. 並列 ATMS のシステムフロッピーディスクから以下のソースファイルを自分のディレクトリにコピーする.

```
lib.kl1, et.kl1, lo.kl1, io.kl1, nodeAP.kl1, nodeC.kl1,
rev.kl1, cnode.kl1, npm.kl1, if.kl1, top.kl1
```

2. コンパイル Multi-PSI あるいは Pseudo Multi-PSI を立ちあげて, login する. 次に, コンパイラを起動し, 1でコピーしたファイルをコンパイルする.

3. アンロード 2でコンパイルしたモジュールをアンローダでファイルにセーブする.

```
SHELL> unload([
               patms_lib, patms_environment_table, patms_label_operation,
               patms_io, patms_ordinary_node, patms_contradiction_node,
               patms_create_node, patms_npm, patms_interface, patms],
               ''patms'')
```

一度アンロードしておけば, つぎに login した時には, セーブファイル (上の場合 "patms.sav") を以下のようにロードするだけで良い.

```
SHELL> load(''patms'')
```

第 4 章

ソースコード

本章では, 並列 ATMS のソースコードを示す.

4.1 モジュール patms

```
%-----------------------------------------------------%
% 並列 ATMS のトップモジュール.
%-----------------------------------------------------%
:- module patms.
:- with_macro pimos.
:- public got/2,go/2,go/3,goE/3,goF/3,
                              patms_lif/1,patms_lif/2,patms_lif/3,patms_lifF/3.
%%%%%%%%% top lebel interface
got(MES,TRANS) :-
        true |
        current_processor(_,X,Y),
        PEsize := X*Y,
        initiate_node_pool(PoolV,PEsize),
        patms_interface:interface_fork(MES,TRANS,PoolV),
        patms_npm:go(TRANS,PEsize,_).

go(MES,PEsize) :-
        integer(PEsize)|
        initiate_node_pool(PoolV,PEsize),
        patms_interface:interface_fork(MES,TRANS,PoolV),
        patms_npm:go(TRANS,PEsize,_).

go(MES,PEsize,Time) :-
        system_timer(_,S), integer(PEsize)|
        initiate_node_pool(PoolV,PEsize),
        patms_interface:interface_fork(MES,TRANS,PoolV),
        patms_npm:go(TRANS,PEsize,End),
        patms_lib:exec_time(End,S,Time).

goE(MES,PEsize,End) :-
        integer(PEsize)|
        initiate_node_pool(PoolV,PEsize),
        patms_interface:interface_fork(MES,TRANS,PoolV),
        patms_npm:go(TRANS,PEsize,End).
```

```
goF(IFileString,PEsize,Time) :-
        system_timer(_,S), integer(PEsize)|
        initiate_node_pool(PoolV,PEsize),
        patms_io:file_input(IFileString,MES),
        patms_interface:interface_fork(MES,TRANS,PoolV),
        patms_npm:go(TRANS,PEsize,End),
        patms_lib:exec_time(End,S,Time).

%%%%%%%%%%%%%%%% low lebel interface
patms_lif(MES) :- true |
        current_processor(_,X,Y),
        PEsize := X*Y,
        patms_npm:go(MES,PEsize,_).
patms_lif(MES,Time) :-
        system_timer(_,S)|
        current_processor(_,X,Y),
        PEsize := X*Y,
        patms_npm:go(MES,PEsize,End),
        patms_lib:exec_time(End,S,Time).
patms_lif(MES,PEsize,Time) :-
        system_timer(_,S), integer(PEsize)|
        patms_npm:go(MES,PEsize,End),
        patms_lib:exec_time(End,S,Time).
patms_lifF(IFileString,PEsize,Time) :-
        system_timer(_,S), integer(PEsize)|
        patms_io:file_input(IFileString,MES),
        patms_npm:go(MES,PEsize,End),
        patms_lib:exec_time(End,S,Time).

%%%%%%%%%%%%%%%%
initiate_node_pool(PoolV,PESize) :- true |
PoolSize = 30,
        pool:keyed_sorted_set(Pool),
        pool:keyed_sorted_set([put(0,{Pool,0},_)|Pools]),
        pool:keyed_sorted_set(Index),
        PoolV = {Index,0,Pools,{PoolSize,PESize}}. %%%%%  Pool size is 30.

%------------------------------------------------%
%------------------------------------------------%
```

