ICOT Technical Memorandum: TM-0495

TM-0495

プロダクションシステム KORE/IEにおける非単調推論

新谷虎松

May, 1988

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

プロダクションシステムKORE/IEにおける非単調推論

An Approach to Non-monotonic Inference Mechanism in Production System KORE/IE

新谷虎松
Toramatsu Shintani

富士通㈱国際情報社会科学研究所
IIAS-SIS, FUJITSU LIMITED
iia213@flab.fujitsu.junet

論理型言語上の前向き推論型プロダクションシステムKORE/IEにおける非単調推論機構の実現方式について述べる。非単調推論を実現する枠組みとしてReiterのデフォルト推論を導入し、矛盾の解消機構としてTMS的機構を実現する。TMSで用いられる効率の悪いdependency-directed backtrackingを回避するために、関係情報(つまり、data dependencies)はネットワーク管理サブシステムKORE/EDENを用いて独立的に管理される。矛盾の解消は、推論を取り消す(つまり、認識-行動サイクルを戻す)ことにより達成する。これは、KORE/IEにおける推論ステッパーをTMSのフレームワークに沿って拡張することにより実現する。非単調推論の例として、選択肢評価支援機構KORE/CDSSのAHPに基づく一対比較の重みづけ管理を示す。非単調推論により、一対比較における重みづけの整合性が効果的に維持される。

1. はじめに

KORE/IEは、論理型言語上に構築された高速なプロダクションシステムであり、柔軟なルール指向的プログラミングを提供する[新谷87]。KORE/IEの推論は、OPS5[Forgy 81]等に代表される前向き推論型プロダクションシステムと同様に、ワーキングメモリー(WM)を意識した認識-行動サイクル(Recognize-Act cycle)を実行することにより実現される。一般に、この種の推論は単調に行われ、WMは、単に、ルールを干渉させるための大域的なデータベースとして用いられる。つまり、WMにおける事実の一貫性は特に問題とされない(むしろ、無視されている)。しかしながら、不確実な知識に基づく推論を効果的に実現するためには非単調推論[松本87]メカニズムが必要とされる。このような非単調な推論メカニズムの実現は、専門的知識(もしくは、経験的知識)の有効性の範囲を広げ、ルール指向的プログラミングを用いたアプリケーション・プログラムの構築を容易にする。本論文では、特に、KORE/IEにおける非単調推論メカニズムの実現方式について論じる。本アプローチでは、関係ネットワーク・サブシステムKORE/EDEN[新谷86a]を用いて知識間の依存関係を表現する。これにより、非単調推論を実現する際に必要となるdependency-directed backtracking相当の機能を効率的に実現する。

2. 非単調推論のための機構

不確実な知識には、①問題解決(例えば、診断など)において知識を取捨選択する際に必要とされるものと、②問題解決(例えば、スケージュリングなど)において常識的な判断情報として必要とされるものがある。①に関する知識には、普通、確信度と呼ばれる不確実性を表す数値が付加され、知識間の整合性は保証されないのが一般的である。一方、②に関する知識は、当面の問題解決を達成するために必要とされるものであり、知識間の一貫性は保持される必要がある。このためには、過去に行われた問題解決の一部分を修正をする必要がある。KORE/IEにおける非単調推論は②に相当する不確実な知識を扱うために設計され、デフォルト推論[Reiter 80]に基づく推論を実現する。デフォルト推論は、反証の欠落(もしくは、暗黙の了解)に基づいた推論であり、②のタイプの不確実な知識(デフォルト知識)を扱うための枠組を提供する。

KORE/IEでは、デフォルト推論を実現する手法として、TMS(Truth Maintenance System)[Doyle 79]で用いられた知識間の矛盾解消のためのフレームワークを導入する。TMSはKORE/IEにおける矛盾解消機構として機能する。TMSを効果的に組み込むためには、KORE/IEにおける(1)推論の制御機構を利用し、(2)デフォルト知識を扱うための手段を新たに構築する必要がある。

2.1. 推論の制御

前向き推論型プロダクションシステムであるKORE/IEにおいて、推論は認識-行動サイクルにより実現され、推論の取り消しは認識-行動サイクルを戻すことに帰着される。認識-行動サイクルは図1で示す①~③の処理過程により構成される。①の照合過程は全てのルールの条件部(LHS(Left Hand Side)と呼ぶ)と大域的なデータベースであるWM(Working Memory)の内容(WM要素と呼ぶ)を照らし合わせて、インスタンシエーション(Instantiation)を選び出す。KORE/IEでは、インスタンシエーションは実行可能なルール名、そのLHSを満たしたWM要素のタイムタグ、及び照合における変数束縛情報により構成される。インスタンシエーションの集合は競合集合と呼ばれ、②の競合解消時に採用されるべきインスタンシエーションが一つ選択される。選択されたインスタンシエーション情報に基づき③の動作過程で選択されたルールのRHS(Right Hand Side)が実行される。

認識-行動サイクルを戻すためには、図1で示したサイクルの実行に伴うWMの変化(WM要素のWMへ

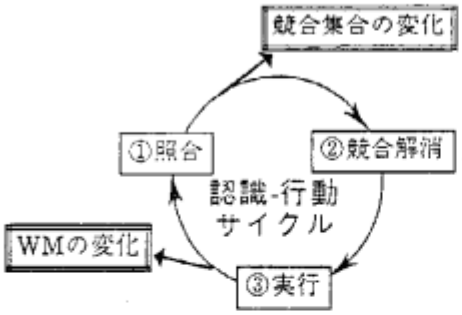

図1. 認識- 行動サイクル

の出入り)、及び競合集合の変化(インスタンシエーションの競合集合への出入り)を記録しておく必要がある。つまり、認識-行動サイクルを一つ戻すことは、一つ前のWM及び競合集合に戻ることに相当する。

