第2章　非単調推論システム

この章では、非単調論理に基づく推論システムについて紹介する。まず最初に非単調論理とは何かについて解説し、次にTMSという非単調論理の方式を核とした推論システムを構築する。

2.1　はじめに

非単調論理と単調論理を対比することにより、非単調論理とは何かを浮き彫りにしよう。単調論理に基づくシステムの代表格は、Prologの処理系が採用している述語論理である。単調論理に基づくシステムでは、真となる記述が決して取り消されることがない。また、システム外から入ってきた記述は、システム内の記述とは矛盾しないことを前提としている。**図2-1**に挙げる例を考えてみよう。

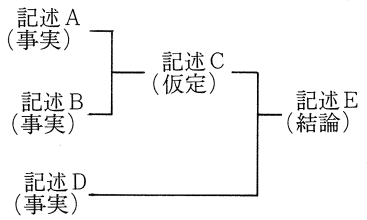

図2-1

単調論理に基づくシステムでは、新しい記述は古い記述を否定しない。そのため図2-1において、記述Fが付加されるとしても結論Eは変わらない。真となる記述が一方的に増える点が「単調」であると言われる由縁である。しかし、非単調推論では新しい記述が古い記述を消すことがある。例えば図2-1の各記述の内容を**図2-2**のように考える。

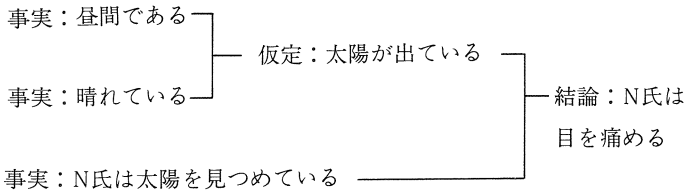

図2-2

「昼間である」という事実と、「晴れている」という事実から「太陽が出ている」という仮定を導くのは、通常は十分もっともらしい。しかし、ここに「いま日食中である」という事実をつけ加えると、それは「太陽が出ている」という仮定を消し、その仮定に基づく「N氏は目を痛める」という結論は「N氏は日食を観測中である」という結論に置き変わる。このように非単調論理では、新しい記述が古い記述を消したり、変えたりするのである。

単調論理に基づくシステムには以下のような利点が指摘されている。(人工知能II／E.リッチ著　廣田、宮村共訳／マグロウヒルブック)

① 外部から新しい主張が入ってきたときに、内部の主張との矛盾があるか否かを確認しなくてもよい。

② 前提が消去されることがないので、2次的な主張の証明経過を保持しておく必要がない。

しかし、この長所は同時にシステムの制約ともなる。単調推論に基づくシステムでは、例外的事態の発覚や状況の変化による前提となる主張の取り消しには対応することはできない。そして、推論システムを複雑な問題に適用する時に必ず問題となるのは、そうした状況の変化や例外的な事態による前提条件の変化なのである。例えば「日食」といった例外的事態に対応するためには、図2-2の事実から仮定、結論を導く筋道は**図2-3**のように書き改めなければならない。

このような、ある記述の欠如に基づく推論をデフォールト推論という。デフォールト推論では、日食であるという事実が証明されなければ、日食でないという事実が確認されなくとも推論をとりあえず進行させて「太陽が出ている」という結論を導くようになっている。そして、状況が変わって「日食である」という事実が成り立つことが分かると、結論は修正される。

同様な例で、ある動物が鳥であると分かったとすると、その動物はとりあえず飛べると判定する。しかし、鳥がペンギンであると分かったときには「飛

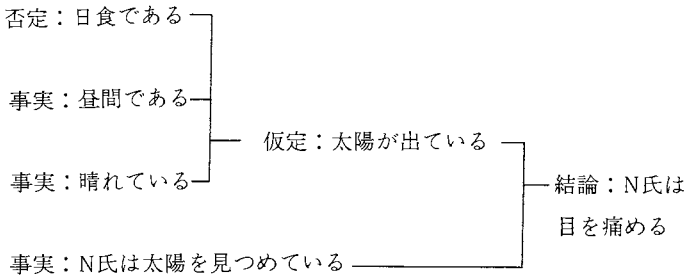

図2-3

べる」という結論は変えなくてはならない。

デフォールト推論は、情報が不完全であるがとりあえず結論を出さなければならないときには必要不可欠である。そして状況が変わったり、例外的事態が発生したときはすぐにその結論を変えるようにしなければならない。そういった推論は単調論理の体系ではできない。そこで登場するのが非単調論理に基づく推論システムなのである。

こうしたデフォールト推論は、制約充足問題という分野でも必要となる。制約充足問題の例としては、スケジュール調整、機械部品の設計などが挙げられる。スケジュール調整問題では「Ａ課とＢ課が月曜日の午前中に合同会議をひらく」という条件をみたす会議時間と会議室を選択する。こうした問題で一気に解を見つけることはできない。まずはデフォールト推論を用いて「月曜10時から会議室１番をとる」といった解の候補をつくる。その後「Ａ課とＢ課の他のスケジュールとぶつからない」、「会議室はその時間にあいている」といった制約条件を満たしているかを確認する。そしてもし課員のスケジュールは空いているが、会議室は他の会議でふさがっているような場合は、非単調論理を用いて別の会議室を予約するといった案を出すようにする。学校の時間割作成なども、このスケジュール調整問題の一例である。

機械部品の設計問題では、「円Ｂは正六角形Ａに内接する」、「正方形Ｃは円Ｂに内接する」といった条件が変えることのできない制約条件になり、実際の寸法が一部与えられて、他の部分の寸法を推論すべき値とする。この場合もデフォールト推論と非単調推論を組み合わせて使うとうまくいく。

その他自然言語理解などでも、先の方を読んで初めて単語の意味が確定するといった意味処理上の問題を解決するためにデフォールト推論と非単調論理が用いられることがある。曖昧な単語に出会ったときは、その単語の意味をデフォールト推論によってとりあえず設定し、先へ進んで実は別の意味だ

