7F−2

拡張前提式を用いたATMSの並列化

○飯塚哲也，西谷泰昭 (群馬大学)

1. はじめに

ATMS[1]は、仮説推論を現在のところ最も適切に定式化した枠組であり、エキスパートシステム等に広く使われている。しかし、仮説として扱われる不完全な知識を組み合わせることにより、組合せ爆発の問題が生じ、実行効率の低化が起こる。そこで、ATMSの並列化により実行効率の向上を図った研究[2][3]が行なわれている。本稿では、理由付けのOR並列性に着目するだけでなく、拡張前提式と呼ぶ論理式を定義することにより、ATMSをより並列化することを考える。

2. 基本の定義

ATMSの理由付けjは、前提ノードの連言と帰結ノードのペアである。理由付けjの前提ノードの集合を$A(j)$で表し、前提ノードの連言を$E(j)$で表す。また、理由付けjの帰結ノードは$C(j)$で表し、帰結ノードがbである理由付けの集合を$J(b)$で表す。ノードの列$a_1, a_2, \cdots, a_m$において、$a_{i-1} \in A(j_i), C(j_i) = a_i$である$j_i$が存在するとき、この列を$a_1$から$a_m$へのパスといい、$m$をそのパスの長さという。

ノードbへ長さk以下のパスが存在するノードの集合を$B^*(b,k)$、bへ長さkのパスが存在するノードの集合を$B(b,k)$と定義する。集合B^*に含まれるノードを祖先ノードと呼ぶ。祖先ノードは、真偽が依存しているノード、つまり、ラベルが依存しているノードを表す。次のようにして$B(b,k), B^*(b,k)$を計算することができる。

$$
\begin{aligned}
B(b,0) &= \{b\} \\
B(b,k) &= \cup_{j \in J(b)} (\cup_{a \in A(j)} B(a, k-1)) \\
B^*(b,0) &= \{b\} \\
B^*(b,k) &= B(b,k) \cup B^*(b,k-1)
\end{aligned}
$$

B, B^*の引数のkのことを段と呼ぶ。段はあるノードから他のノードへ理由付けがいくつ離れているか示す。

F, F^*はそれぞれB, B^*の逆であり、集合F^*に含まれるノードを子孫ノードと呼ぶ。子孫ノードはノードのラベルに変更があったとき、ラベル計算が伝播されるべきノードを表している。

3. 拡張前提式

従来のATMSでは、理由付けは帰結ノードとそのノードのラベルに直接影響を与える前提ノードを持っているだけである。そのため、前提ノードのラベルに変更があった後でないと、帰結ノードのラベル計算を行なうことはできない。しかし、ラベル計算に影響を与える祖先ノードとその組合せを各ノードが何らかの形で持ち、それを利用してラベルを計算することができるなら、直接影響を与える前提ノードのラベルの計算を待つことなく、ラベルを計算することができる。

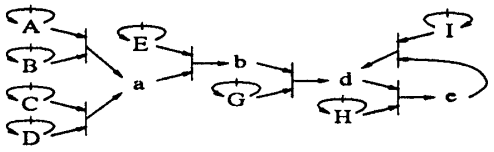
本稿では、祖先ノードを、各ノードに拡張前提式という形で持たせることによりこれを可能にする。拡張前提式は、あるノードへ同じ長さのパスを持つ祖先ノードからなり、そのノードを論理的に包含する論理式である。

$EA(b,k)$は$B(b,k)$内のノードからなる論理式であり、ノードbのk段の拡張前提式と呼ぶ。以下で定義される論理式を拡張前提式と呼ぶ。

$$
\begin{aligned}
EA(b,0) &= b \\
EA(b,k) &= EA(b,k-1)^{a \in B(b,k-1)}_{\vee_{j \in J(a)} E(j)}
\end{aligned}
$$