KORE/IE では、認識-行動サイクルを制御するための推論ステッパーが用意されており、ステッパーとしてrun 及びbackコマンドがある。run コマンドは、推論ステップ数を指示することにより、認識-行動サイクルを前向きに進ませ、これにより推論が実行される。逆に、backコマンドは、推論ステップ数を指定し、このサイクルを戻す(カレントなWM及び競合集合を以前のWM及び競合集合に戻す)ことにより推論の取消しを行う。この推論の取消しに際しては、先に述べた各認識-行動サイクル毎に記録されたWMへのWM要素の出入り、及び、競合集合へのインスタンシエーションの出入りの情報が用いられる。このように、変化の分だけを記録するのは、記憶効率を上げるためである。なぜなら、推論ステップ毎に全てのWMの内容と競合集合を記憶しておくのは非能率的である。

例えば、backのための情報(以後、backデータと呼ぶ)として、サイクル毎に次のようなアサーションが保存される。

back_data(Step, Flag, Instantiation, WM_list)

ここで、Stepは認識-行動サイクルにおけるステップを表す整数であり、推論ステップに相当する。推論ステップ数Stepは認識-行動サイクルの1サイクル毎に1つ加算される。Instantiation は各ステップで実行されたインスタンシエーションを表す。WM_list はWMの出入りを表すリストであり、具体的には、追加されたWM要素及び、削除されたWM要素がそれぞれ、in_wm(<WM 要素>)、out_wm(<WM要素>)で表現される。Flagは、デフォルト知識の発見-変更を実現する際に用いられる重要な情報である(詳細は3.2 節で論じる)。

backコマンドは先のback_data の情報を0ステップ(変数Stepが0のもの)から集めて行くことにより、推論の取り消し(認識-行動サイクルを戻すこと)を実現する。0ステップは特別なbackデータであり、backデータを取り始める以前のWM及び、競合集合を保存する。図2はrun コマンドとbackコマンドの内部処理例を示している。図2において、"run(5)"は推論ステップを5段階進めるrun コマンドの使用例を示す。"back(2)" は推論ステップを2段階戻すbackコマンドの使用例を示す。図で示すように、推論ステップ2段階の取り消しを行なうためには、backコマンドはステップ0~ステップ3までを順にbackデータの情報を集めることによりその機能を実現する。つまり、backデータのWMの変化の記録、及び実行されたインスタンシエーションの情報を順にステップ0の情報へ適用することにより、目標とする推論ステップのWM及び競合集合を再現する。図において、"back データの始め" はbackデータを記録し始めた時点を示す。この時点は、KORE/IE において、"ie_debug_on" コマンドを実行することにより指定する。一方、"ie_debug_off"コマンドにより、backデータの記録をやめる。これらコマンドは、無意味なbackデータの記録を回避し、推論効率を上げるために用いられる。backデータは、ルールをデバッグしたり、非単調推論の際に必要とされ、通常の推論過程では不必要である。

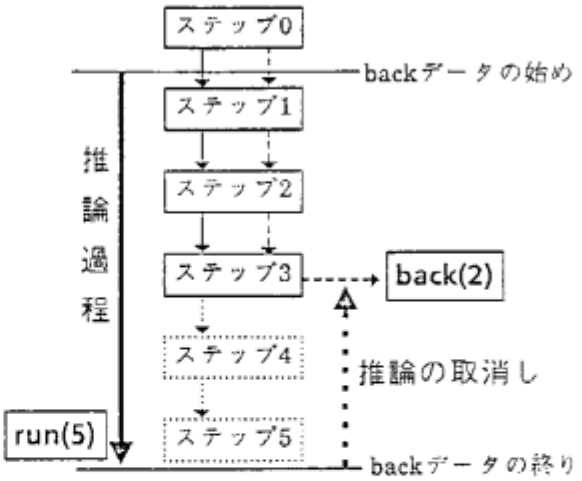

図2. backコマンドとrun コマンドの内部処理例

2.2. デフォルト知識と矛盾

デフォルト推論は不確実な知識に基づく推論である。本論文では、このような不確実な知識をデフォルト知識と呼ぶ。デフォルト知識は、証明不能性に基づいて用いられた知識(もしくは事実)であり、Reiterのデフォルトルールの中では、Mという様相記号を用いて表現される。例えば、次のデフォルト規則において、

$$\frac{\alpha(x) : M\ \beta(x)}{\gamma(x)}$$

前提条件におけるβ(x) は無矛盾を仮定して用いられるデフォルトの事実である。この規則は、α(x) が成立ち、β(x) の否定が証明されない限り、γ(x) を結論とする。その後、新しい事実の追加により、矛盾を生じた場合、原因となるデフォルト知識の修正、及びこの修正されたデフォルト知識を前提とする推

論を取り消す必要がある。つまり、矛盾はデフォルト知識に起因しているとする。ここでのデフォルト知識の修正は、デフォルト知識がその否定を証明されないことにより仮に採用されていたので、デフォルト知識の否定を主張することにより実現する。

KORE/IEでは、このような矛盾解消を行なうための機構としてDoyleのTMSの矛盾解消法のフレームワークに沿った矛盾解消機構を導入する。つまり、矛盾の発生時には、そこに到った推論連鎖の記録に基づいてバックトラッキング(dependency-directed backtracking)し、矛盾発生の原因となるデフォルト知識を見いだし、その取消しを行うものである(詳細は3.3節で述べる)。ここで、TMSと違う点は、TMSが矛盾解消のために矛盾に関連した全ての事実(もしくはノード)の信念(INとOUTの状態)を変更するのに対して、本アプローチでは、デフォルト知識を見つけその修正とそれに基づく推論過程を無効にすることである。TMSにおけるdata dependencies [McDermott 83]は各ノード記述のなかでjustificationとして分散化し記録されるのに対して、本アプローチの特長はdeta dependenciesが推論連鎖の記録としてノードの記述とは別に統一的に管理される点にある(詳細は3.1節で述べる)。これにより、TMSにおけるdependency-directed backtrackingを効率化する。以上のような枠組みにおいて、KORE/IEでデフォルト知識を扱うためには、①デフォルト知識のフラグ、及び②デフォルト知識の否定を表すフラグをWM要素のパターン記述のなかで表現する必要がある。