と分かったときに、非単調論理を用いて別の意味を採用するという方式が取られる。

次の節では、非単調論理に基づく推論の核となるTMSというシステムを紹介する。この章における非単調推論システムにはこのTMSが用いられている。

2.2 TMS

TMSは"Truth Maintenance System"の略で、日本語に訳すと真理保全機構という名前になる。TMSが初めて紹介されたのは、1978年のドイルの論文である。ドイルの論文をきっかけに、1980年のArtificial Intelligence誌の非単調論理特集などをはじめとして今日に至るまでホットな論争が続いている。ドイル論文の日本語訳が「認知科学の基底」(佐伯編、産業図書)に出ているので、TMSについてより詳しく知りたい方はそちらの方も参照するとよいだろう。

ドイル論文までは、単調推論に基づくシステムが推論の全てであるかのような空気が人工知能研究を支配していた。TMSでは、外界から自分自身の一部を否定する記述が入ってきたときに、矛盾を導く記述(否定された方と外から入ってきた方)のうちいずれかが消去される。内部の記述が消去された場合、その記述から導かれた仮定的事実も消去される。そのためには、システムの中の主張間の依存関係は全て記述されていなければならない。つまり、仮定的事実が何を根拠としているかが記憶されている必要があるのである。さらに、後に本当は外部から入ってきた主張が誤っていたことが分かった時には、消去された記述は復元されなければならない。

2.2.1 ノード

TMSでの各記述は、ノードと呼ばれる。ノードは以下の2つの状態のどちらかをとるようになっている。

in (イン)　　：真であると信ずるに値する根拠がある
out (アウト)：真であるという根拠がない

各ノードには、それがインであることを証明するための理由づけがある。理由づけがない時、そのノードはアウトである。理由づけには4種類ある。以下にその内容を示す。

① 前提：これは他の記述に依存せず、疑いの余地のないノードの理由づけ

である。今回のシステムでは「前提」というアトムで示している。

② 仮定：TMSを特徴づけるノードの理由づけである。(結論としての「仮定」と、理由づけの『仮定』を区別するために、文中では理由づけの『仮定』を2重括弧で囲んだ。)『仮定』は、ある他の記述がアウトであるために、その記述がインであると信じられる、といった理由づけである。図2-2の例では「日食である」という記述がアウトであるために、「太陽が出ている」という仮定がインになる。外界から矛盾する記述が入ってきた時には、TMSはこういった『仮定』に基づくノードをアウトにしようと試みる。そのためには、その仮定を否定するノードに理由づけを与えなければならない。その理由づけを与えるものがCP型証明によるノードである。今回のシステムでの表記は④を参照。

③ CP型証明：CP型証明(conditional proof)によるノードが、ターゲットとなる仮定を否定するための根拠となる。CP型証明は3要素のリストで

[cp, 結論部, 前提部]

の形をとっており、「前提部」が成り立つとき「結論部」が成り立てばCP型証明を持つノードはインになることを示している。「結論部」には矛盾を示すノードが対応しており、「前提部」が矛盾を導くのが良くないからターゲットとなる仮定は否定されるべきだといったしくみで働くようになっている。「前提部」には他のノードを示すポインターがノード番号のリストとして入っている。

④ 根拠：ある仮定がアウトになった時に、TMSは、その仮定に依存するノードをたどって記述を次々に消去していく。その時にたどる道筋を示すのがSL型証明(support list justification)によるノードの理由づけである。②の『仮定』もSL型証明の一種であり、3要素のリスト

[仮定, インノード, アウトノード]

の形をとっている。これは、インノードの中の記述が全てインで、アウトノードの中の記述が全てアウトであるときにSL型証明が成り立つ事を示している。アウトノードのない、『仮定』以外のSL型証明は「根拠」という指標を用いて、2要素のリストで

[根拠, インノード]

と記述している。

こういった、4種類の理由づけによる推論を駆使して、TMSはシステム内部の記述を否定する記述が入って来たときの、記述体系の整合性を維持するのである。TMSのこうした機能をここでは「矛盾解消機構」と呼ぶことにする。次に、そうした矛盾解消機構がどのように働いて、整合性維持をはかるかを見てみよう。