4.2 モジュール patms_interface

```
%------------------------------------------------%
%  上層インタフェース
%------------------------------------------------%
:- module patms_interface.
:- with_macro pimos.
:- public interface_fork/3.

%%%%%%%%%% interface fork
interface_fork([exit],TRANS,Pool) :-
        true |
```

```
        terminate_all(Pool,TRANS).
interface_fork([],TRANS,Pool) :-
        true |
        terminate_all(Pool,TRANS).
otherwise.
interface_fork([H|MES],TRANS,PoolV) :-
        true |
        interface(H,TRANS,TRANST,PoolV,NewPoolV)@priority(*,3000),
        interface_fork(MES,TRANST,NewPoolV)@priority(*,2000).

%%%%%%%%%%  upper level interface

%%%% create node process

interface(MES,TRANS,TRANST,Pool,New_pool) :-
        vector_element(MES,0,assumption) |
        assumption(MES,TRANS,TRANST,Pool,New_pool).
interface(MES,TRANS,TRANST,Pool,New_pool) :-
        vector_element(MES,0,jassume) |
        jassume(MES,TRANS,TRANST,Pool,New_pool).
interface(MES,TRANS,TRANST,Pool,New_pool) :-
        vector_element(MES,0,premise) |
        premise(MES,TRANS,TRANST,Pool,New_pool).

%%%%%%%%%%  add justification and nogood environment

interface(aj(Datum,Antecedents),TRANS,TRANST,Pool,NewPool) :-
        true |
%        patms_lib:out(aj(Datum,Antecedents)),
        TRANS_B = [aj(DV,AnV,DVN,AnVE)|TRANST],
        get_pool(Datum,DV,Pool,Npool,TRANS,TRANS_A,DVN),
        get_antecedents(Antecedents,AnV,AnVE,Npool,NewPool,TRANS_A,TRANS_B).

interface(ac(Antecedents),TRANS,TRANST,Pool,NewPool) :-
        true |
%        patms_lib:out(ac(Antecedents)),
        TRANS_B = [ac(AnV,AnVE)|TRANST],
        get_antecedents(Antecedents,AnV,AnVE,Pool,NewPool,TRANS,TRANS_B).

%%%%%%%%%%    show status of nodes
interface(show-label(Datum,Label),TRANS,TRANST,Pool,NewPool) :-
        true |
%        patms_lib:out(swl(Datum)),
        TRANS_A = [show-node(Node,node(_,Label,_),NNode)|TRANST],
        get_pool(Datum,Node,Pool,NewPool,TRANS,TRANS_A,NNode).

interface(show-nogood(Nogood),TRANS,TRANST,Pool,NewPool) :-
        true |
%        patms_lib:out(swn),
        TRANS = [show-nogood(Nogood)|TRANST],
        Pool = NewPool.
```

```
interface(show-node(Datum,Node_data),TRANS,TRANST,Pool,NewPool) :-
        true |
%        patms_lib:out(shownode(Datum)),
        TRANS_A = [show-node(Node,Node_data,NNode)|TRANST],
        get_pool(Datum,Node,Pool,NewPool,TRANS,TRANS_A,NNode).

%%%%%%%%%% utility
assumption({_,Datum},TRANS,TRANST,Pool,New_pool) :- true |
        put_pool_and_check(Datum,auto,a,Pool,TRANS,TRANST,New_pool).
assumption({_,Datum,PE},TRANS,TRANST,Pool,New_pool) :- true |
        put_pool_and_check(Datum,PE,a,Pool,TRANS,TRANST,New_pool).
%
jassume({_,Datum,Ant},TRANS,TRANST,Pool1,New_pool) :- true |
        get_antecedents(Ant,AntV,AntVE,Pool1,Pool2,TRANS,TRANS_B),
        put_pool_and_check(Datum,auto,ja(AntV,AntVE),Pool2,TRANS_B,TRANST,New_pool).
jassume({_,Datum,Ant,PE},TRANS,TRANST,Pool1,New_pool) :- true |
        get_antecedents(Ant,AntV,AntVE,Pool1,Pool2,TRANS,TRANS_B),
        put_pool_and_check(Datum,PE,ja(AntV,AntVE),Pool2,TRANS_B,TRANST,New_pool).
%
premise({_,Datum},TRANS,TRANST,Pool,New_pool) :- true |
        put_pool_and_check(Datum,auto,p,Pool,TRANS,TRANST,New_pool).
premise({_,Datum,PE},TRANS,TRANST,Pool,New_pool) :- true |
        put_pool_and_check(Datum,PE,p,Pool,TRANS,TRANST,New_pool).

%%%%%%%%%% node process stream manager

get_antecedents([],Ans,NewAns,Pool,PoolT,TR,TRT) :-
       true |
       NewAns = [],
       Ans = [],
       TR = TRT,
       Pool = PoolT.
get_antecedents([H|T],Ans,NewAns,Pool,PoolEND,TR,TRT) :-
        list(H) |
        NewAns=[NewH|NewT],
        get_antecedents(T,AnsT,NewT,NPool,PoolEND,TR_A,TRT),
        get_antecedents(H,AnsH,NewH,Pool,NPool,TR,TR_A),
        Ans = [AnsH|AnsT].
otherwise.
get_antecedents([H|T],Ans,NewAns,Pool,PoolEND,TR,TRT) :-
        true |
        NewAns=[NewH|NewT],
        get_pool(H,HV,Pool,Npool,TR,TR_A,NewH),
        get_antecedents(T,ANST,NewT,Npool,PoolEND,TR_A,TRT),
        Ans = [HV|ANST].

%%%%%%%%%%
get_pool(Datum,Stream,{Index,C,Pools,Size},NewPool,TR,TRT,NewStream) :- true |
        NewPool = {NewIndex,NewC,NewPools,Size},
        Index = [get_if_any_and_put(Datum,PoolNum,NewPoolNum)|NewIndex],
```

```
        search_pool(PoolNum,NewPoolNum,Pools,C,Size,NewPools,Datum,Stream,TR,TRT,NewStream,NewC).

search_pool({},NewPoolNum,Pools,C,Size,NewPools,Datum,Stream,TR,TRT,NewStream,NewC) :- true |
        TR =[create-n(Datum,Stream)|TRT],
        NewPoolNum = NewC,
        put_pool(Datum,NewStream,Pools,C,Size,NewPools,NewPoolNum).
search_pool({PoolNum},NewPoolNum,Pools,C,_,NewPools,Datum,Stream,TR,TRT,NewStream,NewC) :-  true |
        TR = TRT,
        C = NewC,
        NewPoolNum = PoolNum,
        Pools = [get_if_any_and_put(PoolNum,{{Pool,CSize}},{NewPool,CSize})|NewPools],
        Pool = [get_if_any_and_put(Datum,{Stream},NewStream)|NewPool].

%%%%%%%%%%  put pool

put_pool(Datum,Stream,Pools,C,Size,NewPools,NewPoolNum) :- true  |
        Pools = [get_if_any_and_put(C,{{Pool,CSize}},{NewPool,NewCSize})|NewPools_],
        check_and_put(CSize,Size,NewCSize,Pool,NewPool,Datum,Stream,NewPools_,NewPools,NewPoolNum,C).
check_and_put(CSize,{Size,PESize},NewCSize,Pool,NewPool,Datum,Stream,Pools,NewPools,PoolNum,C) :-
        CSize < Size |
        Pools = NewPools,
        PoolNum = C,
        NewCSize := CSize + 1,
        Pool = [put(Datum,Stream,_)|NewPool].
otherwise.
check_and_put(CSize,{_,PESize},NewCSize,Pool,NewPool,Datum,
                Stream,Pools,NewPools,PoolNum,C) :- true |
        Pool = NewPool, CSize = NewCSize,
        PENum := PoolNum mod PESize,
        pool:keyed_sorted_set([put(Datum,Stream,_)|NextPool])@node(PENum),
        Pools = [put(PoolNum,{NextPool,0},_)|NewPools],
        PoolNum := C+1.

%-----------------------------------%
put_pool_and_check(Datum,PE,Type,Pool,TRANS,TRANST,NewPool) :-
        true |
        get_existent_node(Datum,Old,New,Pool,NewPool),
        have_created0(PE,Type,Old,Datum,New,TRANS,TRANST).

have_created0(auto,Type,Old,Datum,DVN,TRANS,TRANST) :- true |
        have_created1(Old,Type,Datum,DVN,TRANS,TRANST).
have_created0(PE,Type,Old,Datum,DVN,TRANS,TRANST) :- integer(PE) |
        have_created2(Old,Type,PE,Datum,DVN,TRANS,TRANST).

have_created1({},ja(Ant,NAnt),Datum,New,TRANS,TRANST) :- true  |
        TRANS = [create-ja(Datum,Ant,New,NAnt)|TRANST].
otherwise.
have_created1({},Type,Datum,New,TRANS,TRANST) :- true  |
        TRANS = [{-,create,{Type,Datum,New}}|TRANST].
have_created1({Old},a,_,New,TRANS,TRANST) :- true |
        TRANS = [{-,revision,{a,Old,New}}|TRANST].
```

```
have_created1({Old},p,_,New,TRANS,TRANST) :- true |
        TRANS = [{-,revision,{p,Old,New}}|TRANST].
have_created1({Old},{ja,Ant,NAnt},_,New,TRANS,TRANST) :- true |
        TRANS = [{-,revision,{ja,Old,Ant,New,NAnt}}|TRANST].

%%%%%%%%%%
have_created2({},ja(Ant,NAnt),PE,Datum,New,TRANS,TRANST) :- true  |
        TRANS = [create-ja(Datum,Ant,PE,New,NAnt)|TRANST].
otherwise.
have_created2({},Type,PE,Datum,New,TRANS,TRANST) :- true  |
        TRANS = [{-,create,{Type,Datum,PE,New}}|TRANST].
have_created2({Old},a,_,_,New,TRANS,TRANST) :- true |
        TRANS = [rivision-a(Old,New)|TRANST].
have_created2({Old},p,_,_,New,TRANS,TRANST) :- true |
        TRANS = [rivision-p(Old,New)|TRANST].
have_created2({Old},{ja,Ant,NAnt},_,_,New,TRANS,TRANST) :- true |
        TRANS = [revision-ja(Old,Ant,New,NAnt)|TRANST].

%%%%%%%%%%   get stream

get_existent_node(Datum,Val,NewV,{Index,C,Pools,Size},NewPool) :-
        true |
        NewPool = {NewIndex,NewC,NewPools,Size},
        Index = [get_if_any_and_put(Datum,PoolNum,NewPoolNum)|NewIndex],
        check_index(PoolNum,NewPoolNum,Pools,NewPools,C,Size,Datum,Val,NewV,NewC).

check_index({},NewPoolNum,Pools,NewPools,C,Size,Datum,Val,NewV,NewC) :- true |
        Val = {},
        NewPoolNum = NewC,
        put_pool(Datum,NewV,Pools,C,Size,NewPools,NewPoolNum).
check_index({PoolNum},NewPoolNum,Pools,NewPools,C,_,Datum,Val,NewV,NewC) :- true |
        C = NewC,
        PoolNum = NewPoolNum,
        Pools = [get_if_any_and_put(PoolNum,{{Pool,CSize}},{NewPool,CSize})|NewPools],
        Pool = [get_if_any_and_put(Datum,Val,NewV)|NewPool].

%%%%%%%%%% terminate all node process

terminate_all({Index,C,Pools,_},TRANS) :-
        true |
        Index = □,
        Pools = [get_all(ALLP)],
        all_get(ALLP,ALLM),
        merge(ALLM,ALL),
        input_terminate(ALL,TRANS).

all_get([{_,{Pool1,_}}|T],ALL) :- true |
        ALL = {ALL1,ALL2},
        Pool1 = [get_all(ALL1)],
        all_get(T,ALL2).
all_get(□,ALL) :- true |  ALL = □.
```

```
input_terminate_all([],TRANS) :-
        true |
        TRANS = [].

input_terminate_all([{_,Pool}|T],TRANS) :-
        true |
        Pool = [get_all(ALL)],
        merge({TTRANS,NTRANS},TRANS),
        input_terminate(ALL,TTRANS),
        input_terminate_all(T,NTRANS).

input_terminate([],TRANS) :-
        true |
        TRANS = [].
input_terminate([{_,Val}|T],TRANS) :-
        true |
        TRANS = [terminate(Val)|TTRANS],
        input_terminate(T,TTRANS).

%-----------------------------------------------%
%-----------------------------------------------%
```