KORE/IEにおいてWM要素はタイムタグを持ったパターン記述により構成される。パターン記述は、次に示すように、1)クラス名と2)いくつかの"スロット−値"対であるスロット記述により構成される。

クラス名(スロット$_1$=値$_1$, スロット$_2$=値$_2$, ..., スロット$_n$=値$_n$)

このパターンの内部表現は次のようなProlog項で表現される。

KORE_クラス名(タイムタグ, 値$_1$, 値$_2$, ..., 値$_n$)

この表現において、functor名は、ユーザ定義のfunctor名との混乱を避けるために、"KORE_"の後にクラス名をつけたものを使う。タイムタグは、パターン生成時にユニークに決定される数であり、競合解消時に用いられる情報である。その他の引数はパターンのスロットの値を表す。これら引数の順番は、ユーザが"literalize"コマンドを用いてパターンにおけるスロットの位置を予め定義することにより決定される。literalizeコマンドはパターンの形状を定義するコマンドであり、次のように用いられる。

```
literalize(クラス名,
           [スロット1#タイプ1,
            スロット2#タイプ2,
                ...,
            スロットn#タイプn])
```

ここで、第1引数は先に示したパターンのクラス名、第2引数はスロットを定義するためのスロット定義リストである。このスロット定義リストにおいて、タイプnには、例えば、number、list、logical等があり、それぞれ、数値型、リスト型、論理型を意味する。このように予めパターンの形状を決定することは、認識-行動サイクルにおける照合過程を高速化するための情報を提供する[新谷87]。

そこで、先の①と②のフラグを記述するためのスロットとして、それぞれ"default"及び"proof"をシステムの組み込みのスロットとして用意しておくことにより、デフォルト知識の記述に対処する。組み込みのスロットは、literalizeコマンドで宣言しなくても自動的に付加されるスロットのことであり、他に、組み込みスロットの例として、先に示したタイムタグを記述するための"time_tag"がある。これら、組み込みスロットは、literalizeコマンドの実行時に、先に示したスロット定義リストの先頭にtime_tag、default、proofの順で挿入される。つまり、先のliteralizeコマンドの第2引数にあるスロット定義リストは次のように解釈される。

```
[time_tag#number,
      default#logical,
      proof#logical,
      スロット1#タイプ1,
      スロット2#タイプ2, ..., スロットn#タイプn]
```

default及びproofスロットの値は、logicalであり、"true"もしくは"fail"をとる。例えば、デフォルト知識はスロットdefaultの値がtrueであるパターン記述であり、デフォルト知識の否定を表すためにはスロットproofの値をnilとする。デフォルト知識は、KORE/IEの"make_assumption"コマンドを用いて生成される。make_assumptionコマンドは、makeコマンド(新たに生成するWM要素を引数とする)と同様に、次のように用いられる。

```
make_assumption(man(name = 太郎, weight = 80))
```

このコマンドにより生成されるWM要素のパターンは次の表現と等価である(ただし、literalizeコマンドによりユーザ定義のスロットが次の順次で定義されているとする)。

```
man(time_tag = 12, default = true,
    proof = true, name = 太郎, weight = 80)
```

ここで、整数12はシステムにより、自動的に決定されたタイムタグを示す。つまり、このmake_assumptionコマンドは、次のようなmakeコマンドの実行と同じである。

```
make(man(default = true,
         name = 太郎,weight = 80))
```

これら、コマンドの実行の結果、内部表現は次のようになる。

```
kore_man(12, true, true, 太郎, 80)
```

また、WM要素の否定は、パターンの前に"\"を付加して表すこともできる。例えば、次のmakeコ

マンドにより生成されたパターンのproofスロットの値はnilが設定される。

```
make( \man(name = 太郎, weight = 80))
```

3. KORE/IEにおけるTMS機構の実現

KORE/IEにおいて、TMS機構は命題間の整合性を保ちながら非単調推論を行うための管理機構として用いられ、デフォルト推論を実現するための基本的機能となる。TMSを具体的に実現するためには、①推論連鎖の記録,②①に基づくバックトラッキング,③矛盾の原因となったデフォルト知識の発見・変更等の手段を実現する必要がある。次に、KORE/IEにおける具体的な実現方式を述べる。

3.1. 推論連鎖の記録

推論連鎖の記録は、推論により新たに得られたデータに対して、その推論ルールのトリガーとして用いられたデータをjustificationとして付加することにより達成する。具体的には、ルールのRHSにおいて生成されるWM要素に対して、LHSにおけるLHSパターンにマッチしたWM要素を関係付けることである。KORE/IEでは、この関係付けはWM要素のタイムタグをキーとして行われ、これら関係情報はKORE/EDENを用いて統一的に管理される(図3参照)[新谷 86b]。KORE/EDENは、知識間の関係情報をビット情報として効率的に保存し、これら関係情報により構成されるネットワークを操作するための基本的な機能を提供する。

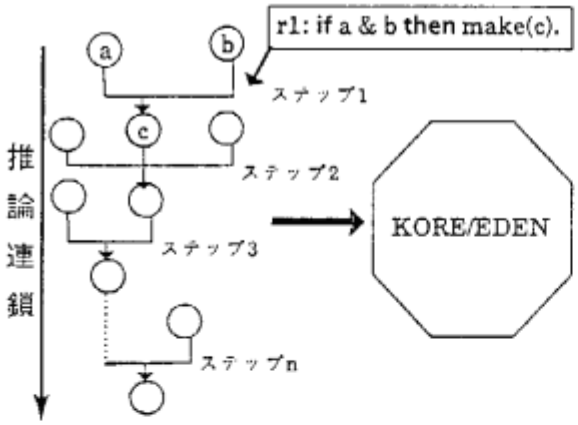

図3.KORE/EDENを用いた推論連鎖の記録

例えば、次のコマンドの実行により、

```
?- make_assumption(a(slot = 1)),
   make(b(slot = 1)).
```