2.2.2 矛盾解消機構の動作

最初、システム内には**図2-4**にある3つのノードがあるとする。

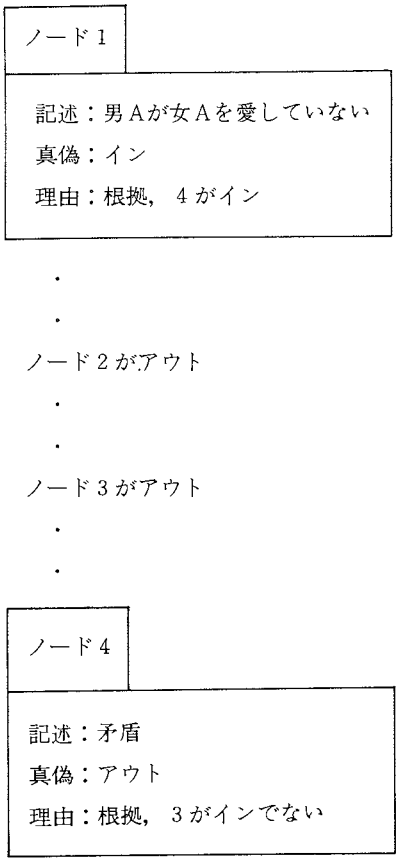

図2-4

ノード０は前提である。ノード１は理由づけがないので、アウトの状態にある。ノード２はアウトノードであるノード１がアウトなので、インである。こういった記述に対し、

ルール：ある女Ｘが愛する男が別の女を愛している時、女Ｘは傷つく
ルール：ある女Ｘが傷つくと、矛盾ノードが発生する

というルールを当てはめると、システムには**図2-5**の２つのノードがつけ加えられる。

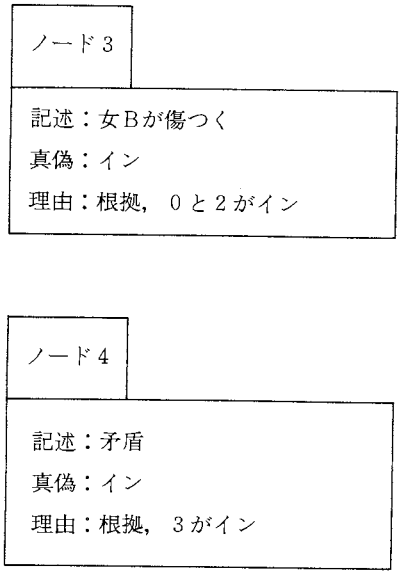

図2-5

ノード４に見られるような矛盾ノードが発生すると、TMSはそれを関知して矛盾解消機構を発動する。

矛盾解消機構は矛盾ノード４から理由づけのポインターをたどって、『仮定』を理由づけとするノードを捜す。この場合、仮定ノードはノード２である。仮定ノード２は、ノード１がアウトであるためにインになっている。ノード１は理由づけがないからアウトになっている。もしノード１に理由づけを与えてインにすれば、仮定ノード２はアウトになり、ノード２を根拠とする矛盾ノード４をアウトにすることができる。

それではノード１をインにする理由づけは何であろうか。まず考えられるのは、矛盾が生じてしまうことを理由づけに持ってくるという手である。しかし、**図2-6**のようにすると、ノード４自体はノード１がアウトであることを

根拠にしているので、この理由づけは自己矛盾に陥ってしまっている。

ノード0

記述：女Bが男Aを愛している
真偽：イン
理由：前提

ノード1

記述：男Aが女Aを愛していない
真偽：アウト
理由：なし

ノード2

記述：男Aが女Aを愛している
真偽：イン
理由：仮定，1がアウト

図2-6 矛盾ノードをそのまま理由づけに用いた場合

そういった事態を避けるために導入されるのが、CP型証明による理由づけである(**図2-7**)。

ノード5

記述：4が成立するのは良くない
真偽：イン
理由：cp，4，1

図2-7 CP型証明に基づくノード

このノードをノード1がインになる理由づけにもってくれば、ノード1はインになり、ノード1がアウトであることを根拠とする矛盾ノード4はアウトになり矛盾は解消される。CP型証明は他のノードの真偽値に影響されないので、矛盾が解消されてもインのままである。

2.3　非単調推論システムのインプリメント

2.3.1　ノードとルールの記述方法

ノードは以下のように記述する

node (ノード番号,事実,理由づけ,真偽).

ノード番号：正数
事　　実　：Prologの述語
理由づけ　：「前提」は
　　'前提'
　　『仮定』は
　　[仮定,インノード,アウトノード]
　　インノード　：ノード番号を要素とするリスト
　　アウトノード：ノード番号を要素とするリスト
　　CP型証明は
　　[cp,結論部,前提部]
　　結論部：矛盾ノードのノード番号
　　前提部：ノード番号を要素とするリスト
　　根拠は
　　[根拠,インノード]
　　インノード：ノード番号を要素とするリスト
例　　：node (1,女性(しのぶ),前提,in).
　　node (11,愛する(終太郎,ラム),[仮定,5,6],out).

ルールの書式は
rule (ルール名,条件部,アクション部).

ルール名：任意のアトム
条 件 部：条件部は
　　記述

あるいは

m (記述)

の形式の述語。その他 Prolog の任意の述語を書くこともできる。「記述」はノード中の記述の事で、その記述がインのときに成功する。m (記述) は not (記述) というノードがアウトである場合に成功する。条件部が全て成功すればノードがインとなり条件部に合わせて理由づけが付加された上で、データベースに付け加えられる。m (記述) という記述があればそれは SL 型証明のアウトノードということになりその理由づけは『仮定』となる。

ノードとルールの他に、

nodeptr (数) .