4.3 モジュール patms_npm

```
%-----------------------------------------------%
% 下層インタフェース
%-----------------------------------------------%
:- module patms_npm.
:- with_macro pimos.
:- public go/3,go/4,npm/6.

go(MES_, PE, End) :- true |
   merge(MES_,MES),
   PEs := PE - 1,
   processor_num(0,PEs,PEn),
   patms_environment_table:initial_table(E_Pool),
   contradiction_fork(Cont, PEn, NPEn, end-Ep),   %   PEn <-> 1
   npm(MES, {}, env(32#{1},E_Pool), processor(NPEn,PEs), Cont, Ep-End).

go(MES_, Nogood, PE, End) :- true |
   PEs := PE - 1,
   processor_num(0,PEs,PEn),
   merge(MES_,MES),
   patms_environment_table:initial_table(E_Pool),
   contradiction_fork(Cont, Nogood, PEn, NPEn, end-Ep)@priority(*,4095),
   npm(MES, Nogood, env(32#{1},E_Pool), processor(NPEn,PEs), Cont, Ep-End)@priority(*,4095).

%-----------------------------------------%
npm([COM|MT],Ng,Env,{processor,CP,PEs},Cont,S-E) :-
    vector_element(COM,1,create) |
    set_vector_element(COM,2,MES,end,_),
```

```
    processor_num(CP,PEs,PE),
    node_fork(MES, Env, NEnv, Cont, NCont, PE, S-Ep1), % 4/9
    npm(MT,Ng,NEnv,{processor,~(PE+1),PEs},NCont,Ep1-E).
npm([COM|MT],Ng,Env,PE,Cont,end-E) :-
    vector_element(COM,1,show) |
%    patms_lib:out(show(node)),
    set_vector_element(COM,2,MES,end,_),
    show_status(MES,Env,NEnv,Cont,NewCont,Ep),
    npm(MT,Ng,NEnv,PE,NewCont,Ep-E).
npm([COM|MT],Ng,Env,{processor,CP,PEs},Cont,S-E) :-
    vector_element(COM,1,revision)|
    set_vector_element(COM,2,MES,end,_),
    processor_num(CP,PEs,PE),
    patms_node_revision:revision(MES, Env, NEnv, Cont, NCont, S-Ep),
    npm(MT,Ng,NEnv,{processor,~(PE+1),PEs},NCont,Ep-E).
npm([{aj,Node,Ant,NNode,NAnt}|MT],Ng,Env,PE,Cont,S-E) :- true |
    Node = {[add(just(Ant, Ng, NAnt, Self, S-Ep))], Self, NNode},
    npm(MT,Ng,Env,PE,Cont,Ep-E).
npm([{ac,Ant,NAnt}|MT],Ng,Env,PE,Cont,S-E) :- true |
    Cont = {[add_contradiction(Ant, NNg, Ng, _, Self, NAnt, S-Ep)], Self, NCont},
    npm(MT,NNg,Env,PE,NCont,Ep-E).
npm([{ac,Ant,FPE,NAnt}|MT],Ng,Env,PE,Cont,S-E) :- true |
    Cont = {[add_contradiction(Ant, NNg, Ng, _, Self, FPE,NAnt, S-Ep)], Self, NCont},
    npm(MT,NNg,Env,PE,NCont,Ep-E).
npm([{get_label,Node,Label,NNode}|MT],Ng,Env,PE,Cont,S-E) :- true |
    Node = {[{show,{label,Label,S-Ep}}],NNode},
    npm(MT,Ng,Env,PE,Cont,Ep-E).
npm([{terminate,Node}|MT],Ng,Env,PE,Cont,S-E) :- true |
    terminate_node(E,Node),
    npm(MT,Ng,Env,PE,Cont,S-E).
npm(□,_,Env,_,Cont,S-E) :- true |
    atms_terminate(S,Cont,Env,E).


%-----------------------------------------------%
node_fork(MES, Env, NEnv, Cont, NCont, PE, S-E) :-  true |
    patms_create_node:node_fork(MES, Env, NEnv, Cont, NCont, PE, S-E)@priority(*,3000).

contradiction_fork(M, PEn, NPEn, S-E) :- true |
    NPEn := PEn + 1,
    create_contradiction_node_exe(M, S-E)@node(PEn).
contradiction_fork(M, Ng, PEn, NPEn, S-E) :- true |
    NPEn := PEn + 1,
    create_contradiction_node_exe(M, Ng, S-E)@node(PEn).

%-----------------------------------------------%
create_contradiction_node_exe(M, S-E) :- true |
    create_contradiction_node(M, S-E)@priority(*,3000).
create_contradiction_node_exe(M, Ng, S-E) :- true |
    create_contradiction_node(M, Ng, S-E)@priority(*,3000).
```

```
create_contradiction_node(M, S-E) :- true |
    merge(M,Me),
    patms_contradiction_node:cnode_process(Me),
    process_end(S,E)@priority($,-100).

create_contradiction_node(M, Ng, S-E) :- true |
    merge(M,Me),
    patms_contradiction_node:cnode_process(Me, Ng),
    process_end(S,E)@priority($,-100).

%-------------------------------------------------%
show_status({node,Node,SNODE,NewNode},{env,Env,E_P},NewEnv,Cont,NewCont,E) :- true |
    Node = {[{show,{node,node(D,Labels,_),Ep}}],NewNode},
    SNODE = node(D,Labels,Datums),
    patms_environment_table:search_datums(Labels,Datums,E_P,NE_P,Ep-E),
%    patms_lib:out(result(Datums)), %%%%%@@@@@
    Cont = NewCont,
    NewEnv = {env,Env,NE_P}.
show_status({nogood,Nogood},{env,Env,E_P},NewEnv,Cont,NewCont,E) :- true |
    Cont = {[{show,{node,node(D,Labels,_),Ep}}],NewCont},
    Nogood = node(D,Labels,Datums),
    patms_environment_table:search_datums(Labels,Datums,E_P,NE_P,Ep-E),
    NewEnv = {env,Env,NE_P}.
show_status({cd,Node,CD,NewNode},Env,NewEnv,Cont,NewCont,E) :- true |
    Node = {[{show,{node,node(_,_,CD),E}}],NewNode},
    Cont = NewCont,
    Env = NewEnv.

process_end(end,E) :- true | E = end.

terminate_node(end,Node) :- true |
    Node = [{terminate,end}].

atms_terminate(end,Cont,{env,_,EP},E) :- true |
        Cont = [{terminate, end}],
        EP = [],
        E = end.

processor_num(C,Size,P) :- C > Size | P = 0.
processor_num(C,Size,P) :- C =< Size | P = C.
%-------------------------------------------------%
%-------------------------------------------------%
```

4.4 モジュール patms_create_node

```
%-------------------------------------------------%
%  ノードプロセス作成
%-------------------------------------------------%
:- module patms_create_node.
:- with_macro pimos.
:- public node_fork/7.
```

```
%-----------------------------------------------%
node_fork(COM, Env, NEnv, A, NA, PE, S-Ep) :- true |
    node_create(COM, Env, NEnv, A, NA, PE, S-Ep).

node_create(COM, Env, NEnv, A, NA, PE, S-E) :-
    vector_element(COM,0,n) |
    Env = NEnv,
    cn(COM,PE,MA,S-Ep),
    A = {[add_attention(MA,Ep-E)],NA}.
node_create(COM, Env, NEnv, A, NA, PE, S-E) :-
    vector_element(COM,0,a) |
    can(COM,Env,NEnv,MA,PE,S-Ep),
    A = {[add_attention(MA,Ep-E)],NA}.
node_create(COM, Env, NEnv, A, NA, PE, S-E) :-
    vector_element(COM,0,ja) |
    cawj(COM,Env,NEnv,A,NA_,MA,PE,S-Ep),
    NA_ = {[add_attention(MA,Ep-E)],NA}.
node_create(COM, Env, NEnv, A, NA, PE, S-E) :-
    vector_element(COM,0,p) |
    A = NA,
    Env = NEnv,
    cpn(COM,PE,S-E).

%-----------------------------------------------%
cn({_,Node_datum, PE, Node_S_M},_,MA,S-E) :- true |
       cn(cn(Node_datum,Node_S_M),PE,MA,S-E).
cn({_, Node_datum, Node_S_M},PE,MA,S-E) :-
       true |
       process_end(S,E),
%       patms_lib:out(node(E,Node_datum,PE)),
       patms_ordinary_node:node_process({MA,Node_S_M}, {Node_datum,{},i})@node(PE).

%-----------------------------------------------%
can({_,Node_datum, PE,Node_S_M}, Env, NEnv, MA, _, S-E) :- true |
    can({_, Node_datum, Node_S_M}, Env, NEnv, MA, PE, S-E).
can({_,Node_datum, Node_S_M}, {env,EnvironmentF,Env_Pool}, NEnv, MA, PE, S-E) :-
   true |
   NEnv = env(Environment, NEnv_Pool),
   patms_label_operation:new_assumption(EnvironmentF, Environment),
   patms_ordinary_node:ass_node_process({MA,Node_S_M}, {Node_datum,
                          EnvironmentF, Env_Pool, NEnv_Pool,0},S-E)@node(PE). %4/9
%-----------------------------------------------%
cawj({_, Datum, Ant, PE, Node_S_M, NAnt},  Env, NEnv, A, NA, MA, _, S-E) :- true |
    cawj({_, Datum, Ant, Node_S_M, NAnt},  Env, NEnv, A, NA, MA, PE, S-E).
cawj({_, Datum, Ant, Node_S_M, NAnt},
               {env,EnvironmentF,Env_Pool}, NEnv, A, NA, MA, PE, S-E) :- true |
      NEnv = env(Environment, NEnv_Pool),
      A = {[{show,{label,Ng,S-Ep}}],NA},
      patms_label_operation:new_assumption(EnvironmentF, Environment),
      patms_ordinary_node:assumed_node_process({MA,Node_S_M},
        {Datum, Ant, NAnt, Ng, EnvironmentF, Env_Pool, NEnv_Pool,0},Ep-E)@node(PE).
```

```
%-------------------------------------------------%
cpn({_,Node_datum, PE, Node_S_M}, _, S-E) :- true |
       cpn({_,Node_datum, Node_S_M}, PE, S-E).
cpn({_,Node_datum, Node_S_M}, PE, S-E) :-
       true |
       process_end(S,E),
%       patms_lib:out(p(E,Node_datum,PE)), %%%%%%%%%%%@@@@@@@
       patms_ordinary_node:node_process(Node_S_M, {premise(Node_datum), {32#{0}}, 0})@node(PE).

%-------------------------------------------------%
process_end(end,E) :- true | E = end.
%-------------------------------------------------%
```