次の、2つのWM要素が生成される。

```
2: b(slot = 1)
1: a(slot = 1)*
```

ここで、数字はタイムタグを示し、"*"はWM要素がデフォルトであることを示している(この表示形式は、KORE/IEにおけるWM表示コマンド"ppwm"により出力される)。このWM変化に伴い、順に次のルールが起動される(ここで、":"の左に示された"ex1"と"ex2"はルール名である。"if"と"then"の間にLHSが記述され、LHSはいくつかの2.2節で述べたパターン記述がデリミタ"&"をはさんで並べられる。"then"と"."の間にRHSが記述され、RHSは、WM操作コマンド(例えば、makeやmodifyは、それぞれ、新たなWM要素の生成、WM要素の更新に使われる)の他、Prologの述語がデリミタ"&"をはさんで並べられる)。

```
ex1: if a(slot = 1) & b(slot = 1)
        then modify(1,slot = 2).
ex2: if a(slot = 2)
        then make(b(slot = 2)).
```

この時、図4のような推論連鎖が作られる。図4にお

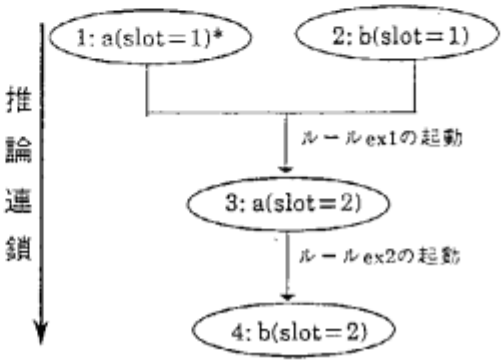

図4. 推論連鎖の例

ける推論連鎖は、次のようにKORE/EDENの"add_to"コマンドを用いて記録される。

```
add_to((3,kore_a,fact),
        [(1,kore_a,assumption),(2,kore_b,fact)],
        example)

add_to((4,kore_b,fact),
        [(3,kore_a,fact)],example)
```

add_toコマンドは、ノード間の隣接関係を宣言するコマンドであり、第1引数にノード、第2引数にこのノードの親ノードのリストを指定し、これら関係を第3引数で指定した名前で保存する。KORE/IEでは、第3引数にルールベース名が用いられる。ノードの表現形式としては、Prologの項を用いる。ここでは、次の表現形式を取る。

(タイムタグ,クラス名,デフォルト)

ここで、"デフォルト"は知識の区分を示すものであり、make_assumptionで生成されたWM要素に対してassumption、それ以外のWM要素に対してはfactが用いられる。

先のノード間の関係は次のようなassertionで内部表現される。

```
eden_table(example, (1, kore_a, assumption),
                                    1, [])
eden_table(example, (2, kore_b, fact), 2, [])
eden_table(example, (3, kore_a, fact), 3, [5])
eden_table(example, (4, kore_b, fact), 4, [26])
```

この項eden_tableの表記は次のように示せる。

eden_table(テーブル名,ノード,位置,行要素リスト)

これらassertionは、図5で示すような知識テーブル[新谷85]を表現しており、"テーブル名"は先のadd_toコマンドの第3引数で用いられた名前に相当する。"位置"はノードのテーブルにおける行(もしくは、列)位置を示し、"行要素リスト"はその行内容を表す。テーブルの行及び列の並びに使われるのはノードであり、図5ではタイムタグのみが示されている。説明の便宜のために、図5のテーブル表記は、一般的なテーブル表記とは異なり、列が右から順に並んでいる。

タグ4	タグ3	タグ2	タグ1	
				タグ1
				タグ2
		01	01	タグ3
	01	10	10	タグ4

図5. 知識テーブルの例

知識テーブルは、有向グラフを表現する隣接行列と到達可能性行列の情報を持つテーブルであり、その各要素は、(1)隣接関係を表現する2進数"01"、(2)到達可能性を表現する2進数"10"、(3)無関係を表現する2進数"00"、で表現される。図5では、便宜のために、(3)の情報を省略している。テーブルにおいて、ノードN_iとノードN_jとの関係は、テーブルのi行j列の要素で表現される。例えば、図5において、タイムタグが4のノード(4行目)に着目すると、第3列目に"01"があるので、隣接ノードとしてタイムタグが3のノードがあることがわかる。さらに、第1列と第2列に"10"があるので、到達可能なノードとして、タイムタグが1及び2のノードがあることがわかる。

先の項eden_tableの"行要素リスト"に現れる正整数は、行要素の2進数を左から右へ順にマージした2進数列を10進数として表現したものである。例えば、4行目を表す項eden_tableの行要素リストにおける正整数26は、知識テーブルにおける2進数列"011010"を10進表記したものである。行要素を正整数のリストで表したのは、長い2進数列に対処するためである。例えば、VAX11/780上のC-Prologでは、正整数は28ビットで構成されているので、ひとつの正整数で知識テーブルの14要素を表現できる。

知識テーブルに必要とされる基本的操作は、①ノード間の関係の記録、及び②関係の検索の2つである。①は、先に述べた知識テーブルの2進数列を表す正整数SのN番目のビットに1を立てることに相当する。N番目のビットに1を立てるために、Prologの整数ビット論理和(V)演算を用いる。例えば、Nビット目に1を立てるには、次のように、マスクとして2^{N-1}の正整数と正整数Sとを整数ビット論理和をとったものが求める正整数ANSとなる。

ANS is S V (2 ^ (N - 1)).

一方、②は、正整数SのN番目のビットに1が立っているかどうかを検査することにより実現できる。この検査には、Prologの整数ビット論理積(Λ)演算を用いる。例えば、正整数SのNビット目に1が立っているかどうかをチェックするには、次のように、マスクとして2^{N-1}の正整数と正整数Sとを整数ビット論理積をとった結果CHECKを求める。

CHECK is S Λ (2 ^ (N - 1)).