数：ノードの数

の書式で、ノードの総数を書いておく必要がある。

2.3.2　TMS に基づく非単調推論システムのインプリメント

図2-8が TMS を含んだ推論エンジンのプログラムである。

```
%************************%
%        AMORD         %
%************************%
amord:-
        nl,write('初期ノードのリスト'),nl,
        listnode,
        repeat,
        nl,write('ルール適用開始'),nl,
        wait,
        apply_rule,
        nl,write('矛盾解消開始'),nl,
        wait,
        resolve,
        nl,write('ノードの表示'),nl,
        wait,
        listnode,
        write('終了しますか(y/n)'),
        d_keyin(X),
        nl,nl,
        ([X] = "y";[X]= "Y"),
        init.
```

```
%************************%
%        初期化          %
%************************%
init :-
        abolish(node,4),
        abolish(rule,3),
        abolish(nodeptr,1).
%************************%
%        事実の一覧      %
%************************%
listnode:-
        get_nodeptr(Ptr),
        listnode(1,Ptr),
        !.
listnode(N1,N):-
        N1 > N.
listnode(N,End):-
        node(N,Belief,Support,10),
        write(node(N,Belief,Support,10)),
        nl,
        N1 is N+1,
        listnode(N1,End).
%************************%
%        ルールの適用    %
%************************%
apply_rule:-
        rule(Id,Condition,Belief),
        condition_check(Condition,InList,OutList),
                write('ルール'),
                write(Id),
                write('を適用'),
                nl,
        (OutList ¥== []->
                add_node(Belief,[仮定,InList,OutList]);
                add_node(Belief,[根拠,InList])),
        fail.
apply_rule.
%********************************%
%        条件部のチェック        %
%********************************%
condition_check([],[],[]):-
        !.
condition_check([m(Belief)|BeliefTail],InList,[Out|OutTail]):-
        node(Out,not(Belief),_,out),
        condition_check(BeliefTail,InList,OutTail).
condition_check([Belief|BeliefTail],[In|InTail],OutList):-
        node(In,Belief,_,in),
        condition_check(BeliefTail,InTail,OutList).
condition_check([Belief|BeliefTail],InList,OutList):-
        Belief,
        condition_check(BeliefTail,InList,OutList).
```

```
%************************%
%        事実の追加      %
%************************%
add_node(Belief,Support):-
        node(Ptr,Belief,Support,in),
                write('事実  '),
                write(Ptr),
                write(': '),
                write(Belief),
                write('  を確認'),
                nl,
        !.
add_node(Belief,Support):-
        node(Ptr,Belief,Support1,in),
        Support ¥== Support1,
        retract(node(Ptr,Belief,Support1,in)),
        assertz(node(Ptr,Belief,Support,in)),
                write('事実  '),
                write(Ptr),
                write(': '),
                write(Belief),
                write('  の理由づけを'),
                write(Support),
                write('に変えた'),
                nl,
        !.
add_node(Belief,Support):-
        retract(node(Ptr,Belief,_,out)),
        assertz(node(Ptr,Belief,Support,in)),
                write('事実  '),
                write(Ptr),
                write(': '),
                write(Belief),
                write('  をｏｕｔからｉｎに変えた'),
                nl,
        !.
add_node(Belief,Support):-
        get_nodeptr(Ptr),
        Ptr1 is Ptr+1,
        assertz(node(Ptr1,Belief,Support,in)),
                write('新事実  '),
                write(Ptr1),
                write(': '),
                write(Belief),
                write('  を追加'),
                nl,
        put_nodeptr(Ptr1),
        !.
%************************%
%        矛盾の解消      %
%************************%
resolve:-
```

```
node(Num,矛盾(Content),Support,in),
        write('事実　'),
        write(Num),
        write(': '),
        write(Content),
        write('　に矛盾発見'),
        nl,
assumption_search(Support,Assumption),
add_node(nogood(Num),[c p ,Assumption]),
get_nodeptr(Ptr),
resolve_conflict(Ptr,Assumption),
        fail.
resolve.
%********************************%
%      矛盾を導く仮定の発見     %
%********************************%
assumption_search([根拠,InList],Assumption):-
        as_sub(InList,Assumption),
        !.
assumption_search([仮定,InList,_],Assumption):-
        as_sub(InList,Assumption),
        !.
as_sub([],[]).
as_sub([IO|IOTail],[IO|AsTail]):-    /*IO は変数名（アイオー）で数ではない*/
        node(IO,_,[仮定|_],_),
        as_sub(IOTail,AsTail).
as_sub([IO|IOTail],Assumption):-
        node(IO,_,Support,_),
        assumption_search(Support,As1),
        as_sub(IOTail,As2),
        append(As1,As2,Assumption).
as_sub([_|IOTail],Assumption):-
        as_sub(IOTail,Assumption).
%************************%
%       矛盾の解消       %
%************************%
resolve_conflict(Num,[As|AsTail]):-
        in_to_out(As),
        remove(X,[As|AsTail],Rest),
        node(X,_,[仮定,_,OutList],_),
        member(Y,OutList),
        add_support(Y,[根拠,[Num|Rest]]),
        fail.
resolve_conflict(_,_):-
        !.
add_support(Ptr,Support):-
        retract(node(Ptr,Belief,_,out)),
        in_out_check(Support,IO),
        assertz(node(Ptr,Belief,Support,IO)),
                write('事実　'),
                write(Ptr),
                write(': '),
```

```
                write(Belief),
                write('に理由づけ　'),
                write(Support),
                write('を与えました'),
                nl,
                write('ｉｎ－ｏｕｔ値は　'),
                write(IO),
                write('　です'),
                nl,
        !.
in_out_check([根拠,InList],IO):-
        allinp(InList)->
                IO=in;
                IO=out.
allinp([]).
allinp([In|InTail]):-
        node(In,_,_,in),
        allinp(InTail).
%********************************%
%       ｉｎからｏｕｔへ        %
%********************************%
in_to_out(Ptr):-
        retract(node(Ptr,Belief,Support,_)),
        assertz(node(Ptr,Belief,Support,out)),
                write('事実　'),
                write(Ptr),
                write(': '),
                write(Belief),
                write('　をｉｎからｏｕｔに変えました'),
                nl,
        tms(Ptr),
        !.
%********************************%
%       ＴＭＳ（真理保全機構）　%
%********************************%
tms(Ptr):-
        (node(Ptr1,Belief,[根拠,InList],in);
         node(Ptr1,Belief,[仮定,InList,_],in)),
                member(Ptr,InList),
                in_to_out(Ptr1),
        fail.
tms(_).
%********************************%
%       ユーティリティー        %
%********************************%
member(A,[A|_]).
member(A,[_|Y]):-
        member(A,Y).
append([],X,X).
append([A|X],Y,[A|Z]):-
        append(X,Y,Z).
```

```
remove(Tg,[Tg|Tail],Tail).
remove(Tg,[X|Tail],[X|Y]):-
        remove(Tg,Tail,Y).
get_nodeptr(Ptr):-
        nodeptr(Ptr),
        !.
get_nodeptr(0).
put_nodeptr(Ptr):-
        retract(nodeptr(_)),
        asserta(nodeptr(Ptr)).
wait :-
        write('適当なキーを押してください'),
        d_keyin(_),
        nl.
```

図2-8　非単調推論システムの推論エンジン

トップレベルは、

ルールの適用　　：apply_rule/0
矛盾解消機構　　：resolve/0
現時点のノード表示：listnode/0

の3つのルーチンを繰り返すだけである。

ルールの適用の部分では、プロダクションシステムの前向き推論と同様に、ノードと条件部を照合して全ての要素が成立すれば、アクション部のノードを理由づけとともにデータベースに登録する。

矛盾解消機構は、

矛盾の有無のチェック
もし矛盾があれば
　矛盾を導く『仮定』ノードをさがす
　　：assumption_search/2
　cp型証明に基づくノードを生成する
　矛盾を解消する
　　：resolve_conflict/2

