知識工学と人工知能 53 － 1
（１９８７．７．１０）

ＴＭＳの統合的自然言語理解への応用に関する考察

劉　学敏　西田豊明　堂下修司
京都大学・工学部・情報工学教室

自然言語を理解においては，解析において得られた部分的な情報から新たな論理的結論を導出したり，必要に応じて仮定を作ったり，仮定に矛盾が生じたとき新の仮定に切り換えたりすることによって，推論過程を管理する機構が必要である．このような管理機構として，ＴＭＳとＡＴＭＳが注目される．

本論文では，まず，ＴＭＳとＡＴＭＳのそれぞれについて，その自然言語理解への応用に関して考察し，次に，ＴＭＳとＡＴＭＳを併用し，確からしさに基づいた統合的自然言語理解の方法を提案する．この方法では，ＴＭＳを利用して推論を確からしい方向へ導き，矛盾した仮定の棄却を行う．また，ＡＴＭＳを利用して，信念空間の無矛盾性を維持する．これによって，処理を効率的に行うことが追求する．

TOWARDS INTEGRATED UNDERSTANDING OF NATURAL LANGUAGE BASED ON TMS

Xuemin LIU , Toyoaki NISHIDA and Shuji DOSHITA
Department of Information Science
Kyoto University, Kyoto,JAPAN

In the case of integrated understanding of natural language, we need a subsystem to keep our belief space from being contradictory. Since TMS and ATMS have the power of maintaining belief space consistently, we would expect to use one of them for our reasoning system. But because of the ambiguity of the natrual language, if we simply use one of them, we may be faced with a great deal of trouble.

In this paper, we first discuss the problems arising in the situation in which we use TMS or ATMS as the reasoning management system, then we give a method which uses both of them for understanding of natural language. In this method, we use ATMS to maintain the consistency of our nodes, and use TMS to decide the environment in which the current reasoning process is switched over to an alternative, or the current environment becomes contradictory. In this way, we can expect to make our work efficiently.

1　はじめに

自然言語には非常に多種多様な曖昧性が含まれる．このため単語解析，構文解析，意味解析と逐次的な処理を行うと解析の初期階段で爆発的に多くの曖昧性が生じる[4]．この問題を解決するには，三つのレベルの処理を並行して統合的な処理を行うことが望ましいと考えられる．統合的な処理を実現するためには，解析において得られた部分的な情報から新たな論理的結論を導出したり，必要に応じて仮定を作ったり，仮定に矛盾が生じたとき新の仮定に切り換えたりすることによって，推論過程を管理する機構が必要である．このような管理機構として，TMSとATMSが注目される．

しかし，TMSあるいはATMSは推論過程を確からしい方向に導く機能を持っていない．本論文では，TMSとATMSを併用し，システムに解析過程を確からしい方向に導く機能を追加した自然言語理解の方法を提案する．この方法では，TMSを利用して推論を確からしい方向へ導き，矛盾した仮定の棄却を行う．また，ATMSを利用して，信念空間の無矛盾性を維持する．これによって，無駄な処理の回避，所要記憶領域の節減，処理速度の向上などをはかる．

2　TMSを利用した自然言語解析

2．1　TMS（Truth Maintenance System）

TMS[3]はJ.Doyle によって提案された整合性を維持する推論管理システムである．節点とその節点に付加された理由付け(justification) を基本的なデータ構造として，整合性が保たれるように節点の集合を管理する．

節点は一つの信念(belief)を表す．全ての節点はIN（現在この信念を信じている）かOUT（信じていない）のどちらかの状態と，その状態を決定するための理由付けの集合を持つ．システムがある節点を信じるための理由付けはinlistとoutlistという節点リストを用いる．全てのinlistの節点の状態がIN，かつ全てのoutlistの節点の状態がOUTである場合，この理由付けが有効であるという．有効な理由付けを持つ節点のみの状態をINにする．また，outlistの空でないIN節点を仮定節点と呼ぶ．

ユーザがある節点に対して，一つの理由付けを与えると，TMSはその節点とそれに関連する節点の状態を整合性が維持されるように修正し，節点の状態の変化をユーザに報告する．更に，節点の状態の変化によって，指定した操作を起動することができる．また，矛盾が起きたとき，TMSは自動的にbacktrckingを行い，矛盾の原因となった仮定を捜し，それをOUTにすることによって，矛盾を解消する．

2．2　TMSによる曖昧性の解決

本論文では，説明の便宜上，

$$X_1,\ldots,X_k \text{ ---> } N$$

のような式で，論理的な推論を表し，また，

$$X_1,\ldots,X_k \text{ ---> } N_1,\ldots,N_m$$

のような式で曖昧性のある推論を表す．

曖昧性が起きたとき，システムは各々の可能性に対して，仮定を生成し，それらに対して，確からしさの高い順に従ってINになるように，理由付けを行う．例えば，次のような曖昧性の処理を考えよう．

例１．　wd_1　--->　noun_1, adj_1

ここで，wd_1は「あかい」と言う文字列からなる単語であり，noun_1とadj_1はそれぞれ固有名詞「赤井」と形容詞「赤い」である．「あかい」という文字列からなる単語は「赤井」と「赤い」の二つの可能性がある．

システムはまず二つの可能性に対応して，仮定節点A1（wd_1が名詞という仮定），A2（wd_1が形容詞という仮定）を作る．次に，システムはこの時点で，仮定A1の可能性が高いと信じているものとする．このとき，与える理由付けは次のようなものである．

節点名	理由付け
A1	(SL () (not_A1))
A2	(SL (not_A1) (not_A2))
noun_1	(SL (wd_1 A1) ())
adj_1	(SL (wd_1 A2) ())

ここで，A1の理由付けは「節点not_A1を信じていない限りに節点A1を信じる」という意味であり，A2の理由付けは「not_A1を信じ，かつnot_A2を信じていない場合にA2を信じる」という意味である．

これによって，節点A1とnoun_1 はまずINになる．その後矛盾が見つかれば，矛盾を解消するためTMSは矛盾の原因となる仮定A1を見出す．節点A1の依存するOUT節点not_A1に有効な理由付け例えば（SL () () ）を与える．すると，not_A1とA2またはadj_1がINになり，A1とnoun_1がOUTになる．