$F^{x \in Y}_{f(x)}$は、F中の変数xのすべての出現を$f(x)$で置き換えることをY中のすべての変数について同時に行なう。

図1に拡張前提式の例を示す。図中の大文字は仮説ノードを表し、小文字は仮説以外のノードを表す。



$$
\begin{aligned}
EA(b,2) &= EA(b,1)^{x \in B(b,1)}_{\vee_{j \in J(x)} E(j)} \\
&= (a \wedge E)^{x \in \{a,E\}}_{\vee_{j \in J(x)} E(j)} \\
&= (A \wedge B \vee C \wedge D) \wedge E
\end{aligned}
$$

図1 理由付けと拡張前提式

4. 拡張前提式の計算とラベル

以下ではnogoodがないと仮定する。ノードbの拡張前提式中に含まれるすべてのノードをそのノードのラベルで置き換えた式は、bのラベル$l(b)$に論理的に等しい。

$$
EA(b,k)^{a \in B(b,k)}_{l(a)} \equiv l(b)
$$

この式が成り立つことから、もし$B(b,k)$内のノードがすべて仮説ノードであれば、$EA(b,k)$はbのラベルに論理的に等しい。このことから、$EA(b,k)$内のノードがすべて仮説ノードになるまでkを増やしながら拡張前提式を計算していけばラベルが得られる。

しかし、理由付けのループが存在すると、拡張前提式内のノードがすべて仮説になることはないため、拡張前提式をどこまで計算すれば良いかということが問題となる。そこで“$B^*(b,k) = B^*(b,k+1)$となるまで拡張前提式$EA(b,k)$を計算する”という条件を考える。ATMSで扱うノードの数は有限であるため、祖先ノードの集合B^*も有限である。そのため、必ず$B^*(b,k) = B^*(b,k+1)$なるkが存在する。bの最終的に得られた拡張前提式において、式中の仮説以外のノードを偽として得られる式は、bのラベルと論理的に等しい。

Parallel ATMS based on extended justifications

Tetsuya Iiduka , Yasuaki Nishitani

Gunma University

nogoodがあるときも、拡張前提式からラベルを得ることができる。nogoodはfalseノードのラベルとして扱う。各ノードの最終的に得られた拡張前提式に、次の3つのことを行なって得られた式は、各ノードのラベルに論理的に等しい。1.選言標準形にする。2.仮説以外のノードを偽にする。3.nogoodを含んでいる連言を取り除く。

以下、EAの計算について述べる。EAの計算を効率的に行なうため、拡張前提式の計算は、2^i段毎に行なえばよい。

$$
\begin{aligned}
EA(b,0) &= \mathrm{b} \\
EA(b,2^0) &= \vee_{j\in J(b)} E(j) \\
EA(b,2^i) &= EA(b,2^{i-1})^{a\in B(b,2^{i-1})}_{EA(a,2^{i-1})}
\end{aligned}
$$

1段づつ拡張前提式を計算したときk回計算が必要であるとすると、この式で計算する場合には、$\log k$回の計算で済む。この式で拡張前提式を計算するときの停止条件は、$B^*(b,2^i)=B^*(b,2^{i+1})$である。$B, B^*, F, F^*$も$EA$と同様に、$2^i$段は$2^{i-1}$段から計算できる。

B, B^*は拡張前提式を計算するために必要であり、F, F^*は次節で計算を並列化するときに、拡張前提式の計算を伝搬するために必要である。図1におけるノードdに対する拡張前提式の計算の例を示す。

k	$EA(d,k)$
0	d
2^0	b∧G∨I∧e
2^1	E∧a∧G∨I∧d∧H
2^2	E∧(A∧B∨C∧D)∧G∨I∧(E∧a∧G∨I∧d∧H)∧H

$B^*(d,2^2)=B^*(d,2^3)=\{A,B,C,D,E,G,H,I,a,b,d,e\}$であるため、$k=2^2$で拡張前提式の計算は停止する。

図2 拡張前提式の計算

以上をまとめると、$EA(b,2^i)$は、$EA(b,2^{i-1})$と、ノード$a\in B(b,2^{i-1})$の$EA(a,2^{i-1})$から計算できることがわかる。このため、すべてのノードの2^{i-1}段の拡張前提式が正しく計算されていれば、各ノードで2^i段の拡張前提式を並列に計算できる。