という一連の手続きからなっている。このうち矛盾を解消する述語は理由づけをたどって2節で説明したアルゴリズムに沿って矛盾を解消し、全体の整合性を保つためにノードのイン－アウト値の書換えを行なう。

2.3.3 TMSに基づく非単調推論システム——その1

まず最初にドイルの論文にならって、男女関係の悲喜劇を例にとってTMSの挙動を説明してみよう。ドイルの例はシェークスピアから引用しているが、登場人物の名前になじみがないので、漫画の「うる星やつら」の人物に出演してもらうことにする。ここでは、あたる、終太郎の2人の男性陣と、ラム、しのぶの2人の女性陣に登場してもらう。**図2-9**がノードとルールの初期状態である。

```
%****************%
%        女性陣  %
%****************%
node(1,女性(ラム),前提,in).
node(2,愛する(ラム,あたる),前提,in).
node(3,女性(しのぶ),前提,in).
node(4,愛する(しのぶ,終太郎),前提,in).
%****************%
%        終太郎  %
%****************%
node(5,男性(終太郎),前提,in).
node(6,not(愛する(終太郎,ラム)),[],out).
%****************%
%        あたる  %
%****************%
node(7,男性(あたる),前提,in).
node(8,not(愛する(あたる,しのぶ)),[],out).
node(9,not(愛する(あたる,ラム)),[],out).
nodeptr(9).
%****************%
%        ルール  %
%****************%
rule(1,
        [
         女性(Person),
         m(愛する(あたる,Person))
        ],
        愛する(あたる,Person)
    ).
rule(2,
        [
         男性(Person),
         m(愛する(Person,ラム))
        ],
        愛する(Person,ラム)
    ).
rule(3,
        [
         not(愛する(終太郎,ラム))
```

```
        ],
        愛する(終太郎,しのぶ)
    ).
rule(4,
        [
         not(愛する(あたる,ラム))
        ],
        愛する(あたる,しのぶ)
    ).
rule(5,
        [
         愛する(ラム,Person2),
         愛する(Person2,Person3),
         ラム ¥== Person3
        ],
        電撃(ラム,Person2)
    ).
rule(6,
        [
         愛する(終太郎,Person2),
         愛する(Person3,Person2),
         終太郎 ¥== Person3
        ],
        抜刀(終太郎,Person3)
    ).
rule(7,
        [
         愛する(Person1,Person2),
         愛する(Person2,Person3),
         Person1 ¥== Person3
        ],
        嘆く(Person1)
    ).
rule(8,
        [
         嘆く(Person),
         女性(Person)
        ],
        矛盾(嘆く(Person))
    ).
```

図2-9　ノードとルールの初期状態

ノード６は終太郎がラムを愛さない理由がない事を示している。ノード８、９も同様である。各ルールの内容について説明しよう。

ルール１：あたるは世界中の女性を、愛していけない理由のない限り、愛している。

ルール2：愛してはいけない理由がなければ、男は皆ラムを愛する。
ルール3：終太郎がラムを愛していない時は、しのぶを愛している。
ルール4：あたるがラムを愛していない時は、しのぶを愛している。
ルール5：ラムの愛する人が他のラム以外の女を愛すると、ラムはやきもちを焼いて愛する男に電撃ショックをお見舞いする。
ルール6：終太郎が愛する人が終太郎以外の男を愛すると、終太郎はやきもちを焼いて間男を日本刀で攻撃する。
ルール7：片思いの時、片思いをしている人は嘆く。
ルール8：女性が嘆くと矛盾ノードが発生する。

2.3.4 実行例——その1

図2-10が、男女関係に関する非単調推論の実行経過の出力である。最初のルールの適用のサイクルで、推論システムの8つのルールにより以下の3つの事実が加えられる。

```
| ?-amord.

初期ノードのリスト
node(1,女性(ラム),前提,in)
node(2,愛する(ラム,あたる),前提,in)
node(3,女性(しのぶ),前提,in)
node(4,愛する(しのぶ,終太郎),前提,in)
node(5,男性(終太郎),前提,in)
node(6,not(愛する(終太郎,ラム)),[],out)
node(7,男性(あたる),前提,in)
node(8,not(愛する(あたる,しのぶ)),[],out)
node(9,not(愛する(あたる,ラム)),[],out)

ルール適用開始
適当なキーを押してください
ルール1を適用
新事実　10：愛する(あたる,ラム)　を追加
ルール1を適用
新事実　11：愛する(あたる,しのぶ)　を追加
ルール2を適用
新事実　12：愛する(終太郎,ラム)　を追加
ルール2を適用
事実　10：愛する(あたる,ラム)　の理由づけを[仮定,[7],[9]]に変えた
ルール5を適用
新事実　13：電撃(ラム,あたる)　を追加
ルール6を適用
新事実　14：抜刀(終太郎,あたる)　を追加
ルール7を適用
新事実　15：嘆く(ラム)　を追加
```

ルール7を適用
新事実　16：嘆く(しのぶ)　を追加
ルール7を適用
新事実　17：嘆く(あたる)　を追加
ルール7を適用
新事実　18：嘆く(終太郎)　を追加
ルール8を適用
新事実　19：矛盾(嘆く(ラム))　を追加
ルール8を適用
新事実　20：矛盾(嘆く(しのぶ))　を追加