そして、この結果CHECKが1であれば、Nビット目に1が立っていることを示し、0であれば、1が立っていないことを示す。知識テーブルでは、基本的には以上の①と②の基本操作を2ビット操作へ拡張することにより(なぜなら、テーブル要素は2ビットを用いて表現されるから)、テーブル要素に対する操作を効率的に実現する。

関係情報を管理するための本アプローチの特長は、関係の検索、及び記録をPrologの算術演算に帰着させたことである。これにより、dependency-directedバックトラッキングに相当する機能を効率良く実現する。例えば、図4を例にすると、タイムタグが4のノードの到達可能なデフォルト知識のタイムタグのリストANS(この例では、ANS = [1])は、次のようにして見つけることができる。

```
?- supers((4,Class,Type),Result,example),
   bagof(DEFAULT,
         C^member((DEFAULT,C,
                   assumption),Result),ANS).
```

ここで、"supers"はKORE/EDENの基本的コマンドであり、第1引数で指定したノードの親方向の到達可能なノードを第2引数"Result"に出力するコマンドである。第3引数はテーブル名を示している。supersコマンドは、先に示したビット演算を正整数26(タイムタグが4のノードのテーブルの行要素)に適用することにより、その解をPrologの算術演算で求めている。この例では、supersの実行後に、Prologの組み込み述語であるbagofとmemberを用いてassumptionタイプのノードを求めており、効率が悪い。そこで、効率化のために、KORE/IEでは、supersの実行過程でassumptionタイプだけを見つけていく"assumption_supers"コマンドを用意している。つまり、コマンド内でチェックすべき項eden_tableを次のように制限している(assumptionだけを定数として指定すること)。

eden_table(TABLE,(TAG,CLASS,assumption),P,E)

KORE/EDENのその他のコマンドとして、例えば、子方向に到達可能なノードを見つける"subs"コマンド、親を見つける"parents"コマンド、子を見つける"children"コマンド等がある。

3.2. TMSバックトラッキングの制御

TMSのためのバックトラッキングは、KORE/IEにおける推論ステッパーを用いて実現する。2.1節で論じたbackデータにおける第2引数のFlagは、矛盾の原因となったデフォルト知識の発見や変更をする際に用いられる情報である。Flagのタイプとし

て、real、unreal、contradictionがある。realは通常の認識-行動サイクルにより獲得された情報、unrealは矛盾解消手続きが実行されているときの認識-行動サイクルにより獲得された情報、そしてcontradictionは矛盾解消手続きの際に後戻りの対象となった情報であることを示す。このようなFlagタイプは、所与の矛盾を解決するために、幾つかのデフォルト知識をチェックする際に必要である。つまり、矛盾解消手続き中の推論において、矛盾が新たに発生した場合に再帰的に矛盾解消手続きが呼び出されることを回避するためのものである。例えば、これらFlagタイプの用途を図示すると図6のように表せる。

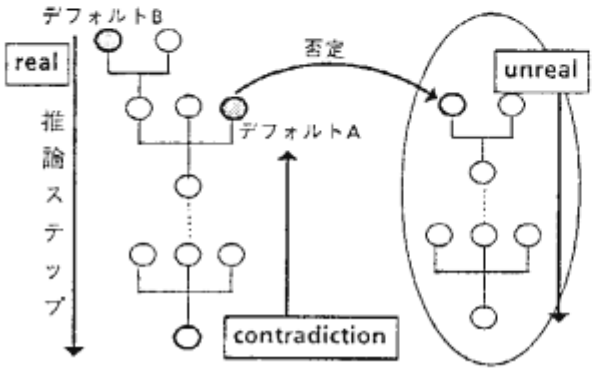

図6. backデータにおけるFlagタイプ

図6において、デフォルトAとデフォルトBはデフォルト知識を示している。図6は矛盾を解消するために(3.1節で述べた推論連鎖の記録からデフォルトAを見つけ、backコマンド機能でデフォルトAの推論ステップまで戻った後に)、デフォルトAの否定を主張することによるその後の推論の様子を示している。デフォルトAまでの推論ステップの後戻りの際には、途中のbackデータのFlagタイプをcontradictionにしている。このbackデータは、推論ステップの後戻りを無効にする際の情報となる。デフォルトAを否定したことによる新たな推論ではbackデータのFlagタイプとしてunrealが用いられる。これは、デフォルトAを仮に修正していることを示すものであり、もし、この推論でさらに矛盾が生じたなら、unrealタイプの推論を無効にし、その他のデフォルト知識であるデフォルトBをチェックする必要がある。デフォルト知識の否定の主張によりさらに矛盾が生じない場合には(あらたな推論の結果、矛盾がないこと)、Flagタイプがcontradictionのbackデータを消去し、さらに、unrealタイプの推論をrealタイプに変更する事により、矛盾の解決が実現されたことになる。

3.3. 矛盾解消手続きcontradiction

デフォルト推論において、矛盾は反証の欠落に基づいて用いられたデフォルト知識が原因となり発生する。そこで、KORE/IEでは、このような矛盾を解消するために、矛盾の原因となるデフォルト知識を見いだし、その否定を宣言する矛盾解消手続きcontradictionがある。手続きcontradictionは、TMSに相当する機能を提供し、ルールのRHSで用いられる。例えば、KORE/IEにおいて、次のように用いられる。

```
cry: if cry(name = X,about = Toy) then
        contradiction.
```

このルールは、LHSのcryに関するパターンとマッチするWM要素がWMに存在したら矛盾であることを示している。もしこのルールが適用されると、矛盾解消手続きcontradictionが呼び出され、このパターンとマッチしたWM要素を導き出す原因となったデフォルト知識を発見・チェックし、矛盾の解消をはかる。具体的には、矛盾の解消手続きは次のステップにより構成される。

ステップ1: 矛盾を導き出した推論連鎖において、矛盾の原因となるデフォルト知識のリストLを求める。そして、ステップ2へ行く。

このリストLは、KORE/EDENを用いて効率的に得られる(3.1節で述べたassumption_supersコマンドを用いる)。具体的には、推論連鎖の中からデフォルト知識のタイムタグを得る。リストLは、このタイムタグにより構成され、これらタイムタグは推論連鎖の深さ順(タイムタグが大きい順)にソートされている。これにより、矛盾に近い知識が先にチェックされる。これは、経験的なものであり、新しい知識の信頼性を先にチェックすることを意図するために採用しており、本質的ではない。