5. 並列化

ATMSの並列化に関して、理由付けのOR並列性に着目した研究がある[3]。これは各ノードを1つのプロセスに対応させ、理由付けを基にネットワークを構築する。そして、各プロセスが独立に再ラベル計算の指示や、ラベルといったメッセージを他のプロセスとやりとりすることにより、各プロセスでラベルを並列に計算する。この並列化では、あるノードのラベルに変更があっても、そのノードを前提に持つ理由付けの帰結ノードのラベルに関し並列に計算が行なわれるだけである。

直接の帰結ノードだけでなく、直接理由付けに現れない子孫ノードに関しても並列にラベルを計算することができれば、さらにATMSを高速化することができる。拡張前提式を利用することで、この並列化を実現できる。

本稿でも、各ノードを1つのプロセスに対応させる。2つのノードの間に2^iの長さのパスが存在すれば、対応する2つのプロセスは結合している。プロセスは、その時点で正しい2^i段のEA, B, B^*, F, F^*を保持しており、理由付けの追加があったとき、それらをプロセス間で並列にやりとりし、新しく計算することにより、ラベルを計算する。

理由付けが新たに与えられたとき、拡張前提式の計算がどのようにノードに伝搬されるかについて説明する。

新たに理由付けが与えられたノードをaとする。まず、時刻0でaの2^0段の拡張前提式$EA(a,2^0)$の計算を行なう。計算が終ると、時刻1でノードaの2^1段の拡張前提式と、2^0の長さのパスで結合している子孫ノードの2^1段の拡張前提式を計算させる。このとき、子孫ノードへの計算の伝搬に$F(a,2^0)$が使われる。また、2^1段の拡張前提式の計算を終えた各ノードは、時刻3で自分自身と、2^1の長さのパスで結合している子孫ノードの2^2段の拡張前提式を計算させる。

このように、一般的に2^i段の拡張前提式を計算したノードは、自分自身の2^{i+1}段の拡張前提式と、2^iの長さのパスで結合している子孫ノードの2^{i+1}段の拡張前提式を計算させる。拡張前提式の計算の伝搬を以下の図3で表す。新たに理由付けを与えられたノードが図中の0段上にあるとし、そのノードから伝搬が始まる。

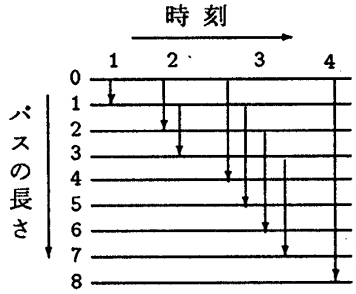


図3 拡張前提式の計算の伝搬

理由付けが与えられたノードに長さkのパスで結合しているノードは、時刻$\lfloor \log k \rfloor + 1$で拡張前提式の計算が伝搬され、計算が始まる。$EA$を矛盾なく計算していくためには$B, B^*, F, F^*$が必要であるため、それらも$EA$と一緒に矛盾なく計算していく必要がある。

6. まとめ

祖先ノードのラベルに変更があったとき、直接の前提のラベルの変更を待つことなく、ラベルの計算が行なえるように拡張前提式を提案した。この拡張前提式を各ノードに持たせ、これを用いてラベルを計算することにより、理由付けのOR並列によるノードだけでなく、さらに多くのノードのラベルを並列に計算することができる。

本研究では、直接ラベル計算に関係ある拡張前提式の計算以外に、祖先ノード、子孫ノードの計算も行なっているため、その計算にかかるオーバーヘッドが問題となる。本研究でどの程度、並列性が高まったかということの評価はこれから行なうつもりである。

参考文献

[1] de Kleer , J. : "An Assumption-based Truth Maintenance System" , Artificial Intellegence , 28 , pp.127-162 (1986)

[2] Michael Dixon and de Kleer , J. : "Massively Parallel Assumption-based Truth Maintenance" , Proc. of AAAI '88 , pp.199-204 (1988)

[3] 原田拓 , 溝口文雄 : "ATMSの並列化に関する一提案", 情報処理学会第40回全国大会 , pp.186-187 (1990)