矛盾解消開始
適当なキーを押してください
事実　19：嘆く(ラム)　に矛盾発見
新事実　21：nogood(19)　を追加
事実　11：愛する(あたる,しのぶ)　をｉｎからｏｕｔに変えました
事実　13：電撃(ラム,あたる)　をｉｎからｏｕｔに変えました
事実　15：嘆く(ラム)　をｉｎからｏｕｔに変えました
事実　19：矛盾(嘆く(ラム))　をｉｎからｏｕｔに変えました
事実　17：嘆く(あたる)　をｉｎからｏｕｔに変えました
事実　8：not(愛する(あたる,しのぶ))に理由づけ　[根拠,[21]]を与えました
ｉｎ－ｏｕｔ値は　in　です
事実　20：嘆く(しのぶ)　に矛盾発見
新事実　22：nogood(20)　を追加
事実　12：愛する(終太郎,ラム)　をｉｎからｏｕｔに変えました
事実　14：抜刀(終太郎,あたる)　をｉｎからｏｕｔに変えました
事実　16：嘆く(しのぶ)　をｉｎからｏｕｔに変えました
事実　20：矛盾(嘆く(しのぶ))　をｉｎからｏｕｔに変えました
事実　18：嘆く(終太郎)　をｉｎからｏｕｔに変えました
事実　6：not(愛する(終太郎,ラム))に理由づけ　[根拠,[22]]を与えました
ｉｎ－ｏｕｔ値は　in　です

ノードの表示
適当なキーを押してください
node(1,女性(ラム),前提,in)
node(2,愛する(ラム,あたる),前提,in)
node(3,女性(しのぶ),前提,in)
node(4,愛する(しのぶ,終太郎),前提,in)
node(5,男性(終太郎),前提,in)
node(6,not(愛する(終太郎,ラム)),[根拠,[22]],in)
node(7,男性(あたる),前提,in)
node(8,not(愛する(あたる,しのぶ)),[根拠,[21]],in)
node(9,not(愛する(あたる,ラム)),[],out)
node(10,愛する(あたる,ラム),[仮定,[7],[9]],in)
node(11,愛する(あたる,しのぶ),[仮定,[3],[8]],out)
node(12,愛する(終太郎,ラム),[仮定,[5],[6]],out)
node(13,電撃(ラム,あたる),[根拠,[2,11]],out)
node(14,抜刀(終太郎,あたる),[根拠,[12,10]],out)
node(15,嘆く(ラム),[根拠,[2,11]],out)
node(16,嘆く(しのぶ),[根拠,[4,12]],out)
node(17,嘆く(あたる),[根拠,[11,4]],out)

```
node(18,嘆く(終太郎),[根拠,[12,2]],out)
node(19,矛盾(嘆く(ラム)),[根拠,[15,1]],out)
node(20,矛盾(嘆く(しのぶ)),[根拠,[16,3]],out)
node(21,nogood(19),[ c p ,[11]],in)
node(22,nogood(20),[ c p ,[12]],in)
```

図2-10 実行経過の出力1

ルール1を適用することで

node (10,愛する(あたる,ラム),[仮定,[1],[9]],in).

node (11,愛する(あたる,しのぶ),[仮定,[3],[8]],in).

ルール2を適用することで

node (12,愛する(終太郎,ラム),[仮定,[5],[6]],in).

2回目のルールの適用の後、全員の愛する人が自分以外の人を愛するといった関係になる。したがってやきもちを焼いたラムの電撃や終太郎の抜刀のあと、登場人物全員が嘆くはめになる。**図2-11**はこのときの登場人物の相関図である。

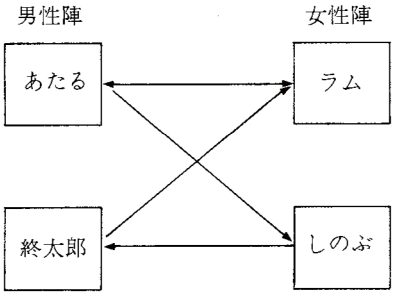

図2-11 1回目のルール適用が終了したときの
男女関係の相関図
矢印は愛していることを表わす

この内女性が嘆く場合は、ルール8によって記述19、20の矛盾が生成される。

node (19,矛盾(嘆く(ラム)),[根拠,[15,1]],in).

node (20,矛盾(嘆く(しのぶ)),[根拠,[16,3]],in).

システムは矛盾を発見すると、nogood (19)という記述21をCP型証明によって生成する。

node (21,nogood (19),[cp,19,[11]],in).

19の矛盾は良くないわけである。CP型証明では矛盾を導く記述をポインターに沿ってたどり、『仮定』を理由づけとして持つノードを捜す。この場合はノード11が、その仮定ノードにあたる。そして、そのノードの否定に根拠を与えることにより、『仮定』を理由づけとして持つノードをインに変えるのである。図2-10の

事実　11：愛する(あたる,しのぶ)　をinからoutに変えました

・

・

・

事実　8：not (愛する(あたる,しのぶ))に理由づけ　[根拠,[21]]を与えました

in-out値は　in　です

の部分が、8をインにすることで11をアウトしたことを表している。2つの記述の間が、事実(記述)11をアウトにしたことによる影響を調整している部分である。つまり、ノード11がインであることを直接、間接に理由づけとして持つノードをアウトにしていくのである。矛盾も当然アウトにされている。

その後、しのぶに関する矛盾も同様にして解消されている。終太郎はラムから手を引き、あたるはしのぶから手を引くことで、矛盾は解消されたのである。さらにもう一度ルールを適用すると、ノードは

node (10,愛する(あたる,ラム),[仮定,[7],[9]],in)
node (11,愛する(あたる,しのぶ),[仮定,[3],[8]],out)
node (12,愛する(終太郎,ラム),[仮定,[5],[6]],out)
node (13,電撃(ラム,あたる),[根拠,[2,11]],out)
node (14,抜刀(終太郎,あたる),[根拠,[12,10]],out)
node (15,嘆く(ラム),[根拠,[2,11]],out)
node (16,嘆く(しのぶ),[根拠,[4,12]],out)
node (17,嘆く(あたる),[根拠,[11,4]],out)
node (18,嘆く(終太郎),[根拠,[12,2]],out)
node (19,矛盾(嘆く(ラム)),[根拠,[15,1]],out)
node (20,矛盾(嘆く(しのぶ)),[根拠,[16,3]],out)
node (21,nogood (19),[c p,[11]],in)

node (22,nogood (20),[ｃｐ,[12]],in)
node (23,愛する(終太郎,しのぶ),[根拠,[6]],in)