ステップ2: もし、リストLが空リストなら、矛盾解消手続きは失敗となり、手続きを終了する。さもなければ、ステップ3へ行く。

矛盾の解消の失敗は、本手続きで採用された矛盾の解消法(デフォルト知識の否定を宣言すること)の失敗を意味する。この時、システムは、矛盾解消を支援するための情報として、チェックしたデフォルト知識、及び矛盾に至った推論連鎖を提示する。ユーザは、この情報に基づき推論ルールを修正し、推論を再度やり直すことにより、矛盾の解消をはかる必要がある。

ステップ3: リストLの先頭のタイムタグL_{head}(つまり、$L=[L_{head}|L_{tail}]$)が示すデフォルト知識の否定を宣言する。そして、ステップ4へ行く。

具体的には、タイムタグL_{head}がある推論ステップまでbackコマンドを用いて推論サイクルを戻し(途中、backデータのFlagをcontradictionに変更する)、タイムタグL_{head}が示すデフォルト知識の否定を宣言する。例えば、否定の宣言は、2.2節で論じたように、デフォルト知識を表すパターンのproofスロットの値をnilに変更することにより実現する。これは、KORE/IEコマンドの"modify"コマンドを用いて実現する。

ステップ4: もし、ステップ3によりさらに新たな矛盾が発生したなら、ステップ3を無効にして、$L=L_{tail}$としてステップ2へ行く。そう

でなければ、タイムタグL_{head}が示すデフォルト知識に依存した知識を消去し(つまり、backデータのFlagの位置にcontradictionがあるものを消去する)、ステップ5へ行く。

ステップ3の宣言を無効にするには、①backデータのFlagの位置にunrealがあるbackデータを消去し、②Flagタイプが'contradictionであるbackデータに基づきステップ3の処理の前の状態に復活させる。

ステップ5: backデータのFlagがunrealであるものをrealに変更する。この時、矛盾解消手続きは成功し、手続きを終了する。

本手続きにおけるFlagの変更やbackデータの消去は、Prologの単純なバックトラッキング制御を用いてシーケンシャルに行われる。例えば、ステップ5での変更は次のよう実現できる。

```
?- retract(back_data(Step,unreal,Inst,WM)),
   assert(back_data(Step,real,Inst,WM)),fail.
```

同様に、ステップ4でのbackデータのFlagの位置にunrealがあるbackデータの消去はPrologのretract機能を用いて実現する。

つまり、本手続きにおいては、TMSにおけるdependency-directed backtrackingに相当する特別な制御機構はいらない。これにより、TMSに比べ、矛盾解消のためのバックトラキングによる負担を軽減する。

3.4. 本アプローチの特長

TMSは、矛盾の解消のために、矛盾の原因となる仮定ノードに依存した全てのノードのstatus(つまり、INもしくはOUT)をボトムアップ的に変更することにより矛盾解消をはかる。これに対して、本アプローチによるcontradiction手続きの特長は、トップダウン的に仮定ノードに相当するデフォルト知識の反証をテストして行くことにより矛盾解消をはかることである。つまり、TMSでは、データベース(もしくは、知識ベース)の信念の無矛盾性を保持するために、矛盾に関連した全てのノードのstatusを変更する。これは、信念をノードのなかに静的に分散化し記述していることに相当する。一方、KORE/IEでは、信念はルールを用いた推論として動的に管理され、WM要素(TMSのノードに相当する)に特別な信念を表すための記録は必要ない。つまり、矛盾解消のための信念の変更はこれらルールを用いた推論として実現している。

また、TMSでは、本システムにおけるデフォルト知識に相当する仮定ノードは、justificationにおけるINリストが空であるので、常に、推論連鎖木において親ノードのないノードに相当する。一方、本アプローチでは、必ずしもデフォルト知識が連鎖木において、このようなノードである必要はない。例えば、KORE/IEでは次のような、ルールが記述可能である。

```
ex3: if a(slot = 3) & b(slot = 1)
     then make_assumption(c(slot = 1)).
```

このルールは、次のような、WM要素(デフォルト知識ではない)により適用可能になる。

```
4: a(slot = 3)
3: b(slot = 1)
```

適用された結果、WMは次のようになる。

```
5: c(slot = 1)*
4: a(slot = 3)
3: b(slot = 1)
```

ここで、新たに生成されたタイムタグ5のWM要素はタイムタグ4と3のWM要素に依存したデフォルト知識となる。このようなデフォルト知識が矛盾の原因となった場合、矛盾の解消のためには、先に示したcontradiction手続きだけでは不十分である。さらに、ルール自身の修正も必要とされる。本アプローチでは、ルールのRHSを更新することでこれに対処している。例えば、先の例においてタイムタグ5の否定を主張することにより矛盾の解決がはかられたなら、先のルールは次のように更新される。

```
ex3: if a(slot = 3) & b(slot = 1)
     then make_assumption( \c(slot = 1)).
```

つまり、RHSの中で、タイムタグ5を生成するために使われたmake_assumtionコマンドの引数はその否定パターンに変更される。

KORE/EDENにおいて関係情報を記録するのにビット列(つまり、知識テーブル)を用いた利点は、リストで関係を直接に持つことに比べ、(1)サイクリックな関係(ループ構造)のチェックが容易である,(2)関係の検索が高速化できる点である。これは、記憶効率を犠牲にして検索効率と機能の向上を効果的に実現することを意図したものである[Shintani 86c]。行列がsparseの場合はリストで関係を直接持つ方が記憶効率が良い。KORE/EDENでは行列がsparseになる場合の記憶効率向上のためにテーブルの行要素のみを正整数のリストとして圧縮して保存している。さらに、記憶効率向上のためのいくつかの知識テーブルの圧縮機能を実現している[新谷 86a,86c]。一方、KORE/IEでは、関係情報は推論連鎖を記録するために用いられ、関係情報の量は推論のステップ数に依存する。関係情報(つまり、推論ステップ)の記録は2.1節で述べたように、必要な時(例えば、非単調推論の実行時)にのみ行なわれる。これは、単調な推論と非単調推論を組み合わせて用いることを前提にしている。これにより、大きな知識ベースを用いた推論における関係情報の記録の増大を回避する。例えば、第4節のKORE/CDSSの例では、重みづけの処理の部分でこのような非単調推論を用いており、その他の処理過程は通常の単調な推論が行なわれる。