4.5 モジュール patms_ordinary_node

```
%-------------------------------------------------%
% 仮説および前提ノードプロセス
%-------------------------------------------------%
:- module patms_ordinary_node.
:- with_macro pimos.
:- public node_process/2, ass_node_process/3, assumed_node_process/3.


%%%%%%%%%% node process
node_process(MessageM, {Datum,Label,Cd}) :- true |
       merge(MessageM, Message),
       pool:keyed_sorted_set(Consq),
       node_process_(Message, {Datum,Label,□,Consq,Cd}).

ass_node_process(MessageM, {Datum,Env,Envp,NewEnvp,Cd},S-E) :- true |
       merge(MessageM, Message),
       patms_environment_table:set_table(Env,Datum,Envp,NewEnvp,S-E),
       pool:keyed_sorted_set(Consq),
       node_process_(Message, {assumption(Datum),{Env},[[{Env}]],Consq,Cd}).

assumed_node_process(MessageM,{Datum,Ant,NewAnt,Ng,EnvF,Envp,NewEnvp,Cd},S-E) :- true |
       patms_lib:connect(Ant, Datum, Label_ListM, Self, NewAnt, S-Ep),
       merge(Label_ListM,Label_List),
       NewLabelList=[{EnvF}|Label_List],
       patms_lib:label_updating(NewLabelList, Ng, {}, NewLabel, _, NewCd, _, Ep-Ep1),
       merge({Self,MessageM}, Message),
       patms_environment_table:set_table(EnvF,Datum,Envp,NewEnvp,Ep1-E),
       pool:keyed_sorted_set(Consq),
       node_process_(Message, {assumption(Datum),NewLabel,[NewLabelList],Consq,NewCd}).

node_process_([Message|T], NodeV) :- true |
                     node(Message, NodeV, NewNodeV),
                     node_process_(T, NewNodeV).
node_process_(□, _) :- true | true.

%%%%%%%%%% node
```

```
node({remove,Remove_Label,S-E}, NodeV, NewNodeV) :- true  |
    set_vector_element(NodeV,1,Label,NewLabel,NodeV0),
    set_vector_element(NodeV0,2,JL,NewJL,NodeV1),
    set_vector_element(NodeV1,4,Cd,NewCd,NewNodeV),
    remove_environment(Label,JL,Cd,Remove_Label,NewLabel,NewJL,NewCd,S-E).
node({show,Mes}, NodeV, New_NodeV) :- true |
        show_node_info(Mes, NodeV, New_NodeV).
alternatively.
node({terminate,end}, NodeV, _) :-
        vector_element(NodeV,0,D),
        vector_element(NodeV,3,Consequent) |
%        patms_lib:out(node(terminate)),
        patms_lib:c_terminate(Consequent).
node({add, Mes}, NodeV, NewNodeV) :- true |
        update_node(Mes, NodeV, NewNodeV).
node({rev, Mes}, NodeV, NewNodeV) :- true |
        patms_node_revision:revision(Mes, NodeV, NewNodeV).

show_node_info({node,Sound_Node,E},NodeV,NewNodeV) :-
        vector_element(NodeV,0,D),
        vector_element(NodeV,1,L),
        vector_element(NodeV,4,Cd) |
        NodeV = NewNodeV,
        patms_lib:sound_node(end,Sound_Node,D,L,Cd,E).
show_node_info({label, Label, S-E}, NodeV, NewNodeV) :-
        vector_element(NodeV,1,LabelC) |
        Label = LabelC,
        NodeV = NewNodeV,
        patms_lib:process_end(S,E).
show_node_info({datum,Datum}, NodeV, NewNodeV) :-
        vector_element(NodeV,0,D) |
        NodeV = NewNodeV,
        show_datum(D,Datum).
show_node_info({contradictory,Cd}, NodeV, NewNodeV) :-
        true |
        set_vector_element(NodeV,4,Cd,Cd,NewNodeV).
update_node({just,Add_justification,CNogood, New_add_justification,
                    Self, S-E},
                    NodeV, NewNodeV ) :-
      vector_element(NodeV,0,D) |
%      patms_lib:out(E,node(just)),
      set_vector_element(NodeV,1,L,NewLabel,NodeV1),
      set_vector_element(NodeV1,2,JL,NewJL,NodeV2),
      set_vector_element(NodeV2,3,C,New_C2,NodeV3),
      set_vector_element(NodeV3,4,Cd,NewCd,NewNodeV),
      patms_lib:connect(Add_justification, D, Label_ListM, Self, New_add_justification, S-Ep1),
      merge(Label_ListM,Label_List),
      patms_lib:label_updating(Label_List, CNogood, L, NewLabel, _, NewCd, NewNg, Ep1-Ep2),
      NewJL = [Label_List|JL],
      patms_lib:check_and_propagation(L, NewLabel, C, CNogood, New_C2, Ep2-E).
```

```
update_node({consequents,Key,Consq,S-E},NodeV,NewNodeV) :-
      true |
      set_vector_element(NodeV,3,C,NewC,NewNodeV),
      C = [put(Key,Consq,Old)|NewC],
      patms_lib:check_old_consq(Old),
      patms_lib:process_end(S,E).
update_node({label,New_nogood, CL, NL,  S-E}, NodeV, NewNodeV) :-
     vector_element(NodeV,0,D)|
      set_vector_element(NodeV,1,L,NewLabel,NodeV0),
      set_vector_element(NodeV0,2,JL,NewJL,NodeV1),
      set_vector_element(NodeV1,3,C,NewC,NodeV2),
      set_vector_element(NodeV2,4,Cd,NewCd,NewNodeV),
%      patms_lib:out({update,D}),
      patms_lib:renewal(New_nogood, L, JL, CL, NL, C, Cd, NewLabel, NewJL, NewCd, NewC, S-E).

%%%%%%%%%%  utility
remove_environment(Label,JL,1,_,NewLabel,NewJL,NewCd,S-E) :- true |
      NewLabel = Label,
      JL = NewJL,
      NewCd = 1,
      patms_lib:process_end(S,E).
remove_environment(Label,JL,Cd,RL,NewLabel,NewJL,NewCd,S-E) :-
      vector(Label,1),vector(RL,1),
      vector_element(Label,0,Le),vector_element(RL,0,RLe),
      string(Le,1,_),string(RLe,1,_),
      string_element(Le,0,LeS),string_element(RLe,0,RLeS),
      and(LeS,RLeS,0)  |
      NewLabel = Label,
      JL = NewJL,
      NewCd = Cd,
      patms_lib:process_end(S,E).
remove_environment({32#{0}},JL,Cd,_,NewLabel,NewJL,NewCd,S-E) :- true |
      NewLabel = {32#{0}},
      JL = NewJL,
      Cd = NewCd,
      patms_lib:process_end(S,E).
remove_environment(L,_,_,L,NewLabel,NewJL,NewCd,S-E) :- true |
      NewLabel = {},
      NewJL = □,
      NewCd = 1,
      patms_lib:process_end(S,E).
otherwise.
remove_environment(_,JL,_,NgL,NewLabel,NewJL,NewCd,S-E) :-
      true |
      patms_label_operation:all_label_union(JL,{},{},NgL,{},NewLabelC,NewJL,_),
      vector(NewLabelC,Size,NewLabel),
      patms_lib:check_contradictory(Size,NewCd,S-E).

show_datum(premise(D),Datum) :-true | Datum = D.
show_datum(assumption(D),Datum) :-true | Datum = D.
show_datum(assumed(D),Datum) :-true | Datum = D.
```

```
otherwise.
show_datum(D,Datum) :-true | Datum = D.
%-----------------------------------------------%
%-----------------------------------------------%
```

4.6 モジュール patms_contradiction_node

```
%-----------------------------------------------%
% 矛盾ノードプロセス
%-----------------------------------------------%
:- module patms_contradiction_node.
:- with_macro pimos.
:- public cnode_process/1, cnode_process/2.

cnode_process(MessageM) :- true |
                    merge(MessageM,Message),
                    CNodeV = {fail,{},[], 1, []},
                    cnode_loop(Message, CNodeV).
cnode_process(MessageM,Nogood) :- true |
                    merge(MessageM,Message),
                    CNodeV = {fail,Nogood, [], 0, []},
                    cnode_loop(Message, CNodeV).

cnode_loop([Message|T], CNodeV) :-
                    true |
                    nogood(Message, CNodeV, NewCNodeV),
                    cnode_loop(T, NewCNodeV).
cnode_loop([], _) :- true | true.

%%%%%%
%%%%%%  Contradiction node process
%%%%%%

nogood({show,{label,Label, S-E}}, NodeV, NewNodeV) :-
        true |
        set_vector_element(NodeV,1,Label,Label,NewNodeV),
        patms_lib:process_end(S,E).
nogood({show,{node,Sound_Node,E}},NodeV,NewNodeV) :-
        vector_element(NodeV,0,D),
        vector_element(NodeV,1,L),
        vector_element(NodeV,3,Cd)|
        NodeV = NewNodeV,
        patms_lib:sound_node(end,Sound_Node,D,L,Cd,E).
nogood({add_attention, AddA, S-E}, NodeV, NewNodeV) :- true |
        set_vector_element(NodeV,4,A,NA,NewNodeV),
        NA = [AddA|A],
        patms_lib:process_end(S,E).
nogood(Mes,NodeV, NewNodeV) :-
        vector_element(Mes,0,add_contradiction),
        vector_element(Mes,6,FEP),integer(FEP)|
        nogood_e(Mes, NodeV, NewNodeV)@node(FEP).
nogood({add_contradiction,Add_justification, NewLabel, _, Add_env, Self, Add_justification_1, S-E},
```

```
                        NodeV, NewNodeV) :-
      true |
%      patms_lib:out(E,nogood(end)),
      set_vector_element(NodeV,1,L,NewLabel,NodeV1),
      set_vector_element(NodeV1,2,JL,NewJL,NodeV2),
      set_vector_element(NodeV2,3,Cd,NewCd,NodeV3),
      set_vector_element(NodeV3,4,Attention,NAttention,NewNodeV),
      patms_lib:connect(Add_justification, fail, Label_ListM, Self, Add_justification_1,S-Ep),
      merge(Label_ListM,Label_List),
      patms_lib:label_updating(Label_List, {}, L, NewLabel, Add_env, NewCd, _, Ep-Ep2),
      check_attention(Add_env,Attention,NAttention,Ep2-E),
      NewJL = [Label_List|JL].

nogood({add,{label,_, CL, NL, S-E}}, NodeV, NewNodeV) :-
      true |
      set_vector_element(NodeV,1,L,NewLabel,NodeV0),
      set_vector_element(NodeV0,2,JL,NewJL,NodeV1),
      set_vector_element(NodeV1,3,Cd,NewCd,NodeV2),
      set_vector_element(NodeV2,4,A,NA,NewNodeV),
      check_attention(NL,A,NA,Ep-E),
      patms_lib:renewal({}, L, JL, CL, NL, fail, Cd, NewLabel, NewJL, NewCd, _, S-Ep).

nogood({terminate,end}, NodeV, _) :-
       vector_element(NodeV,4,A) |
%       patms_lib:out(nogood(terminate)),
       broadcast_end(A).

%%%%%
nogood_e(Mes, NodeV, NewNodeV) :- true |
       nogood_sub(Mes, NodeV, NewNodeV)@priority(*,2000).

nogood_sub({add_contradiction,Add_justification, New_Label,
                  Nogood, Add_env, Self, _, New_add_justificationE, S-E},NodeV, NewNodeV) :-
       true |
       nogood({add_contradiction,Add_justification, New_Label,
                  Nogood, Add_env, Self, New_add_justificationE, S-E},NodeV, NewNodeV).

%%%%%%%%%% nogood utility
check_attention({}, Attention, New_Attention, end-E) :-
      true |
      E = end,
      Attention = New_Attention.
otherwise.
check_attention(Add_Label, Attention, New_Attention, end-E) :-
      true |
      broadcast(Attention,Add_Label,New_Attention, end-E).

broadcast([A|AT],Add_Label,New_Attention, S-E) :-
      A = [{remove,Add_Label,S-Ep}|NA],
      New_Attention = [NA|NAT],
      broadcast(AT,Add_Label,NAT,Ep-E).
```

```
broadcast(□,_,NA, S-E) :- true |
      NA = □,
      patms_lib:process_end(S,E).

broadcast_end([A|AT]) :-
      A = □,
      broadcast_end(AT).
broadcast_end(□) :- true |  true.

%-------------------------------------------------%
%-------------------------------------------------%
```