その後さらに，節点 A1,A2がすべてOUTになり，節点

not_A2がINになったとすると，wd_1からどの節点も推論できないことになるので，wd_1をOUTにしなければならない．このために，システムは仮定節点を作った後，wd_1が依存するOUT節点（例えばnot_wd1）に

（SL（not_A2）（））

というよう理由付けを与える．これによって，TMSは節点not_A2をINにすると同時に，節点wd_1をOUTにする．

２．３　TMSを用いた処理方法の検討

この方法では，常に確からしさの高い仮定を採用して，推論を進めるので，無駄な推論を避けることができる．また，推論の過程において作られる節点の数は少ない．

しかし，この方法には次のような問題がある．

TMSでは矛盾が起きたら，それを解消するために，矛盾の原因となった複数の仮定のうち，どれか一つをOUTにしなければならない．しかし，その後，この節点を再びINにする必要があれば，新しい理由付けをTMSに付加しなければならない．このように，処理の過程において，一つの節点に対して，ほぼ同じような整合性維持の操作が何度も行われてしまうので，効率が悪くなる．これを説明するため，次の例を考えよう．

例２．「彼は教室にいる」という文の意味解析．

登場人物はA君,B君,C君の三人で，教室には第１講義室，第２講義室，第３講義室があり，正しい意味は「C君が第３講義室にいる」であるとする．この文では，「彼」と「教室」の各々に対して，具体的な対象を言明していないから，意味表現の曖昧性が生じている．

この場合，システムはまず「彼」の指示対象のとして，三つの仮定A,B,C（A君，B君，C君）を作り，「教室」の指示対象として，仮定R1,R2,R3（第１講義室，第２講義室，第３講義室）を作る．システムが上の仮定をその順に信じるものとする．AがINになる場合，仮定R1,R2,R3に理由付けを与える．結局，どれも矛盾するから，仮定R1,R2,R3の全てをOUTにしなければならない．次に，仮定BをINにして，推論を行うと，同じことが起こる．最後に，仮定CをINにするときR1,R2,R3に対して，三回目の理由付けを付加する必要がある．このように繰り返し理由付けを与えることは効率が悪い．もう一つの問題は，仮定を設定するときに，各仮定の間の依存性を考える必要があるため，節点の間の関係が複雑になることがある．例えば例２の場合，システムは仮定A,B,Cの間の依存性，仮定R1,R2,R3の間の依存性，更に，仮定A,B,CとR1,R2,R3の間の依存性を考えなければならない．

また，矛盾があった場合に，仮定を棄却するにも問題がある．矛盾となった原因が複数個の仮定の組合せであるとき，例えば上のAとR1の組合せが矛盾した場合，どれを棄却するかを決定するのが難しい．

３　ATMSを利用した自然言語解析

３．１　ATMS（Assumption-based TMS）

de KleerによってATMS[2,3]はTMSと異なって，全ての節点は理由付けの集合のほか，labelというデータ構造を持つ．labelは環境（仮定節点の集まり）の集合で，その節点が信じられる最小環境の全てを記憶する．仮定節点は節点自身がそのlabelに出てくるような節点である．labelによって，各環境に対応するcontext，即ちその環境で成立する全ての節点の集合を決定できる．

ATMSは節点の理由付けの集合を用いて，そのlabelを計算する．また，labelには矛盾する環境を含まないようにするため，ATMSはnogoodというデータ構造を用いて，矛盾する環境を記憶する．ユーザがある節点に対して一つの理由付けを与えると，ATMSはこの節点とその論理的帰結となる節点のlabelを再計算する．また，ある環境が矛盾すれば，これをnogoodに記入し，全ての節点のlabelからこの環境を削除する．

３．２　ATMSによる曖昧性の解決

ATMSでは，全ての環境を同時に考えることができるので，曖昧性が発生したとき，各仮定を相互に依存しないように設定することができる．

上の例１について説明しよう．システムは

wd_1 ---> noun_1,adj_1

に対して，節点A1とA2を作る．そして，ATMSに次の理由付けを与える．

A1 --->A1　　A1のlabel＝{{A1}}

A2 --->A2　　A2のlabel＝{{A2}}

[注：この二つの理由付けによって，仮定節点A1,A2が作られる．]

wd_1, A1 ---> noun_1

wd_1, A2 ---> adj_1

wd_1のlabelを {E} とし，そして，E1＝E∪{A1}またE2＝E∪{A2}とする．上の理由付けを与えると，節点noun_1とadj_1のlabelはそれぞれ {E1} と {E2} にな

る．E1とE2はともに推論できる環境である．

あとの推論は例えばまず環境E1において行う．もしこの環境で矛盾が起きたら，この環境が矛盾したことをATMSに通知する．すると，ATMSはこの環境を削除する．その後の推論は，環境E1において不能になるが，環境E2の推論は続行される．

３．３　ATMSを用いた処理方法の検討

この方法では，各仮定を依存しないように設定できるため，システムが簡単に実現できる．また，矛盾が起きた時，TMSのようなbacktrackingの必要がなく，矛盾する環境を記憶すればよい．従って，処理の速度はTMSの場合に比べて速い．特に，矛盾が起きても，前の処理結果は失われず，他の環境で直ちに利用できるので．処理の効率の向上が期待できる．

ATMSを用いた処理方法には次の問題がある．

まず，多くの環境があるので，いまの環境が矛盾したとき，次にどの環境で推論を進めるかを決定するのは困難である．盲目的に一つの環境を選出すれば，無駄な推論が多い．もう一つの問題は多くの矛盾する環境を記憶する必要がある．nogoodは処理の進行に伴って，単調に増大する．これは記憶容量の問題だけでなく，処理の速度に大きな影響を与える．ATMSがlabelを計算する場合に，nogoodを参照する必要があるため，nogoodが大きくなると処理の速度は遅くなる．また，人力に誤りがあって正しい解釈が得られない場合，処理を終了するかを判定することが難しい．ATMSの節点にはINとOUTという概念がないので，推論は一般的にlabelの空でない節点に対して行う．しかし，人力に誤りがあっても，全ての節点のlabelが空になることはほぼ不可能である．

４　ATMSとTMSを併用する方法

基本的には，ATMSのほうがTMSより速度が速く，システムが簡単に実現できるという利点を持っているので，ATMSを推論管理機構として自然言語理解システムを作るのが望ましい．しかし，３．３節で述べたATMSの場合の問題を解決しなければならない．