知識テーブルへの関係の記録は(a)隣接関係"01"のセット,(b)到達可能関係"10"の伝播により達成される。その際、到達可能性"10"を伝播させる処理(Prologの論理ビット演算ORを用いる)でテーブル要素に"11"が生じたならその時点でループ構造が生じたことになる[新谷 86a]。ループチェックもPrologの論理ビット演算で容易に実現できる。例え

ば、ループがあるかどうかをチェックのための述語check_loopはC-Prologを用いて次のように定義できる。

```
check_loop(Number) :-
  Test is (Number ∧ 178956970) ∧
          ((Number << 1) ∧ Number),
  Test > 0.
```

ここで、Numberは行要素を表す正整数であり、"178956970"は次の2進数列

"10101010101010101010101010"

を表している。この2進数列の長さは、Prologシステムにおける正整数を表すための内部表現に依存する。述語check_loopはテーブル要素(つまり、3.1節で述べた2進数列)に"11"があるかどうかをチェックする。述語check_loopが成功した場合にはループがあることになる。

しかしながら、KORE/IEでの関係付はWM要素のタイムタグをキーとして行なわれるのでサイクリックな連鎖は生じない。つまり、同じパターンでもタイムタグが違うものは違うノードとして区別され記録されるからである。本アプローチは非単調推論機能を実現するために、logical な deductionを対象としたDoyle のTMS とは異なり、むしろ前向き推論型プロダクションシステムにおける推論形式にうまく適合したTMS 的な機構の実現を目指したものである。この違いは、先に述べたように、DoyleのTMSではノードのstatusの正当性に着目した処理を前提とするのに対して、本アプローチでは推論ステップの正当性に着目する点にある。

4. 応用例 -選択肢評価支援機構KORE/CDSS-

一対比較法は、意思決定者の主観的判断を取り入れることができる利点を有している。一方、その重みづけの整合性を成立させるためには、要素間で煩雑な一対比較のやり直しを強いられるのが一般的である。選択肢評価支援機構KORE/CDSS(Choice Design Support System) では、重みづけの整合性を保存するために、KORE/IE の非単調推論機能を用いている。これにより、矛盾を含む一対比較は調整され、重みづけの整合性を効果的に保持する。KORE/CDSS は意思決定者の選択問題における意思決定を支援し、主目標を達成するための副目標の設定過程を支援するための機能を提供する。目標は、目標を達成するために必要とされる項目(例えば、副目標や評価属性)を階層的に細分化することにより分析される。これにより、意思決定者の選択問題における意思決定を支援する。具体的には、選択問題の評価属性を抽出し、これら属性間の重みづけをAHP(Analytic Hierarchy Process)[Saaty 1980] に基づく一対比較法により実現する。

4.1. AHPに基づく重みづけ

AHP は、システムズ・アプローチと主観的判断を組み合わせることにより、定量分析では扱いきれない決定問題に対処する手法である。この手法では、意思決定者の勘や経験を生かすことがその主眼となっている。AHP における重みづけは一対比較がベースとなっており、求めた重みを行列として扱うことにより効率的に一対比較を行なうことができる。この行列の特徴は、①対角要素は1、②行列の要素の値はaij＝1/ajiである(aijは項目iの重みと項目jの重みを一対比較して得られた相対的重みづけ)。例えば、②は次のようにKORE/IE のルールを用いて記述できる。

```
comparison2:
  if   matrix_element(name = X,column = C,row = R,
                      weight = (W > 0)) &
     - matrix_element(name = X,column = R,row = C)
  then
       make(matrix_element(name = X,column = R,
                    row = C,weight = compute(1/W)).
```

ここで、matrix_elementはcolumn行row 列の行列要素を示す。重みはスロットweightで表現される。LHS の2つめの"－"を伴ったパターンは否定パターンと呼び、WMに存在しない場合に真となるパターンである。否定パターンの動作はPrologのnot (つまり、negation as failure [Clark 78])とは違い、WMにおける絶対的な"あるなし"を規定するものである。例えば、あるルールの否定パターンとして規定されたパターンがWMから消去された場合も、そのルールのトリガーとなる。このルールにより、非単調推論の結果、もし行列の要素の値aijが否定された場合、値ajiも取り消される。

一対比較による重みの整合性のチェックはこの得られた行列の最大固有値λ_{max} と対象の個数N を用いて、次の式(1)

$$(\lambda_{max} - N)/(N-1) \quad \cdots\cdots(1)$$

を用いることにより判定できることが知られている[Saaty 1980]。式(1) が0.1 以下なら整合性があると判断できる。

一対比較による要素間の重みづけは9点法が用いられ、その重みづけ(評点)は別の第三の要素から見て決定される相対的重みづけである。AHP では問題が階層的な構造に分解されていることが前提とされ、各レベルには幾つかの要素が並べられ、一対比較で比較基準とする第三の要素は一つ上のレベルの要素が用いられる。各レベルでの要素間の優先度(最終目標に関する重み)は先に述べた行列の固有ベクトルから得られ、レベル毎の優先度を合成することにより階層構造全体から見ての個々の要素の絶対的な重みを計算できる。そこで、AHP を効果的に用いるためには、①問題の階層構造を抽出するための階層化機構、及び②重みづけの整合性チェックのための重みづけ管理機構を効率的に構築する必要がある。本論文では、特に、KORE/IE の非単調推論の例として②の重みづけの管理について述べる。