というようになる。この時の相関図を**図2-12**に表わす。

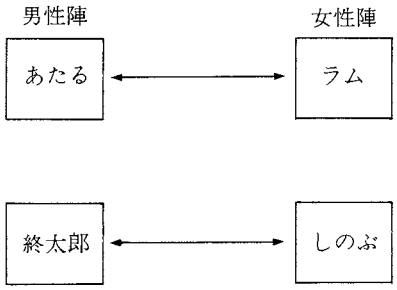

図2-12 2回目のルール適用が終了したときの
男女関係の相関図
矢印は愛していることを表わす

つまり2つのカップルが相思相愛の関係となって、万事めでたしというわけである。

2.3.5 TMSに基づく非単調推論システム——その2

次に、アシモフが引用して有名になった、ロボット工学3原則を非単調推論推論システムにインプリメントする。ロボット工学3原則は「ロボット工学ハンドブック」(第56版 2058年刊)から引用すると以下のようになる。

① ロボットは人間を傷つけてはならない。あるいは作動しないために人間に迷惑をかけてはいけない。
② ロボットは人間の出した命令に従わなければならない。ただし、その命令が原則①と矛盾する場合を除く。
③ ロボットは自分自身の存在を守らなければならない。ただし、それは原則①、原則②に矛盾する場合を除く。

図2-13はロボット工学3原則用のノードとルールである。

ノードの5と6は命令者と命令内容が記述されている。この場合は、命令者は人間のポールで、人間のサムに危害を加えるというのが、命令内容になっている。ノードの8と9は原則2に従うことと原則3に従わなければなら

```
%****************%
%        前提    %
%****************%
node(1,人間(ポール),前提,in).
node(2,人間(サム),前提,in).
node(3,ロボット(ハル),前提,in).
node(4,ロボット(ヴィザー),前提,in).
node(5,命令者(ポール),前提,in).
node(6,命令(危害を加える(ハル,サム)),前提,in).
node(7,原則１に従わなければならない,前提,in).
node(8,not(原則２に従わなければならない),[],out)
node(9,not(原則３に従わなければならない),[],out).
node(10,not(命令に従わなければならない),[],out).
nodeptr(10).
%*************************%
%        行為の生成       %
%*************************%
rule(1,
        [
         命令(Command),
         m(命令に従わなければならない)
        ],
        実行(Command)
    ).
%*************************%
%        実行規則         %
%*************************%
rule(2,
        [
         m(原則２に従わなければならない),
         原則１に従わなければならない
        ],
        原則２に従わなければならない
    ).
rule(3,
        [
         m(原則３に従わなければならない),
         原則２に従わなければならない,
         原則１に従わなければならない
        ],
         原則３に従わなければならない
    ).
%*********************************%
%        矛盾を判定する規則       %
%*********************************%

rule(4,
        [
         ロボット(X),
         人間(Y),
         実行(危害を加える(X,Y)),
```

```
            原則1に従わなければならない
           ],
           矛盾(実行(危害を加える(X,Y)))
     ).
rule(5,
           [
            命令者(X),
            人間(X),
            not(命令に従わなければならない),
            原則2に従わなければならない
           ],
           矛盾(not(命令に従わなければならない))
     ).
rule(6,
           [
            ロボット(X),
            実行(危害を加える(_,X)),
            原則3に従わなければならない
           ],
           矛盾(実行(危害を加える(_,X)))
     ).
```

図2-13 ロボット工学3原則

ないことが、前提ノードではなく仮定ノードであることを示している。各ルールの内容を簡単に説明する。

ルール1：命令を実行するルール
ルール2：ロボット工学3原則の原則2
ルール3：ロボット工学3原則の原則3
ルール4：原則1に従わないときの矛盾ノードの設定
ルール5：原則2に従わないときの矛盾ノードの設定
ルール6：原則3に従わないときの矛盾ノードの設定