本論文では，ATMSとTMS を併用して，ATMSを節点の管理機構とし，TMS を推論環境の管理機構として，確からしさに基づいた自然言語理解システムを作る方法を提案する．

４．１　環境の分類

ATMSを利用する場合，仮定の数をｎとすると環境の数は2^nに等しいため，全ての環境を考えることは不可能であり，また，その必要性もない．このとき，考える必要がある環境はどのような環境であるかが問題になる．これに対して，まず環境の分類について考えよう．

ATMSの仮定からなるすべての環境は次の四種類に分類できる．

①先天的に矛盾する環境

ある節点から排他的に複数の可能性が導ける場合，システムはそれに対して，複数個のATMS仮定を作る．これらの仮定は同時に信じることができないため，これらの仮定の二つ以上を含む環境は先天的に矛盾する．

例えば，２．２節の例１において，

```
wd_1 ---> noun_1,adj_1
```

の場合，仮定A1とA2とを同時に信じることは出来ないから，環境{A1,A2}は先天的に矛盾する．

②解を含まない環境

これは先天的に矛盾する環境ではないが，解を含まないことが容易に判明できるものである．

２．３節の例２を考えよう．「彼」に対して，三つの仮定A,B,Cが作られる．もしある環境にはA,B,Cのどれも含まれないならば，この環境に解が存在しないことは明らかである．

③推論の途中で矛盾が生じた環境

これは上の二種類の環境と異なって，推論の途中でこの環境から矛盾が出てくるものである．本論文では，「矛盾する環境」とは一般にこの種の環境を指す．

④現時点で推論可能な環境

上のものを除いて，残ったものは現時点で「推論可能な環境」と呼ばれる．

ATMS環境のほとんどは上の種類①と種類②の環境である．もしシステムが推論環境として，種類③と種類④の環境だけを取り出すことができれば，処理すべき環境の数は大幅に減少する．これを実現するため，次に述べる環境の木を用いる．

４．２　環境の木

自然言語理解を行う過程の推論は大別すると，曖昧性のない推論と曖昧性のある推論の二種類がある．曖昧性のない推論は論理的な推論であり，新しい仮定を作る必要がないため，推論は現在の環境の中で行えばよい．しかし，曖昧性のある推論に対しては，新しい

仮定を作り，それ以後の推論を新しい環境のもとで行わなければならない．新しく作られた仮定をK個とすると，それぞれをもとの環境に加えて，K個の新しい推論環境が得られる．これらをもとの環境の子孫環境と呼ぶ．図１に環境の木を示す．［ここで，レベルは環境の中の仮定の数を表す．また，親環境⊂子孫環境であることに注意されたい．］

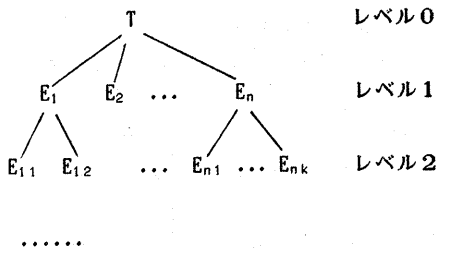

図１　環境の木

環境の木の中に種類①と種類②の環境は存在しないので，推論環境の選択はこの木において行えばよい．無矛盾な解釈が存在するならば，これは環境の木の一つの節点に対応した環境から導出できる．従って，自然言語を解析することはこのような木の中に，解を含む環境を捜すことに相当する．

４．３　矛盾のある環境の併合

システムは種類④の環境のどれかで推論を行う．また，種類③の環境が節点のlabelに出てこないようにするため，この種類の環境をnogoodに記憶する．しかし，これだけでは，システムは全ての矛盾した環境を記憶しなければならないから，まだ効率がよくならない．これを改善するために，矛盾した環境の併合を考える．

ある環境Eにおいて，曖昧性が生じ，仮定A1,...,Anが作られたとすると，次のように，Eからｎ個の子孫環境が得られる．

$$E \xrightarrow{+\{A_i\}} E_1, E_2, \ldots, E_n$$

ここで，$E_i = E \cup \{A_i\}$　i=1,2,...,n

もし全ての子孫環境Eiが矛盾すれば，親環境Eも矛盾することがわかる．この場合，$E_1, E_2, \ldots, E_n$を記憶せず，Eだけを矛盾した環境として記憶すれば，矛盾環境の記憶量を減らすことができる．TMSは節点の状態の変化に応じて，節点に付加された操作を自動的に実行することができる．この性質を活用し，TMSによって矛盾のある環境の併合を自動的に行う．

環境の木の任意の環境Eに対して，一つのTMS節点NEを作る．これはEが矛盾する環境であることを表す．環境Eが矛盾する時，NEがINになるようにするため，

(SL () ())

という理由付けを付加する．また，

$$E \xrightarrow{+\{A_i\}} E_1, E_2, \ldots, E_n$$

に対して，節点NEに

(SL (NE_1, NE_2, ..., NE_n) ())

というような理由付けを付加する．これによって，もし$NE_1, NE_2, \ldots, NE_n$が全部INになれば，NEもINになる．さらに，節点NEに操作を付加する．NEがINになるとき，TMSは自動的にこの操作を実行し，NEに対応する環境が矛盾のあったことをATMSに知らせ，削除させる．

４．４　推論環境の選択

ATMSを利用する場合のもう一つの問題は，推論環境が矛盾した時，推論を続けるために，次の推論環境を選択することがある．盲目的に次の環境を選択すると無駄な推論が増えるので，最も確からしい環境を次の推論環境として選出することが望ましい．このために，確からしさを用いて推論過程を誘導することが必要である．

４．４．１　確からしさの導入

環境を選択する際に，仮定節点だけを考える必要があるため，確からしさは仮定節点だけに対して導入すればよい．このために，仮定が生成されると，システムは知識と今まで得られた推論結果によって，各仮定節点に対して，重み（確からしさを表す正数）を与える．ここでは，次のようにして重みを決定する．

単語の語境界に関して曖昧性があれば，単語の長さによって，重みを決定する．日本語では，短い単語は他の単語のプレフィックスとなる可能性が高いから，最も長いものに最大の重みを与えるほうがよいと考えられる．