4.2. 非単調推論を用いた重みづけの管理

重みづけは、一対比較法による判断の結果に矛盾がない(つまり、重みづけに関して推移律が成り立つ)場合に整合性があるという。重みづけの整合性

は4.1節で述べたように式(1)を用いることによりチェック可能である。しかしながら、幾つかの一対比較のうちどの比較が不適切であったかは知ることはできない。重みづけの修正のためには、不適切な一対比較を見いだし新たに重みづけをやり直す必要がある。最悪な場合、全ての重みづけのやり直しが必要になる。本機構では、重みづけを行なう際に(a)確実な重みづけ、(b)仮の重みづけという区別を表明することにより不適切な重みづけの修正を支援する。重みづけの修正は、先ず(b)のタイプの重みづけの修正を順次行なうことにより達成する。この修正過程はKORE/IEの非単調推論機能を用いて実現される。整合性のチェックは次のルールの呼び出しで行なわれる。

```
consistency_check:
  if matrix(max_eigen > 0.1) then contradiction.
```

このルールは、矛盾解消手続きcontradictionを用いてルールのif部の条件を生じさせた原因を見いだしその修正を試みる。これら原因は、先に挙げたタイプ(b)の重みづけに帰着され、システムはこれら重みの変更を行なうことにより整合性を保存する。手続きcontradictionは、4.1節の式(1)の値が0.1より小さくなった時点でその起動を終了する。チェックされるべき重みづけは次のルールを適用することにより変更される。

```
change_weight:
    if \matrix_element(name = X,column = C,
                                    row = R,weight = W)
    then
      remove(1) & new_weight(W,W2) &
      make(matrix_element(name = X,
                  column = C,row = R,weight = W2)).
```

ルールの条件部は事実matrix_elementの否定("\")を表現しており、先のcontradiction手続きにより生成される。この事実はチェックされるべき重みづけに相当する。結論部における手続きnew_weightは重みのスケールを変更するために用いられる。スケールの変更は、新たに得られる行列の最大固有値が小さくなる(つまり、先の式(1)の値が小さくなる)ように設定される。スケールの変更は±2の小さな範囲で行なわれる。これは、一対比較により得られる結果がそれほど矛盾しないことを前提としている。この前提は、問題を階層化した際、各レベルの要素の数を制限したことにより得られる。

5. おわりに

KORE/IEでは、非単調推論を実現する枠組としてデフォルト推論を導入し、これを実現する技法としてTMSで用いられている矛盾解消手法のフレームワークを用いた。TMSでは、データベースの無矛盾性を保持するために、矛盾に関連した全てのノードに記録された信念(つまり、status情報)を次々に変更することにより矛盾の解消をはかる。本アプローチでは、信念はルールを用いた推論として動的に管理され、矛盾の解決もKORE/IEの推論ステッパーを用いた推論の取り消し等により実現する。本アプローチの特長は、次の点にある。①システムアーキテクチャーを大幅に変更することなしに、前向き推論型プロダクションシステムKORE/IEに非単調推論を実現した。この非単調推論実現手法は、プロダクションシステム一般に容易に適用可能である。②TMSで用いられる効率の悪いdependency-directedバックトラッキングを回避するために、関係情報(つまり、data dependencies)の管理を独立させた。この関係情報の管理には、ネットワーク管理サブシステムKORE/EDENを用いて効率的に行なわれた。③KORE/IEの推論ステッパーを拡張する事により、効率が悪いとされるメタプログラミングを用いることなしに、推論の無矛盾性チェックを行なった。この無矛盾性チェックのための推論過程は、KORE/IEの普通の推論過程に帰着され効率よく実行される。4節では、KORE/IEにおける非単調推論機能を用いた応用例を示した。この例では、一対比較における重みづけの整合性を保持するために非単調推論機能を用いた。これにより、矛盾を含む一対比較は調整され、重みづけの整合性は効果的に保持された。

謝辞

日頃よりご指導頂く当研究所北川敏男会長ならびに榎本肇所長に感謝いたします。本研究を進めるに当たり、貴重な御意見を頂いた当研究所戸田光彦主任研究員及び片山佳則、平石邦彦研究員に深謝いたします。

尚、本研究は第5世代コンピュータプロジェクトの一環として行われたものである。

参考文献

[Clark 78]K.Clark: Negation as Failure, in "Logic and Data Bases" Plenum Press, pp.292-308(1978)

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

[Forgy 81]C.L.Forgy, OPS5 User's Manual, CMU-CS-81-135, July (1981)

[松本87]松本: 非単調推論とその応用, 電子通信学会誌 Vo.70, No.8, pp.804-807(1987)

[McDermott 83]D.McDermott : Contexts and Data Dependencies :A Synthesis,IEEE Trans., Vol. PAMI-5,No.3,pp.237-246(1983)

[Reiter 80]R.Reiter: A Logic for Default Reasoning, Artificial Intelligence,13,pp.81-132,(1980)

[Satty 80]T.L.Saaty: The Analytic Hierarchy Process, McGraw Hill(1980)

[新谷85]新谷,片山: 知識テーブル(その実現)ー知識ベースにおける知識記憶管理方式とその応用ー,情報31回全国大会,1P-8,(1985)

[新谷86a]新谷,片山, 平石: 問題解決環境KORE(その2)ー知識記憶利用機構KORE/EDENとその応用ー,情報32回全国大会,5L-9,(1986)

[新谷86b]新谷, 片山, 平石, 戸田: 論理型言語によるハイブリッドな問題解決支援環境KOREの設計-

知的意思決定支援システムへの接近-, The Logic Programming Conference 86, 予稿集pp.43-50(1986)

[Shintani 86c]T.Shintani: Knowledge Table: An Approach to Knowledge Utilization Mechanism for Knowledge Base, IIAS-SIS FUJITSU LIMITED Research Report No.70, p38(1986)

[新谷 87]新谷:推論エンジンKORE/IE -反駁メカニズムに基づく高速な推論エンジン-, The Logic Programming Conference 87, 予稿集pp.233-242,(1987)