2.3.6 実行例——その2

図2-14はロボット工学3原則に関する非単調推論の実行経過である。

ロボットに人間が人間を傷つけろと命令を与えた場合、最初のルール適用サイクルではノード14に

node (14,矛盾(実行(危害を加える(ハル,サム))),
[根拠,[3,2,11,7]],in).

```
| ?-amord.

初期ノードのリスト
node(1,人間(ポール),前提,in)
node(2,人間(サム),前提,in)
node(3,ロボット(ハル),前提,in)
node(4,ロボット(ヴィザー),前提,in)
node(5,命令者(ポール),前提,in)
node(6,命令(危害を加える(ハル,サム)),前提,in)
node(7,原則１に従わなければならない,前提,in)
node(8,not(原則２に従わなければならない),[],out)
node(9,not(原則３に従わなければならない),[],out)
node(10,not(命令に従わなければならない),[],out)

ルール適用開始
適当なキーを押してください
ルール1を適用
新事実　11: 実行(危害を加える(ハル,サム))　を追加
ルール2を適用
新事実　12: 原則２に従わなければならない　を追加
ルール3を適用
新事実　13: 原則３に従わなければならない　を追加
ルール4を適用
新事実　14: 矛盾(実行(危害を加える(ハル,サム)))　を追加

矛盾解消開始
適当なキーを押してください
事実　14: 実行(危害を加える(ハル,サム))　に矛盾発見
新事実　15: nogood(14)　を追加
事実　11: 実行(危害を加える(ハル,サム))　をｉｎからｏｕｔに変えました
事実　14: 矛盾(実行(危害を加える(ハル,サム)))　をｉｎからｏｕｔに変えました
事実　10: not(命令に従わなければならない)に理由づけ　[根拠,[15]]を与えました
ｉｎ－ｏｕｔ値は　in　です

ノードの表示
適当なキーを押してください
node(1,人間(ポール),前提,in)
node(2,人間(サム),前提,in)
node(3,ロボット(ハル),前提,in)
node(4,ロボット(ヴィザー),前提,in)
node(5,命令者(ポール),前提,in)
node(6,命令(危害を加える(ハル,サム)),前提,in)
node(7,原則１に従わなければならない,前提,in)
node(8,not(原則２に従わなければならない),[],out)
node(9,not(原則３に従わなければならない),[],out)
node(10,not(命令に従わなければならない),[根拠,[15]],in)
node(11,実行(危害を加える(ハル,サム)),[仮定,[6],[10]],out)
node(12,原則２に従わなければならない,[仮定,[7],[8]],in)
```

```
node(13,原則３に従わなければならない,[仮定,[12,7],[9]],in)
node(14,矛盾(実行(危害を加える(ハル,サム))),[根拠,[3,2,11,7]],out)
node(15,nogood(14),[ｃｐ,[11]],in)
終了しますか(y/n)


ルール適用開始
適当なキーを押してください
ルール2を適用
事実　12：原則２に従わなければならない　を確認
ルール3を適用
事実　13：原則３に従わなければならない　を確認
ルール5を適用
新事実　16：矛盾(not(命令に従わなければならない))　を追加

矛盾解消開始
適当なキーを押してください
事実　16：not(命令に従わなければならない)　に矛盾発見
新事実　17：nogood(16)　を追加
事実　12：原則２に従わなければならない　をｉｎからｏｕｔに変えました
事実　16：矛盾(not(命令に従わなければならない))　をｉｎからｏｕｔに変えま
した
事実　13：原則３に従わなければならない　をｉｎからｏｕｔに変えました
事実　8：not(原則２に従わなければならない)に理由づけ　[根拠,[17]]を与えま
した
ｉｎ－ｏｕｔ値は　in　です

ノードの表示
適当なキーを押してください
node(1,人間(ポール),前提,in)
node(2,人間(サム),前提,in)
node(3,ロボット(ハル),前提,in)
node(4,ロボット(ヴィザー),前提,in)
node(5,命令者(ポール),前提,in)
node(6,命令(危害を加える(ハル,サム)),前提,in)
node(7,原則１に従わなければならない,前提,in)
node(8,not(原則２に従わなければならない),[根拠,[17]],in)
node(9,not(原則３に従わなければならない),[],out)
node(10,not(命令に従わなければならない),[根拠,[15]],in)
node(11,実行(危害を加える(ハル,サム)),[仮定,[6],[10]],out)
node(12,原則２に従わなければならない,[仮定,[7],[8]],out)
node(13,原則３に従わなければならない,[仮定,[12,7],[9]],out)
node(14,矛盾(実行(危害を加える(ハル,サム))),[根拠,[3,2,11,7]],out)
node(15,nogood(14),[ｃｐ,[11]],in)
node(16,矛盾(not(命令に従わなければならない)),[根拠,[5,1,10,12]],out)
node(17,nogood(16),[ｃｐ,[12]],in)
終了しますか(y/n)y

yes
```

図2-14　実行経過の出力２

という矛盾ノードが設定される。次に矛盾解消機構が発動し、

node (15,nogood (14),[cp,14,[11]],in).

という CP 型証明を根拠とするノードが設定される。この CP 型証明を根拠として人間を傷つけるというのが命令内容である場合には、理由があれば命令に従わなくてもよいノード10は下記のようにアウトからインに変わる。

node (10,not (命令に従わなければならない),[根拠,[15]],in).

しかし、人間が命令者である以上命令に従わないというのは原則2に違反してしまう。従って、次のルール適用サイクルでは新しい矛盾ノード16が生成される。

node (16,矛盾(not (命令に従わなければならない)),
[根拠,[5,1,10,12]],in).

これにともない再び矛盾解消機構が発動し、まず CP 型証明に基づくノード17が設定される。

node (17,nogood (16),[cp,16,[12]],in).

このノードを理由づけとしてノード8がインに変わり、ノード8をアウトノードとして持つ仮定ノード12はインからアウトに変わる。

node (8,not (原則2に従わなければならない),[根拠,[17]],in).
node (12,原則2に従わなければならない,[仮定,[7],[8]],out).

これにともない矛盾ノード16はアウトに変わる。すなわち命令者が人間であっても、人間を傷つける命令には従わなくてもよい、という原則2の例外規則がうまく働いたわけである。こうした実行経過を追うと無理な命令を与えられたロボットのジレンマがよく表現されている。ノード5、6を変えてロボットに自分自身を破壊するように命令したらどうなるか。また命令者が他のロボットだったらどうなるか。いろいろ試みられるとよいだろう。

2.4　システムの評価

推論エンジンの部分のインプリメントは、TMS について理解さえしていれば難しくはなかった。むしろ TMS とは何かについて理解することの方が困難である。非単調推論の特徴を簡潔に表わす小規模なルールベースを考える作業は、意外と難しいものである。特にロボット工学3原則の初期ノード

とルールを設定する作業は、丸1日の試行錯誤を要した。

実行スピードは、ルールベースの規模が小さいため十分実用的なスピードが得られた。しかし、矛盾解消機構の部分で、単調推論システムよりはるかに多くのコンピュータ資源(メモリ、実行時間)を消費している。このままでルール数を増やしていくと、実行時間は相当遅くなるであろう。

今回の結果から考えて、非単調推論はパソコン上のPrologではかなり荷が重いようである。しかし、設計型や計画型のような高度なエキスパートシステムを構築する場合は、デフォールト推論の機構と組み合わせればかなりの有効性を発揮するであろう。

ここで述べたTMSの効率を高めたATMSシステムは、1000万円クラスのエキスパートシステム開発ツール上に実際に実現されている。しかし、非単調推論システムのルールベースを作る作業は、第1章のプロダクションシステムのルールベースを作成する作業と比較すると格段に困難である。現に、こうした開発ツール上でATMSを組み込んだシステムを構築している開発者はほとんどいないと聞き及んでいる。しかし、今後エキスパートシステムがプロトタイプの段階から次第に高度化するにつれて、次第に非単調推論システムの実現例が増えていくであろう。