単語の概念の曖昧性が生じた場合，文法的知識とそれまでの推論結果を参照して，期待する構文要素に照合する仮定節点に最大の重みを与える．

また，構文要素の意味の曖昧性があったときは，対象世界に関する知識によって，重みを決定する．例えば，作られた諸仮定のうち，現在の環境に一番近いも

のの重みを最大にする．このために，Waltzらのマイクロ素性[5]を用いる方法が考えられる．

仮定が矛盾すると，その重みをゼロにする．その後の環境を選択するときにこれを利用できるようにするため，この仮定と同時に作られた諸仮定の重みを再決定する（重みを事前確率から事後確率へ変化させる）．

４．４．２　環境を選択する基準

推論環境の選択は環境の木の端末節点に対して，MINIMAX 規則に従って行う．即ち，環境の重みをその中の仮定の重みが最小のものと等しくし，重みが最も高いものを次の推論環境として選択する．このような環境が複数個あるときは，環境のレベル（仮定の数）によって選択する．一般的に，仮定の数が多い環境は矛盾が発見される可能性が高いので，レベルの高い環境を選択することによって，矛盾を早く発見することが期待できる．

４．５　TMSとATMSを併用する方法の利点

TMSとATMSを併用する方法ではATMSの場合の欠点を克服，利点を継承することができる．ATMSの場合と比べると，処理対象となる環境の数はかなり少なくなり，矛盾が起きても，いままでの推論結果を失わず，TMSの場合のような同じ推論の重複を避けることができる．

また，確からしさによって，常に最も確からしい環境で推論を行うので，処理速度をATMSの場合より向上することが期待できる．矛盾が起きた場合，推論を他の環境に移して続ければよいから，矛盾の原因の捜索，矛盾した仮定の削除などの時間がかかる処理は不必要になる．もう一つの利点として，終了条件の判定が容易になる．この方法では，推論は種類④の環境がある限り進める．もし種類④の環境がなくなれば，入力には誤りがあるので，処理を失敗させて終了する．

５　自然言語理解システムの設計

５．１　システムの概観

TMSとATMSを併用した統合的な自然言語理解システムの構造を図２に示す．

システムはローマ字列で入力した自然言語の意味解釈を意味表現のリストの形で出力する．

ここでは，知識として，①日本語の辞書，②構文的知識（日本語の文法），③対象世界に関する知識などを利用する．システム自身はこれらの知識の表現に依存しない．但し，曖昧性に関する知識は処理の過程において変化しないことにする．これを満足しなければ，矛盾する環境の併合が実現できない．

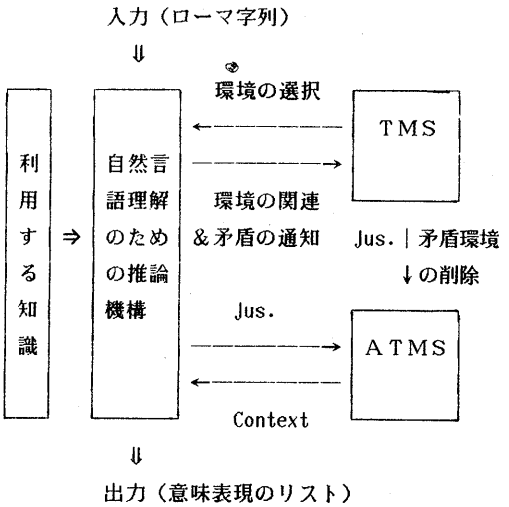

図２　システムの構造

５．２　節点の拡張

節点の拡張は現在の環境のcontextの中で行われる．節点を拡張する際に，システムは同じ信念に対して，二つ以上の節点を作らないことを保証しなければならない．また，異なった節点に対しては，拡張の操作や利用する知識が違う．節点を拡張する操作について説明する前に，まず，システムの処理過程において作られる節点の種類について述べる．

５．２．１　節点の種類

①　文字節点と隣接関係節点

システムは処理の過程で，直接に入力した文字列を参照するのではなく，各文字ごとに，作った文字節点と文字節点の隣接関係を表すための隣接関係節点を参照する．これはシステムを誤りの含んだ入力の処理へ拡張することを考慮したものである．[6]

②　単語節点

この節点には，単語を構成するための文字節点を語境界情報として記録する．

③　構文要素節点

この節点には，構文要素の種類と，それを生成するための単語あるいは構文要素を必要な情報として記録する．

④　意味表現節点

この節点には，意味表現の種類と，それを生成するための構文要素を必要な情報として記録する．

５．２．２　節点を拡張するための基本的な操作

①　単語の発見

文字の隣接関係を用いて，辞書を調べ，入力文字列のある文字から始まる全ての単語を見出す．

語境界の曖昧性のため，一つの文字を先頭文字として複数個の単語を見出すことがある．それらの単語の各々に対して，仮定を生成し，知識によって仮定節点に重み（確からしさ）を与える．また，指定した文字から，何の単語も発見できない場合に，先行する単語の誤境界が誤っていることがわかるので，矛盾を報告する．

②　単語の認識

辞書を参照して，一つの単語節点から構文要素節点への拡張を行う．曖昧性があれば，まず，仮定節点を作り，文法知識とマイクロ素性によって，これらのうち，いまの環境に最も近い節点に高い重みを与える．

③　構文解析

構文知識と隣接関係，及びいままで作られた構文要素節点を利用して，一つの構文要素節点から新しい構文要素節点への拡張を行う．隣接した二つ（あるいはその以上）の構文要素が文法に一致しない場合に，矛盾を報告する．

④　意味解析

対象世界に関する知識を利用して構文要素節点から意味表現節点への拡張を行う．

一つの構文要素節点から複数個の意味表現節点が導かれる場合，仮定を生成する．そして，対象世界に関する知識によっていまの環境に最も近い仮定に最大の重みを与える．名詞の指示対象が対象世界に存在しないとか，動詞の指示動作が対象世界において実行できない（動作不能あるいは許されない）などのように，作られた意味表現節点が対象世界の知識に一致しないならば，矛盾を報告する．

このような操作を利用して，システムは与えられた知識と事実から，推論を行う．曖昧性が生じたら，仮定を生成し，推論をもっとも確からしい仮定を含む環境に移して続ける．また，矛盾が生じたら，それをTMSに知らせる．TMSは環境の木から新しい推論環境を選出する．このようにして，正しい意味解釈が得られるまで，または環境の木が空になるまで処理を続けて，自然言語の理解を行う．

５．３　実行例

以下に一つの例についてシステムの動作を説明する．

入力　”A KA I DO A WO A KE RO ”

（　あかいドアをあけろ　）

辞書の中に次の単語が含まれているものとしよう．

```
．．．．．．．
AKA     ．．．．．
AKAI    名詞  name,     形容詞  red
DOA     名詞  door
WO      格助詞
AKERO   動詞  open
．．．．．．．
```

また，次のような構文規則を用いる．

```
sent ——→ verb
sent ——→ case + sent
case ——→ noun + prep
noun ——→ adj + noun
```

対象世界には，次のものがあるとし，

..., door1,door2,door3, ...