4.7 モジュール patms_node_revision

```
%-------------------------------------------------%
% ノード修正
%-------------------------------------------------%
:- module patms_node_revision.
:- with_macro pimos.
:- public revision/3, revision/6.

%%%%%%%%%% revision/6
revision({a,Node,NewNode},Env,NewEnv,Cont,NewCont,S-E) :- true |
        Cont = NewCont,
        Node = {[{rev,{a,Env,NewEnv,S-E}}],NewNode}.
revision({p,Node,NewNode},Env,NewEnv,Cont,NewCont,S-E) :- true |
        Env = NewEnv, Cont = NewCont,
        Node = {[{rev,{p,S-E}}],NewNode}.
revision({ja,Node,Ant,NewNode,NewAnt},Env,NewEnv,Cont,NewCont,S-E) :- true |
        Node = {[{rev,{ja,Env,NewEnv,Ant,NewAnt,Cont,NewCont,Self,S-E}}],Self,NewNode}.

%%%%%%%%%% revision/6
revision({a,Env,NewEnv,S-E},NodeV,NewNodeV) :-
        true  |
        set_vector_element(NodeV,0,Datum,Datum,NodeV2),
        typeCheck(Datum,Check,a),
        revision_a(Check,Env,NewEnv,NodeV2,NewNodeV,S-E).
revision({p,S-E},NodeV,NewNodeV) :-
        true  |
        set_vector_element(NodeV,0,Datum,Datum,NodeV2),
        premise_check(Datum,Check),
        revision_p(Check,NodeV2,NewNodeV,S-E).
revision({ja,Env,NewEnv,Ant,NewAnt,Cont,NewCont,Self,S-E},NodeV,NewNodeV) :-
        true  |
        set_vector_element(NodeV,0,Datum,Datum,NodeV2),
        typeCheck(Datum,Check,ja),
        revision_ja(Check,Env,NewEnv,Ant,NewAnt,NodeV2,NewNodeV,Cont,NewCont,Self,S-E).

%%%%%%%%%%  Type Check

typeCheck(assumption(_),Check,_) :- true | Check = no.
typeCheck(premise(_),Check,_) :- true | Check = no.
```

```
otherwise.
typeCheck(_,Check,Type) :- true | Check = Type.

premise_check(premise(_),Check) :- true | Check = no.
otherwise.
premise_check(_,Check) :- true | Check = p.

%%%%%%%%%%  revision
revision_a(no,Env,NewEnv,NodeV,NewNodeV,S-E) :- true |
        Env = NewEnv,
        NodeV = NewNodeV,
        patms_lib:process_end(S,E).
revision_a(a,{env,EnvF,EnvPool},NewEnv,NodeV,NewNodeV,S-E) :- true|
        set_vector_element(NodeV,0,D,assumption(D),NodeV0),
        set_vector_element(NodeV0,1,L,NewLabel,NodeV1),
        set_vector_element(NodeV1,2,JL,NewJL,NodeV2),
        set_vector_element(NodeV2,3,C,NC,NodeV3),
        set_vector_element(NodeV3,4,_,NCd,NewNodeV),
        set_label_go({EnvF},L,JL,{},NewLabel,NewJL,NCd,C,NC,S-Ep),
        patms_environment_table:set_table(EnvF,D,EnvPool,NewEnvPool,Ep-E),
        NewEnv = env(Environment, NewEnvPool),
        patms_label_operation:new_assumption(EnvF, Environment).

revision_p(no,NodeV,NewNodeV,S-E) :- true|
        NodeV = NewNodeV,
        patms_lib:process_end(S,E).
revision_p(p,NodeV,NewNodeV,S-E) :- true|
        set_vector_element(NodeV,0,D,premise(D),NodeV0),
        set_vector_element(NodeV0,1,L,NewLabel,NodeV1),
        set_vector_element(NodeV1,2,JL,NewJL,NodeV2),
        set_vector_element(NodeV2,3,C,NC,NodeV3),
        set_vector_element(NodeV3,4,_,NCd,NewNodeV),
        set_label_go({32#{0}},L,JL,{},NewLabel,NewJL,NCd,C,NC,S-E).

revision_ja(no,Env,NewEnv,Ant,NewAnt,NodeV,NewNodeV,Cont,NewCont,Self,S-E) :- true|
        Self = [],
        Ant = NewAnt,
        Cont = NewCont,
        Env = NewEnv,
        NodeV = NewNodeV,
        patms_lib:process_end(S,E).
revision_ja(ja,{env,EnvF,EnvPool},NewEnv,Ant,NewAnt,NodeV,NewNodeV,Cont,NewCont,Self,S-E) :- true|
        set_vector_element(NodeV,0,D,assumption(D),NodeV0),
        set_vector_element(NodeV0,1,L,NewLabel,NodeV1),
        set_vector_element(NodeV1,2,JL,NewJL,NodeV2),
        set_vector_element(NodeV2,3,C,NC,NodeV3),
        set_vector_element(NodeV3,4,_,NewCd,NewNodeV),
        Cont = {[{show,{label,Ng,S-Ep}}],NewCont},
        set_label_go(NewLabel_,L,JL,Ng,NewLabel,NewJL,NewCd,C,NC,Ep-Ep1),
        NewEnv = env(Environment, NewEnvPool),
        patms_environment_table:set_table(EnvF,D,EnvPool,NewEnvPool,Ep1-Ep2),
```

```
        patms_label_operation:new_assumption(EnvF, Environment),
        patms_lib:connect(Ant, D, Label_ListM, Self, NewAnt, Ep2-Ep3),
        merge(Label_ListM,Label_List),
        NewLabelList=[{EnvF}|Label_List],
        patms_lib:label_updating(NewLabelList, Ng, {}, NewLabel_, _, _, _, Ep3-E).

%%%%%%%%%%   set label

set_label_go(CL,{},_,CNogood,NewL,NewJL,NCd,C,New_C,S-E) :-  vector(CL,Size) |
        NewL = CL,
        NewJL = [[CL]],
        patms_lib:propagation(goahead, C, CNogood, {}, CL, New_C, Ep-E),
        patms_lib:check_contradictory(Size,NCd,S-Ep).
set_label_go({},L,JL,_,NL,NewJL,NCd,C,New_C,SWICH) :-  true |
        C = New_C,
        JL = NewJL,
        vector(L,Size,NL),
        patms_lib:check_contradictory(Size, NCd, SWICH).
set_label_go({32#{0}},L,JL,CNogood,NL,NewJL,NCd,C,New_C,SWICH) :-  true |
        NL = {32#{0}},
        NewJL = [[{{0}}]|JL],
        NCd = 0,
        patms_lib:propagation(goahead, C, CNogood, L, {32#{0}}, New_C, SWICH).
otherwise.
set_label_go(CL,L,JL,CNogood,NL,NewJL,NCd,C,New_C,S-E) :-  true |
        NL = NLabel,
        NewJL = [[CL]|JL],
        patms_label_operation:label_union_list([CL,L],CNogood,NLabel0,_),
        patms_lib:check_and_propagation(L, NLabel0, C, CNogood, New_C, Ep-E),
        vector(NLabel0,Size,NLabel),
        patms_lib:check_contradictory(Size,NCd,S-Ep).

%-------------------------------------------------%
%-------------------------------------------------%
```