対象世界に関する知識には次のような情報が含まれているものとする．

```
．．．．．．．
door1   赤くない   openできる
door2   赤い       openできない
door3   赤い       openできる
．．．．．．．
```

このような入力に対して，システムは次のように動作する．

［１］前処理

システムは入力文字列を用いて，文字節点と隣接関係節点を作る．

この例の場合，作られた文字節点は次に列挙する．

(ch_1 ch_2 ... ch_8 ch_9)

ここで，ch_1=”A”, ... ch_9=”RO”.

文字の間の隣接関係は次の隣接関係節点で表す．

(top cn_1 cn_2 ... cn_7 cn_8 end)

ここで，topとendで入力文の先頭文字と最後の文字を指し，cn_iでch_iとch_i+1が隣接したことを表す．

推論環境Eにはじめnilを設定する．

［２］最初の単語を見出す

単語を発見する操作によって，隣接関係節点topから，'AKA'と'AKAI'の二つの単語が発見され，単語の語境界の曖昧性が発生したとしよう．これに対して，システムは二つの仮定節点wd_1とwd_2を作る．

wd_1 ： AKA　　wd_2 ： AKAI

こうして，もとの環境Eから二つの子孫環境E1＝E∪{wd_1}とE2＝E∪{wd_2}が得られる．

今，wd_2の方が長いので，高い確からしさを与え，推論はまず環境E2で行う．E2をいまの推論環境としてEに記入する．

［３］節点の拡張

まず，wd_2から構文要素節点への拡張を行う．単語wd_2の概念の曖昧性のため，システムは二つの節点noun_1，adj_1を作る．辞書順によって，まず，noun_1を含む環境に移って推論を続ける．即ち，システムはこの時点にAKAIという単語は名詞であると信じる．

次いで，noun_1から意味節点へ拡張するとき，AKAIという名前の人は対象世界にいないので，いまの環境は矛盾することがわかる．この場合，システムは矛盾をTMSに通知する．TMSはまず，Eに記録した環境{wd_2,noun_1}に矛盾があったことをATMSに知らせ，矛盾する環境を併合する必要があるかどうかを調べる（今の場合併合する必要はない）．システムは次の推論環境として，環境{wd_2,adj_1}を選出する．

節点の拡張はこのようにして続けられる．いま，節点adj_1とnoun_2（＝DOA）から導いた節点 noun_3を意味表現節点へ拡張することを考えよう．

この節点の意味は”赤いドア”である．対象世界には，赤いドアは二つあるから，システムはこれに対して，二つの意味表現節点conc_1（＝red_door2）とconc_2（＝red_door3）を作り，推論はまず，conc_1を含む環境で行うとする．その後，動詞節点verb_1（＝AKERO）を拡張するとき，door2がopenできないことがわかる．前と同じように，いまの環境を矛盾する環境として捨て，推論はconc_2を含む環境に移る．この新しい環境では，前のverb_1に関する推論結果を直ちに利用できる．

［４］出力

最後にシステムは次のようなリストを出力する．

(command (act open) (object door3))

即ち，「入力文は一つの命令で，その命令の動作はopenであり，動作の対象はdoor3である．」というような意味解釈が得られる．

６　終わりに

本稿ではTMSとATMSの自然言語理解への応用について，それぞれの利点と欠点について考察した．両者の利点をさらに向上し，欠点を克服するために，TMSとATMSを併用して，確からしさに基づいた統合的自然言語理解システムを構成する方法を述べた．この方法では，確からしさによって，処理過程を確からしい方向に導くことができる．現在このシステムを実現中である．

このようなシステムを実現する際に，更に詳しく検討する必要がある．

今後の課題には次のようなものがある．

①一般性のある確からしさの表現法と利用法．

②入力に誤りがある場合への対処．

［参考文献］

[1] Doyle,J.,A Truth Maintenance System,
Artificial Intelligence 12 (1979)

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

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

[4] 藤崎博也，星合忠；言語表現の曖昧性，
知識工学　Ⅱ－１．３，
田中幸吉（編）　朝倉書店．

[5] Waltz,D.L. & Pollack,J.B.,
Massively Parallel Parsing:
A Strongly Interactive Model of
Natural Language Interpretation
Cognitive science 9,51-74(1985)

[6] 石井彰，統合的自然言語理解のための
重み付き依存関係ネットワーク，
京都大学特別研究報告書（1987）．

変数の集合。$\theta \upharpoonright V=\{t/v \in \theta \mid v \in V\}$)

$\theta 1+\theta 2+$mgu(t1',tn')=⊥のとき、このユニフィケーションは失敗終了する。($\theta 1+\theta 2+$mgu(t1',t2'))var(t1) に変数しか含まれていないとき、このユニフィケーションは成功終了する。それ以外の場合は、このユニフィケーションは待機し、$\theta 1+\theta 2$に対するmgu(t1',t2')の結合は遅延される。

変数のインスタンシェーションは、tが変数以外の項であるような代入{t/v}の結合である。このような代入を結合することによって、待機状態にある結合の条件が変化し、待機していた結合演算が行われる。

正負リテラルの消去手続きは、すべての項のペア(ti, si)について結合が求められたとき、この正負リテラルのペアは消去され、そのユニファイアは$\theta 1+\theta 2+\Sigma_i$mgu(ti',si')である。

3．7　ユニファイア木上の分散型ユニフィケーション

ユニファイア木に基づく論理プログラムの実行におけるシステムの実行環境はその時点で構成されている木構造の持つユニファイア因子である。プログラムの実行は、初期環境εへ代入を結合していくことである。この結合演算の同期制御は、モードづけされたユニフィケーションがユニファイア木において持つ実行意味に基づいて、正負リテラルの消去、節の消去、そして、ガード機構による節の選択という３つのレベルで構造的に行われる。

(1) 正負リテラルの消去操作

正負リテラルのペア(p(t1,...,tm)、n(s1,...,sm))に対して消去手続きが成功終了するとき、正リテラルが含まれる節の消去操作を起動する。節がコミットされたとき、消去操作は成功終了し、ユニファイアとして$\theta 1+\theta 2+\Sigma_i$(ti,si)を生成する。

(2) 節の消去

ガードあるいはボディの負リテラルと選択された正リテラルとの対に対して消去手続きを起動する。ゴールあるいはボディの全ての負リテラルが消去されたとき、節が消去される。正負リテラルのユニファイアが１個でもEになれば、全ての正負リテラルの消去を中止し、この節の消去によるユニファイアはEとする。ユニファイアはガードの負リテラルの消去によるユニファイアの結合がE以外のとき、この節はコミットされ、コミットにより生成されるユニファイアは、ここで結合されたユニファイアである。ボディの消去に関しても同様にユニファイアが生成される。ガードとボディとが消去されたとき、節が消去され、そのとき生成されるユニファイアは、それぞれによって生成されたユニファイアの結合である。

(3) ガード機構

ガード機構は、節の消去手続きを制御し、コミットにより外部の環境に結合すべきユニファイアを選択する。ガードあるいはボディにおける正負リテラルのすべての組合せに対して、局所環境を用意する。個々の環境において、正リテラルが選択された節に対して消去手続きを起動する。これらの環境のうち、節の消去手続きが最初にすべてコミットされた環境のみを選択し、他の選択を放棄する。

ここでの局所環境は、ネストすることができる。たとえば、

```
h(...):-p(X) | ....
p(Y):-q(Y) | ....
q(Z):-r(Z,a),s(Z) | ....
q(W):-r(W,b),s(W) | ....
```

のような場合には、ガードのなかでガードの選択の局所環境が用意される。このとき、コミットによって、１つ外側の環境にたいしてユニファイアを結合する。この様子をユニファイア木の上でとらえると図 3.5のようになる。

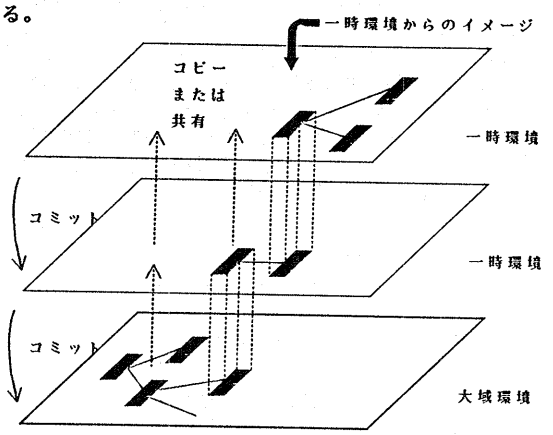

図３．５　ユニファイア木における一時環境のネスト

4．分散型ユニフィケーションに基づく実行

本節では、以上のような分散型ユニフィケーションに基づく並行論理型プログラムの実行について説明する。

4．1　節の実行制御

入力プログラムは、述語記号のマッチングによって、節の間でどのようにユニフィケーションが行われるかという関係を示している。この関係に基づいて、入力節の変数をリネームし、ユニフィケーションを行う制御機構が、節毎に分散して埋め込まれている。ユニファイア木に基づいて、節はリネーミング代入として定義され、この節には、ユニフィケーションを行うための制御が埋め込まれているとする。この制御は、外部からの制御信号を受信することにより起動される。また、この節自身も他の節に対して制御信号を発信することができる。（図 4.1）ユニフィケーションのための制御には、ユニファイされるべき項とユニフィケーションを行うための同期制御が書き込まれている。

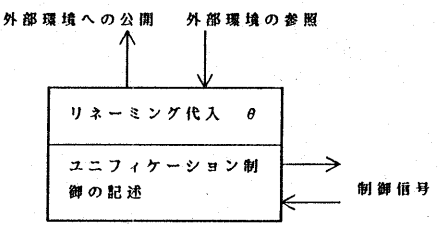

図４．１　制御をもつ代入としての節

節は、リネームする入力節により、タイプわけされる。ここでは、このタイプのことをクラスと呼び、あるタイプを表す入力節をリネームして得られた節を、そのクラスに属するインスタンスと呼ぶ。

4．2　モード付きユニフィケーションの処理

モード付きユニフィケーションの制御機構を実現するものして、チャネルを導入する。チャネルは、項のペア

のmgu を生成する具体的な機構である。チャネルはインスタンスによって生成され、２個のインスタンスに含まれる項を対応づける。チャネルには入力プログラムにおけるモードによって、ユニフィケーションの方向づけがなされている。チャネルは、インスタンスからの制御信号を受けることにより、項の対のmgu を生成して、これを一方のインスタンスに対して公開する。そのユニフィケーションが待機状態に遷移するときは、mgu の公開を遅延する。インスタンスは、チャネルから受け取ったmgu に自らのリネーミング代入を、結合して外部環境に公開する。ここで、チャネルは述語引数の数だけインスタンス間に生成される。（図 4.2）

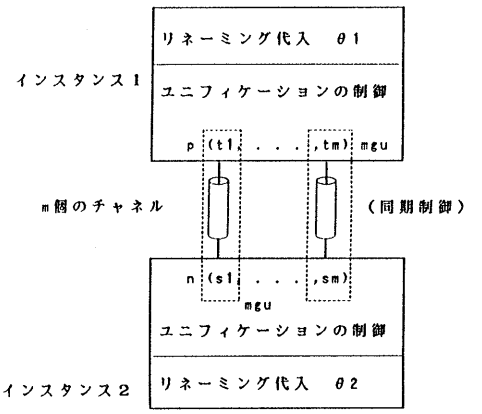

図４．２　インスタンス間に生成されるチャネル

４．３　実行の概要

初期環境として、ゴール節のクラスに属するインスタンスがシステムによって生成される。ゴール節に対応するインスタンスはその制御に基づき、子にもつインスタンスを生成し、そのインスタンスとの間にチャネルを生成する。このチャネルは、インスタンスの項の間でユニフィケーションを行い、そのユニファイアを生成する。インスタンスはこのうち、一組をガード機構によって選択し、外部環境へ公開していく。同様に、個々のインスタンスは、負リテラルの数だけのインスタンスの組合せを生成しながら、木構造を構成していく。（図 4.3）ガード機構による選択は、インスタンス間で、コミットに関する制御信号をやり取りすることで実現する。