4.8 モジュール patms_label_operation

```
%-------------------------------------------------%
% ラベル計算
%-------------------------------------------------%
:- module patms_label_operation.
:- with_macro pimos.
:- public new_assumption/2, env_or/3, env_and/3,
          label_union/4, label_union_list/4,
          super_vs_ng/4, minimal_label/2, complete_label/7,
          append_vector/3, all_label_union/8.

%%%%%% string use
new_assumption(OldAss, NewAss) :-
        string(OldAss,Ssize,_) |
        new_assumption(OldAss,~(Ssize-1),NewAss).
```

```
new_assumption(OldAss,Po,NewAss) :-
        string_element(OldAss,Po,PoE),
        shift_left(PoE,1,NewPoE),
        NewPoE > 0 ,
        NewPoE =< 1073741824 |
        set_string_element(OldAss,Po,NewPoE,NewAss).
otherwise.
new_assumption(_,Po,NewAss) :- true |
        new_string(OldNewAss,~(Po+1),32),
        Ss= 32#{1},
        builtin#append_string(OldNewAss,Ss,NewAss).

all_label_union([JLH|JLT],OldL,NewL,Ng,Seed,NewLabel,NewJLabel,NewNogood) :- true |
        append_vector(NewHLabel,Seed,NewLabelLH),
        minimal_label(NewLabelLH,NewLabelL),
        NewJLabel = [NewJLH|NewJLT],
        re_label_union(OldL,NewL,JLH,Ng,NewHLabel,NewJLH,NewNg),
        all_label_union(JLT,OldL,NewL,NewNg,NewLabelL,NewLabel,NewJLT,NewNogood).
all_label_union([],_,_,Ng,Seed,NewLabel,NewJLabel,NewNogood) :- true |
        Ng = NewNogood,
        Seed = NewLabel,
        NewJLabel = [].

re_label_union(OldL,NewL,JL,Ng,NewLabel,Labels,NewNg) :- true |
        henkan(JL, OldL, NewL, Labels),
        label_union_list(Labels,Ng,NewLabel,NewNg).

%%%%%%%%%% utility

henkan([],_,_,Labels) :-  true | Labels = [].
henkan([OldL|T],OldL,NewL,Labels) :-  true |
        Labels = [NewL|LabelT],
        henkan(T,OldL,NewL,LabelT).
otherwise.
henkan([H|T],OldL,NewL,Labels) :-  true |
        Labels = [H|LabelT],
        henkan(T,OldL,NewL,LabelT).

label_union(LabelV,Ng,UnionL,NewNg) :-
        vector(LabelV,Size) |
        pool:keyed_sorted_set(Set),
        complete_label(1,Size,LabelV,Ng,Set,CompL,NewNg),
        minimal_label(CompL,UnionL).

label_union_list(LabelL,Ng,UnionL,NewNg) :-
        true |
        pool:keyed_sorted_set(Set),
        complete_label_list(LabelL,Ng,Set,CompL,NewNg),
        minimal_label(CompL,UnionL).

complete_label_list([],Ng,Set,CompL,NewNg) :-
```

```
        true |
        Ng = NewNg,
        Set = [],
        CompL = {}.
complete_label_list([L],Ng,Set,CompL,NewNg) :-
        true |
        vector(L,LSize,NewL),
        complete_2_label(0,LSize,NewL,{32#{0}},Ng,Set,CompL,NewNg,[]).
complete_label_list(LabelL,Ng,Set,CompL,NewNg) :-
        LabelL = [L1,L2|T] |
        vector(L1,L1Size,NewL1),
        complete_2_label(0,L1Size,NewL1,L2,Ng,Set,UL,Ng_,NewSet),
        complete_label_list([UL|T],Ng_,NewSet,CompL,NewNg).

complete_label(P,P,LabelV,Ng,Set,CompL,NewNg) :-
        true |
        Ng = NewNg,
        Set = [],
        set_vector_element(LabelV,0,CompL,_,_).
complete_label(_,0,LabelV,Ng,Set,CompL,NewNg) :-
        true |
        Ng = NewNg,
        Set = [],
        set_vector_element(LabelV,0,CompL,_,_).
complete_label(P,Size,LabelV,Ng,Set,CompL,NewNg) :-
        Size >= 2,
        P < Size |
        set_vector_element(LabelV,0,L1,UL,LabelV1),
        set_vector_element(LabelV1,P,L2,_,LabelV2),
        vector(L1,L1Size,NewL1),
        complete_2_label(0,L1Size,NewL1,L2,Ng,Set,UL,Ng_,NewSet),
        complete_label(~(P+1),Size,LabelV2,Ng_,NewSet,CompL,NewNg).

complete_2_label(P,L1Size,L1,L2,Ng,Set,UL,NewNg,NewSet) :-
        P < L1Size,
        vector(L2,L2Size),
        vector_element(L1,P,EP) |
        label_vs_env(0,L2Size,L2,EP,Ng,Set,Ng_,Set_),
        complete_2_label(~(P+1),L1Size,L1,L2,Ng_,Set_,UL,NewNg,NewSet).
complete_2_label(P,P,_,_,Ng,Set,UL,NewNg,NewSet) :-
        true |
        Ng = NewNg,
        Set = [{get_all,ALL}|NewSet],
        change(ALL,0,[],UL).
%%%%%%%
label_vs_env(P,P,_,_,Ng,Set,NewNg,NewSet) :-
        true |
        Ng = NewNg,
        Set = NewSet.
label_vs_env(P,Lsize,L,E,Ng,Set,NewNg,NewSet) :-
        P < Lsize,
```

```
        vector_element(L,P,EP) |
        env_or(E,EP,OEP),
        super_vs_ng(Ng,OEP,CheckedE,Ng_),
        Set = [put(CheckedE,CheckedE,_)|Set_],
        label_vs_env(~(P+1),Lsize,L,E,Ng_,Set_,NewNg,NewSet).
%%%%%%%

minimal_label(CompL,UnionL) :-
        vector(CompL,Size)|
        pool:keyed_sorted_set(Set),
        minimal_label(0,Size,0,CompL,Set,UnionL).
minimal_label(P,P,VS,_,Set,UnionL) :- true |
        Set = [{get_all,ALL}],
        change(ALL,0,[],UnionL).
minimal_label(P,Size,VS,CompL,Set,UnionL) :-
        P < Size,
        vector_element(CompL,P,PE) |
        super_check(0,Size,VS,NewVS,PE,CompL,CPE,NewCompL),
        Set = [put(CPE,CPE,_)|Set_],
        minimal_label(~(P+1),Size,NewVS,NewCompL,Set_,UnionL).
%%%%%%%
super_check(_,_,VS,NewVS,none,CompL,CPE,NewCompL) :-
        true |
        CPE = none,
        NewVS = VS,
        CompL = NewCompL.
otherwise.
super_check(P,P,VS,NewVS,E,CompL,CPE,NewCompL) :-
        true |
        CPE = E,
        NewVS := VS + 1,
        CompL = NewCompL.
super_check(P,Size,VS,NewVS,E,CompL,CPE,NewCompL) :-
        P < Size,
        vector_element(CompL,P,OneE) |
        env_and_eqcheck(E,OneE,AndE,P,CompL,CompL_),
        super_env(AndE,OneE,P,Size,VS,NewVS,E,CompL_,CPE,NewCompL).

super_env(AndE,AndE,_,_,VS,NewVS,_,CompL,CPE,NewCompL) :- true |
        VS = NewVS,
        CompL = NewCompL,
        CPE = none.
otherwise.
super_env(_,_,P,Size,VS,NewVS,E,CompL,CPE,NewCompL) :-
        true |
        super_check(~(P+1),Size,VS,NewVS,E,CompL,CPE,NewCompL).
%%%%%
super_vs_ng(Ng,OEP,CheckedE,NewNg) :-
        vector(Ng,Size) |
        super_check_vs_ng(0,Size,0,_,OEP,Ng,CheckedE,NewNg).
```

```
super_check_vs_ng(P,P,VS,NewVS,E,CompL,CPE,NewCompL) :-
        true |
        CPE = E,
        NewVS := VS + 1,
        CompL = NewCompL.
super_check_vs_ng(P,Size,VS,NewVS,E,CompL,CPE,NewCompL) :-
        P < Size,
        vector_element(CompL,P,OneE) |
        env_and(E,OneE,AndE),
        super_env_vs_ng(AndE,OneE,P,Size,VS,NewVS,E,CompL,CPE,NewCompL).

super_env_vs_ng(AndE,AndE,_,_,VS,NewVS,_,CompL,CPE,NewCompL) :- true |
        VS = NewVS,
        CompL = NewCompL,
        CPE = none.
otherwise.
super_env_vs_ng(_,_,P,Size,VS,NewVS,E,CompL,CPE,NewCompL) :-
        true |
        super_check_vs_ng(^(P+1),Size,VS,NewVS,E,CompL,CPE,NewCompL).
%%%%%%%
env_or(E1,E2,UnionE) :-
            string(E1,S1,_),
            string(E2,S2,_) |
            union_string(S1,S2,E1,E2,UnionE).

union_string(S1,S2,E1,E2,UnionE) :-
        S1 >= S2 |
        unionS(0,S1,S2,E1,E2,UnionE).
union_string(S1,S2,E1,E2,UnionE) :-
        S1 < S2 |
        unionS(0,S2,S1,E2,E1,UnionE).

unionS(P,S1,S2,E1,E2,UnionE) :-
        P < S2,
        string_element(E1,P,P1),
        string_element(E2,P,P2),
        or(P1,P2,UP)|
        set_string_element(E1,P,UP,NewE1),
        unionS(^(P+1),S1,S2,NewE1,E2,UnionE).
unionS(P,_,P,E1,_,UnionE) :- true |
        E1 = UnionE.

%%%%%%
env_and_eqcheck(E,E,IntersecE,P,CompL,NewCompL) :- true |
            set_vector_element(CompL,P,_,none,NewCompL),
            IntersecE = dummy.
env_and_eqcheck(none,_,IntersecE,_,CompL,NewCompL) :- true |
            CompL = NewCompL,
            IntersecE = dummy.
env_and_eqcheck(_,none,IntersecE,_,CompL,NewCompL) :- true |
            CompL = NewCompL,
```

```
            IntersecE = dummy.
otherwise.
env_and_eqcheck(E1,E2,IntersecE,_,CompL,NewCompL) :- true |
            CompL = NewCompL,
            env_and(E1,E2,IntersecE).

env_and(E1,E2,IntersecE) :-
            string(E1,S1,_),
            string(E2,S2,_) |
            intersec_string(S1,S2,E1,E2,IntersecE).

intersec_string(S1,S2,E1,E2,IntersecE) :-
        S1 =< S2 |
        intersecS(0,S1,S2,E1,E2,IntersecE).
intersec_string(S1,S2,E1,E2,IntersecE) :-
        S1 > S2 |
        intersecS(0,S2,S1,E2,E1,IntersecE).

intersecS(P,S1,S2,E1,E2,IntersecE) :-
        P < S2,
        string_element(E1,P,P1),
        string_element(E2,P,P2),
        and(P1,P2,IP)|
        set_string_element(E1,P,IP,NewE1),
        intersecS(~(P+1),S1,S2,NewE1,E2,IntersecE).
intersecS(P,P,_,E1,_,IntersecE) :- true |
        E1 = IntersecE.

%%%%%%%
to_vector(_,_,[],UnionL0,UnionL) :-  true |
        UnionL0 = UnionL.
to_vector(P,VS,[none|T],UnionL0,UnionL) :-
        true |
        to_vector(P,VS,T,UnionL0,UnionL).
otherwise.
to_vector(P,VS,[H|T],UnionL0,UnionL) :-
        P < VS |
        set_vector_element(UnionL0,P,_,H,UnionL1),
        to_vector(~(P+1),VS,T,UnionL1,UnionL).
%%%%%%%

change([{none,_}|T],C,L,UL) :- true |
    change(T,C,L,UL).
otherwise.
change([{_,H}|T],C,L,UL) :- true |
    NC := C + 1,
    NL = [H|L],
    change(T,NC,NL,UL).
change([],C,L,UL) :- true |
    new_vector(UL0,C),
    to_vector(0,C,L,UL0,UL).
```

```
%%%%%%%

append_vector(V1,V2,NewV) :-
    vector(V1,Size1),vector(V2,Size2) |
    Size := Size1 + Size2,
    new_vector(NewV0,Size),
    in(0,Size1,0,V1,NewV0,NewV1),
    in(0,Size2,Size1,V2,NewV1,NewV).

in(P,P,_,V,NewV0,NewV1) :- true | NewV0 = NewV1.
in(P,Size,VP,V,NewV0,NewV1) :-
    P < Size,
    vector_element(V,P,E) |
    set_vector_element(NewV0,VP,_,E,NewV10),
    in(~(P+1),Size,~(VP+1),V,NewV10,NewV1).
%-----------------------------------------------%
%-----------------------------------------------%
```