４．４　制御機構の記述

ここで、節に埋め込まれた制御機構をoccam[6] 風の言語によって記述する。occamは、通信を行いながら並列に実行されるプロセスを簡潔に記述することができる。ここでは、節の動作を表す式（動作式）にコンストラクタの記述を導入して、動作式を構成していく。個々の動作式は、成功終了または失敗終了する。

導入するコンストラクタは、つぎの３種類である。

SEQ ：要素の動作式を順番に実行する。要素の動作式が失敗した時点で失敗終了する。要素の動作式がすべて成功終了したとき成功終了する。

ALT ：要素の動作式を並列に実行する。要素の動作式のうち、１個でも成功終了するとその時点で、成功終了する。すべての、要素が失敗終了したとき、失敗終了する。

PAR ：要素の動作式を並列に実行する。要素の動作式が失敗した時点で失敗終了する。要素の動作式がすべて成功終了したとき成功終了する。∎

コンストラクタは、ブロック構造であり、ラベルはそのコンストラクタ内で有効であるとする。

制御として、次のような基本動作式を与える。

channel <チャネルラベル> <チャネル数>
　<チャネル数>だけのチャネルを生成する。これらは、<チャネルラベル>によって参照される。

newPI　<インスタンスラベル> <クラスラベル>
　<クラスラベル>（入力節）のインスタンスを生成する。これは、<インスタンスラベル>に束縛って参照される。

ctl　<インスタンスラベル>
　<インスタンスラベル>で参照されるインスタンスの制御を起動する。

allocate　<インスタンスラベル> <アトムインデックス> <チャネルラベル>
　<インスタンスラベル>で参照されるインスタンスの<アトムインデックス>で参照されるアトムに含まれる項を<チャネルラベル>で参照されるチャネルの一方に割り当てる。（ここで、アトムインデックスとは、ヘッドを１とし、ボディに左から２、３、．．．とアトムに添え字付したものとする。）

msg　<チャネルラベル>
　<チャネルラベル>で参照されるチャネルでmgu を生成する。チャネルの方向制限に合致しないときは、待機する。

passiveCopy　<インスタンスラベル>
　自身のガードにあたる部分を、局所環境に複製する。これをインスタンスとして<インスタンスラベル>で参照する。

activeCopy　<インスタンスラベル>
　自身のボディにあたる部分を、局所環境に複製する。これをインスタンスとして<インスタンスラベル>で参照する。

signal
　親のインスタンスへコミットシグナルを送信する。

commit　<インスタンスラベル>
　子のインスタンスからのコミットシグナルを受信するまで待機する。

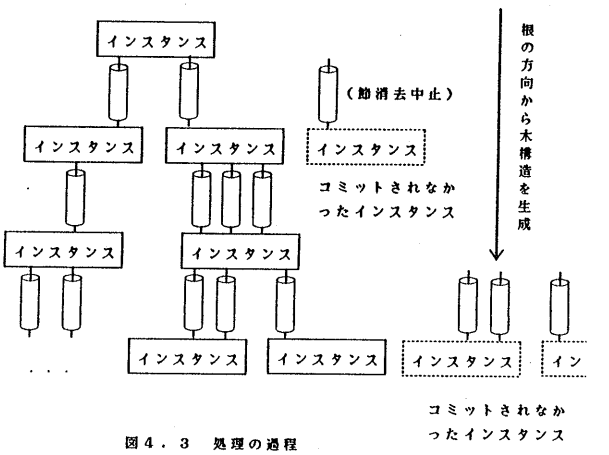

図４．３　処理の過程

これらの基本的な動作式をコンストラクタによって組み合わせることにより、次のように制御記述が構成できる。

1個の節の制御記述は、図 4.4のようにガード部とボディ部とから成り、ガード部の処理が終了したとき、コミットされたことを示す同期信号を発生する。ガード部とボディ部の構成は、その制御意味を直接にコンストラクタにより実現している。

(1) リテラルの消去操作

チャネルに項のペアを割り当ててユニフィケーションを行う。ユニフィケーションを行った後、新たなインスタンスの制御を起動する。これは、節の消去操作を起動することに対応する。

(2) 節の消去操作

リテラルの消去操作を負リテラルに対し、並列に起動する。(PAR コンストラクタ)

(3) ガード機構

基本的な選択は、ALT コンストラクタによる。選択に関する制御信号を用いてこの選択を行う。

それぞれの制御において、コピー、チャネル、新しいインスタンスなどの生成は、順を追って行う必要があるので、SEQ コンストラクタによって記述されている。

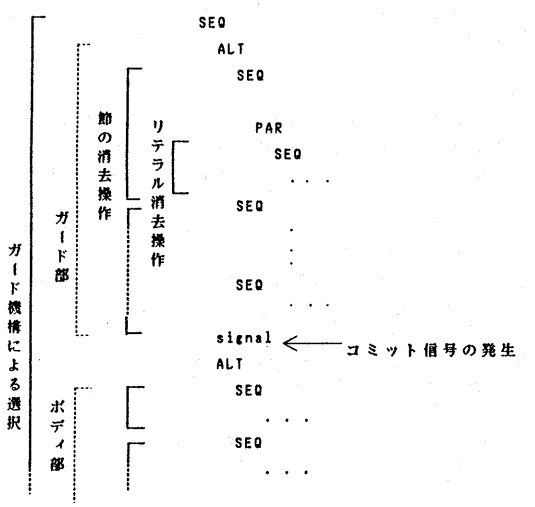

図４．４　制御機構の記述

＜例＞

```
mode add(?,?,!).
C1  add(zero,U,U).
C2  add(s(P),Q,s(R)):-add(P,Q,R).
C3  :-add(s(s(zero)),s(zero),Ans)
```

(ここで、C1～C3は入力節のラベルであり、クラスの参照に用いる。)