4.9 モジュール patms_environment_table

```
%-----------------------------------------------%
% 環境管理
%-----------------------------------------------%
:- module patms_environment_table.
:- with_macro pimos.
:- public initial_table/1,set_table/5,search_datums/5.

initial_table(NPool) :- true |
        pool:keyed_sorted_set(Pool),
        Pool =[put(environment,□,_)|NPool].

set_table(Env,Datum,Pool,NPool,S-E) :- wait(Env) |
        Pool = [get_if_any_and_put(environment,{ODatum},[(Env,Datum)|ODatum])|NPool],
        patms_lib:process_end(S,E).

search_datums(Envs,Datums,Pool,NPool,S-E) :- vector(Envs,Size) |
        Pool = [get(environment,ODatum),put(environment,ODatum,_)|NPool],
        search_datums(0,Size,Envs,ODatum,Datums,S-E).

search_datums(P,Size,EnvV,ODatum,Datums,S-E) :-
        vector_element(EnvV,P,H) |
        Datums = [Datum1|Datum2],
        merge(Datum10,Datum1),
        get_j(ODatum,H,Datum10,S-Ep),
        search_datums(~(P+1),Size,EnvV,ODatum,Datum2,Ep-E).
search_datums(P,P,_,_,Datums,S-E) :- true |
        patms_lib:process_end(S,E),
        Datums = □.

get_j([(E1,D1)|TO],H,Datum,S-E) :- true |
        Datum = {Datum1, Datum2},
        patms_label_operation:env_or(H,E1,Tmp),
```

```
        get_datums(Tmp,H,D1,Datum1,S-Ep),
        get_j(TO,H,Datum2,Ep-E).
get_j(□,_,Datum,S-E) :- true |
        patms_lib:process_end(S,E),
        Datum = □.

get_datums(T,T,D,Datum,S-E) :-  true | Datum = [D], S = E.
otherwise.
get_datums(_,_,_,Datum,S-E) :-  true | Datum = □, S = E.
%-----------------------------------------------------%
%-----------------------------------------------------%
```

4.10 モジュール patms_io

```
%-----------------------------------------------------%
% 入出力
%-----------------------------------------------------%
:- module patms_io.
:- public s_o/1,file_input/2.
:- with_macro pimos.

s_o(Out) :- true |
        shoen:raise(pimos_tag#shell, get_std_out, Out).

file_input(FileNameString, IN) :-
        true |
        shoen:raise(pimos_tag#task, general_request, FI),
        FI = [file(normal(FI1, _, _))],
        FI1 = [open(FileNameString, read(normal(FIR,_,_)))],
        buffer:interaction_filter(INTR,FIR),
        INTR = [gett(IN)].
%-----------------------------------------------------%
%-----------------------------------------------------%
```