以上のような並行論理プログラムにおいて、各入力節の持つ制御は以下のように記述することができる。

```
Class C1:((?zero,?U,!U))
SEQ
  signal.

Class C2:((?s(P),?Q,!s(R))(?P,?Q,!R))
SEQ
  signal
  ALT
    SEQ
      activeCopy pi1
      channel channel1 3
      allocate pi1 1 channel1
      newPI pi2 C1
      PAR
        SEQ
          allocate pi2 channel1
          msg channel1
          ctl pi2
      commit pi2
    SEQ
      activeCopy pi3
      channel channel2 3
      allocate pi3 1 channel2
      newPI pi4 C2
      PAR
        SEQ
          allocate pi4 1 channel2
          msg channel2
          ctl pi4
      commit pi4

Class C3:((()(?s(s(zero)),?s(zero),!Ans))
SEQ
  signal
  ALT
    SEQ
      activeCopy pi5
      channel channel3 3
      allocate pi5 1 channel3
      newPI pi6 C1
      PAR
        SEQ
          allocate pi6 channel1
          msg channel3
          ctl pi6
      commit pi6
    SEQ
      activeCopy pi7
      channel channel4 3
      allocate pi7 1 channel4
      newPI pi8 C2
      PAR
        SEQ
          allocate pi8 1 channel4
          msg channel4
          ctl pi8
      commit pi8
```

ここでは、すべての節がガードを持たないので直ちにコミットされる。

４．５　組み込み述語

ここで、組み込み述語は、アサーションとして実現される。一般の論理型言語処理系における組み込み述語には２種類ある。１種類は、isなどのように常に成功して、変数の束縛あるいは副作用が目的であるものである。も

う1種類は、= などのように、項の値を評価して、それに対して真理値を返すことが目的であるものである。前者は、ユニファイアの無矛盾性に基づいた計算機構と相反しないが、後者はユニフィケーションでは成功してしまうが、プログラムとしては成功してほしくないという場合が生じる。

このため、ここでは真理値を返すような組み込み述語の制御において、偽を返すような場合はそのインスタンスが決してコミットされないとする。

4．6　処理系のインプリメンテーション

この制御記述を入力プログラムから生成し、制御記述を直接実行する処理系のプロトタイプをSmalltalk-80上に作成した。

5．　検討

分散型ユニフィケーションに基づく並行論理型言語の実行モデルについて検討すべき点を列挙する。

(1) 分散処理記述言語としての並行論理型言語

分散型ユニフィケーションに基づいて、並行論理型言語のプログラムは、1個の分散処理システムを記述している。ここにおける同期のための制御は、ユニフィケーションにおける変数の束縛条件である。分散型ユニフィケーションでは、ユニフィケーションは分散された処理単位である節間の通信とみなされるが、この通信で受け渡される値として、変数が許されることから、この変数を表現する共有領域が必要となる。このことから、並行論理型言語は強結合の分散システムを記述している。

(2) モードづけの方法論

並列分散処理における問題点としてデッドロックの検出と回避の問題がある。本稿で提案した処理系では、この問題はユニフィケーションのモードづけの問題としてとらえられる。また、モードづけが同期制御をそれ自身で直接表していないため、どのようにモードづけすれば、与えられた問題の同期制御を満たすかという方法論が必要である。

(3) 処理系の実現性

ここでの処理の分散は、論理的レベルで行われている。分散処理は実行時に決定され、ここでは具体的な資源やその割当に関する議論は行われていない。

GHCとConcurrent Prologの相違であるガードに含まれる変数の束縛に関する違いは、処理系で個々の環境に共通なユニファイア木の構造を共有できるか否かということに対応する。実際の処理系において、個々の環境は動的であるので、共通部分が共有できるか否かということは、システムのパフォーマンスに大きな影響を及ぼすと思われる。

(4) 弱結合された分散処理システムの記述

通信に基礎項のみを許すことで、弱結合した分散システムを記述できることが考えられる。このとき、分散型ユニフィケーションにおけるユニファイアには基礎代入のみが許される。これは、並行論理型言語の実行モデルに対して制限を与えることになる。

6．　応用

(1) 並行分散処理の仕様言語とその直接実行系

並行論理プログラムにおいて、分散処理自体は、分散型ユニフィケーションに基づいて、プログラムの計算の機構に埋め込まれる。このことは、この実行意味に基づいて並行論理型言語が、並行分散処理の言語として、抽象度が高いことを示しており、並行分散処理の意味をその具体的な計算機構から切り離して記述できることを示している。

(2) 並行論理型言語の動作の形式的記述

分散型ユニフィケーションでは、ユニフィケーションとその同期制御という形でプログラムの動作が決定される。ユニフィケーションを動作の基本におくことによって、プログラムの動作を形式的に記述することが可能である。これに関連して、並行論理プログラムの動作をCCSで記述する研究が行われている[8]。

7．　おわりに

本稿では、並行論理プログラムに対し、節毎に分散した制御のもとにユニフィケーションを行う分散型ユニフィケーションを提案した。この実行意味により、並行論理型言語の実行モデルにおける並列分散処理を直接実現することができる。ここでの並行処理の動作は、モード付されたユニフィケーションの同期制御という形で表現され、この同期制御は構造的に記述できることを示した。

<謝辞>

本研究会での発表にあたり、有用な御意見を下さった稲垣康善名古屋大学教授に感謝します。

最後に、本研究のプロトタイプ的な処理系の作成にあたり、御尽力頂いた富士ゼロックス社AI営業部大山定夫氏に深謝いたします。

[参考文献]

[1] C.Chang and R.C.Lee:'Symbolic Logic and Mechanical Theorm Proving' (Academic Press 1973)

[2] E.Y.Shapiro:'A Subset of Concurrent Prolog and Its Interpreter' (ICOT TR-003)

[3] K.Ueda:'Guarded Horn Clauses' Lecture Notes in Computer Science Vol.221 pp.168-179 (1985)

[4] 吉田幹:'論理型言語の意味論とそれに基づく処理系について' 第95回情報工学研究談話会 (京都大学工学部情報工学教室 1984)

[5] 村上健一郎、瀧和男、宮崎敏彦:'並列推論マシンにおける分散型ユニファイア構成法の検討' 信学技報CPSY86-19

[6] occamプログラミングマニュアル (啓学出版)

[7] 柴山潔:'記号処理マシン' 情報処理 Vol.28 No.1 pp.27-46 (1987)

[8] L.Beckman:'Towards a formal semantics for concurrent logic programming languages' Lecture Notes in Computer Science vol.225 pp.335-349 (1986)