4.11 モジュール patms_lib

```
%-----------------------------------------------------%
% ライブラリ
%-----------------------------------------------------%
:- module patms_lib.
:- with_macro pimos.
:- public j_terminate/1, c_terminate/1, sound_node/6,
          process_end/2, mmerge/3, mmerge/4,
          connect/6, label_updating/8, check_and_propagation/6,
          renewal/12, propagation/7,
          out/1,out/2,to_list/2, exec_time/3,
          check_old_consq/1, check_contradictory/3.

j_terminate(□) :- true | true.
j_terminate([H|_]) :- integer(H) | true.
j_terminate([[H|TH]|T]) :- true | H = □, j_terminate_s(TH), j_terminate(T).
```

```
j_terminate_s([]) :- true | true.
j_terminate_s([H|T]) :- true | H = [], j_terminate_s(T).

c_terminate(Consq) :- true |  Consq = [get_all(All)],
                              to_list(All,AllConsq),
                              terminate_c(AllConsq).
terminate_c([]) :- true | true.
terminate_c([H|T]) :- true | H = [], terminate_c(T).

to_list([{_,L}|T],CompL) :- true | to_list(T,CompLT),CompL = [L|CompLT].
to_list([],CompL) :- true | CompL = [].
%%%%%
sound_node(end,Sound_Node,premise(D),L,Cd,End) :- true |
        Sound_Node = node(D,L,Cd),
        End = end.
sound_node(end,Sound_Node,assumption(D),L,Cd,End) :- true |
        Sound_Node = node(D,L,Cd),
        End = end.
sound_node(end,Sound_Node,assumed(D),L,Cd,End) :- true |
        Sound_Node = node(D,L,Cd),
        End = end.
sound_node(end,Sound_Node,[assumed,D,_],L,Cd,End) :- true |
        Sound_Node = node(D,L,Cd),
        End = end.
otherwise.
sound_node(end, Sound_Node,D,L,Cd,End) :- true |
        Sound_Node = node(D,L,Cd),
        End = end.

process_end(end,E) :- true | E = end.

mmerge([], Tail, St) :- true | Tail = [], St = [].
mmerge([H|T], Just, [HEE|TEE]) :-
     true |
     Just = [Hj|Tj],
     HEE = {H, Hj},
     mmerge(T, Tj, TEE).

mmerge([], Tail, St, S-E) :- true | Tail = [], St = [], process_end(S,E). %@priority($,-100).
mmerge([H|T], Just, [HEE|TEE], SWITCH) :-
     true |
     Just = [Hj|Tj],
     HEE = {H, Hj},
     mmerge(T, Tj, TEE, SWITCH).

%%%%%%%%%% connect for list
connect([], _, Labels, Self, New_J, S-E) :- true |
      Labels = [], New_J = [], Self = [], process_end(S,E).
connect([Node|JT], D, Labels_List, Self, New_J, S-E) :-
      true |
      New_J = [New_Node|Tail],
```

```
        Self = {MESJ, MEST},
        Labels_List = {[Labels],Eij},
        Node = [add(consequents(D,MESJ,S-Ep)),show(label(Labels,Ep-Epp))|New_Node],
        connect(JT, D, Eij, MEST, Tail, Epp-E).

%%%%%%%%%% connect for list
connect(NodeV, D, LabelsList, Self, NewNewJ, S-E) :-
        vector(NodeV,Size) |
        new_vector(NewJ,Size),
        connect(0,Size,NodeV,D,LabelsList,Self,NewJ,NewNewJ,S-E).

connect(P,P, _, _, Labels, Self, NewJ, NewNewJ, S-E) :- true |
        Labels = [], NewJ = NewNewJ, Self = [], process_end(S,E).
connect(P,Size, NodeV, D, Labels_List, Self, NewJ, NewNewJ, S-E) :-
        P < Size,
        vector_element(NodeV, P, Node) |
        set_vector_element(NewJ,P,_,NewNode,NewJ_),
        Self = {MESJ, MEST},
        Labels_List = {[Labels],Eij},
        Node = {[add(consequents(D,MESJ,S-Ep)),show(label(Labels,Ep-Epp))],NewNode},
        connect(^(P+1),Size, NodeV, D, Eij, MEST, NewJ_, NewNewJ, Epp-E).

%%%%%%%%%% label_updating
label_updating(LabelList, Nogood, Label, NewLabel, AddLabel, NewContradictory, NewNogood, S-E) :-
        true |
        patms_label_operation:label_union_list(LabelList,Nogood,AddLabel,NewNogood),
        check_status(AddLabel, Label, NewLabel, NewContradictory, S-E).

check_status({}, Node_Label, New_Label, Contradictory, S-E) :-
        true |
        vector(Node_Label,Size,New_Label),
        check_contradictory(Size, Contradictory, S-E).
otherwise.
check_status(AddLabel, NodeLabel, NewLabel, Contradictory, S-E) :-
        true |
        patms_label_operation:append_vector(AddLabel,NodeLabel,NewLabel_),
        patms_label_operation:minimal_label(NewLabel_,NewLabelM),
        vector(NewLabelM,Size,NewLabel),
        check_contradictory(Size, Contradictory, S-E).

%%%%%%%%%% check contradictory
check_contradictory(0, Contradictory, S-E) :-
        true |
        process_end(S,E),
        Contradictory = 1.
otherwise.
check_contradictory(_, Contradictory, S-E) :-
        true |
        process_end(S,E),
        Contradictory = 0.
```

```
%%%%%%%%%% renewal
renewal(Nogood, L, [], CL, NL, Conseq,  Contrad, New_Label, NewJL, New_Contrad, New_Conseq, S-E) :-
      true |
      propagation(goahead, Conseq, Nogood, CL, NL, New_Conseq, S-E),
      Contrad    = New_Contrad,
      New_Label  = L,
      NewJL = [].
renewal(Nogood, Label, JL, CL, NL, fail,  _, NewLabel, NewJL, New_Contrad, _, S-E) :-
      true |
      patms_label_operation:all_label_union(JL, CL, NL, Nogood, {}, New_Label_, NewJL, _),
      check_propagation(Label, New_Label_, _, New_LabelC),
      vector(New_LabelC,Size,NewLabel),
      check_contradictory(Size, New_Contrad, S-E).
renewal(Nogood, Label, JL, CL, NL, Conseq,  _, NewLabel, NewJL, New_Contrad, New_Conseq, S-E) :-
      true |
      propagation(Check, Conseq, Nogood, Label, New_LabelC, New_Conseq, Ep-E),
      patms_label_operation:all_label_union(JL, CL, NL, Nogood, {}, New_Label_, NewJL, _),
      check_propagation(Label, New_Label_, Check, New_LabelC),
      vector(New_LabelC,Size,NewLabel),
      check_contradictory(Size, New_Contrad, S-Ep).

check_propagation(L, L, Check, NL) :- true | Check = unnecessary, L = NL.
otherwise.
check_propagation(Label, New_Label, Check, NL) :- true | Check = goahead,
      patms_label_operation:append_vector(Label, New_Label, New_Label_),
      patms_label_operation:minimal_label(New_Label_, NL).

check_and_propagation(L, L, Conseq, _, NewC, S-E) :- true |
            NewC = Conseq,
            process_end(S,E).
otherwise.
check_and_propagation(CL, NL, Conseq, Nogood, NewC, SWICH) :- true |
      propagation(goahead, Conseq, Nogood, CL, NL, NewC, SWICH).

propagation(unnecessary, Consq, _, _, _, New, S-E) :- true |
            New = Consq, process_end(S,E).
propagation(goahead, Consq, Nogood, CL, NL, NewConsq, S-E) :- true |
      Consq = [get_all(ALL)|NC],
      propagation_(ALL,NC,NewConsq,Nogood,CL,NL,S-E).

propagation_([{Key,O}|T], Consq, NewConsq, Nogood, CL, NL, S-E) :-
      true |
      Consq= [put(Key,NewO,_)|NewConsq_],
      O = [add(label(Nogood, CL, NL, S-Ep))|NewO],
      propagation_(T, NewConsq_, NewConsq, Nogood, CL, NL, Ep-E).
propagation_([], Consq, NewConsq, _, _, _, S-E) :- true | Consq = NewConsq, process_end(S,E).

check_old_consq({}) :- true | true.
check_old_consq({Old}) :- true | Old = [].
%%%%%%%%%% exec_time
exec_time(end,StartTime,ExecTime):-
```

```
    system_timer(_,EndTime) |
    ExecTime:=(EndTime-StartTime)/16.

%%%%%%%%%% out(put)
    out(O) :- true |
    patms_io:s_o(Out),
    Out = [putt(O),flush(_)].
    out(end,O) :- true |
    patms_io:s_o(Out),
    Out = [putt(O),flush(_)].

%-----------------------------------------------------%
%-----------------------------------------------------%
```

参考文献

[deKleer 86] deKleer, J. : "An Assumption-based TMS", *Artif. Intell.* , Vol. 28, pp. 127–162(1986).

[Dixon 88] Dixon, M. , de Kleer, J. : "Massively Parallel Assumption-based Truth Maintenance", *Proc. of the AAAI-88*, pp. 199-204(1988).

[Harada 89] Harada, T. and Mizoguch, F. : "Pararell Constraint Satisfaction by Paralleling ATMS", *Proc. of the PRICAI*, pp. 462-455(1990).

[Inoue 88] Inoue, K.: "Problem Solving with Hypothetical Reasoning",*Proc.of the International Conference on Fifth Generation Computer Systems*, Vol.3, pp.1275–1281(1988).

[太田 91] 太田好彦, 井上 克巳: "ATMS を用いた前向き仮説推論システムにおける効率的な推論方式", 人工知能学会誌, Vol.6, No.2, pp. 247 259(1991).

[奥乃 90] 奥乃 博: "網: 新しい ATMS の処理系とその共有メモリ型マルチプロセッサ上での並列処理", 人工知能学会誌, Vol. 5, No. 3, pp. 79-88(1990).

[Rothberg 89] Rothberg, E. and Gupta, A. : "Experiences Implementeting a Parallel ATMS on a Shared Memory Multiprocessor", *Proc. of the IJCAI*, pp. 199-205(1989).

[和田 90] 和田 真一: "疎結合型並列マシンにおける ATMS の並列化方式について", 人工知能学会全国大会 (第 4 回) 論文集, pp. 261-264(1990).

付録 A

ラベル計算例

理由付けの集合 J を以下とし，仮説集合の要素を大文字で表す:

$$\begin{array}{rcccl} & A & \Rightarrow & p & (1) \\ B, & C & \Rightarrow & p & (2) \\ A, & C & \Rightarrow & q & (3) \\ & D & \Rightarrow & q & (4) \\ A, & D & \Rightarrow & \bot & (5) \\ p, & q & \Rightarrow & r & (6) \end{array}$$

p のラベル L_p は (1),(2) の理由付けより，{{A},{B,C}}(論理的には $A \vee (B \wedge C)$) であり，同様に q のラベル L_q は (3),(4) の理由付けより，{{A,C},{D}} (論理的には $(A \wedge C) \vee D$) である．

また (5) より矛盾環境は {A, D} である．このとき，(6) の理由付けにしたがって，r のラベルを求める．

1. L_p と L_q の各環境の和集合に関して直積 L' を求める．

$$L' = \{\{A, C\}, \{A, D\}, \{A, B, C\}, \{B, C, D\}\}$$

2. L' から矛盾環境を包含する要素を削除したのち，その極小集合を求める (L_{new})．

矛盾環境が $\{A, D\}$ であり，$\{A, C\} \subset \{A, B, C\}$ であるから，

$$L_{new} = \{\{A, C\}, \{B, C, D\}\}$$

となる．

付録 B

下層インタフェースメッセージ例

本文3.3において取り上げた男女ペア問題を並列ATMSの下層インタフェースを用いて解決する場合のメッセージ例を示す(使用するプロセッサの数は8とする). メッセージ中の変数はそれぞれノードプロセスに対するストリームを表している.

```
atms:go(M,8,_ExecTime),
M = [
          create-a(pair(a,e),AE),
          create-a(pair(a,g),AG),
          create-a(pair(b,e),BE),
          create-a(pair(b,h),BH),
          create-a(pair(c,h),CH),
          create-a(pair(c,g),CG),
          create-a(pair(d,f),DF),
          create-a(pair(d,h),DH),
          create-n(apair,Apair),
          create-n(bpair,Bpair),
          create-n(cpair,Cpair),
          create-n(dpair,Dpair),
          create-n(pair,Pair),
          ac([AE,BE],[AE1,BE1]),
          ac([BH,CH],[BH1,CH1]),
          ac([AG,CG],[AG1,CG1]),
          ac([CH1,DH],[CH2,DH1]),
          ac([BH1,DH1],[BH2,DH2]),
          aj(Apair,[AE1],Apair1,[AE2]),aj(Apair1,[AG1],Apair2,[AG2]),
          aj(Bpair,[BE1],Bpair1,[BE2]),aj(Bpair1,[BH1],Bpair2,[BH2]),
          aj(Cpair,[CH2],Cpair1,[CH3]),aj(Cpair1,[CG1],Cpair2,[CG2]),
          aj(Dpair,[DH2],Dpair1,[DH3]),aj(Dpair1,[DF],Dpair2,[DF1]),
          aj(Pair,[Apair2,Bpair2,Cpair2,Dpair2],Pair1,[Apair3,Bpair3,Cpair3,Dpair3]),
          t(AE2),t(AG2),t(BE2),t(BH2),t(CH3),t(CG2),t(DH3),t(DF1),
          t(Apair3),t(Bpair3),t(Cpair3),t(Dpair4),t(Pair1) ]
          show-node(pair,PairNode) ].
```