知識ベースにおけるデフォルト推論システムへの接近†

新谷虎松††　溝口文雄††

知識ベースにおける推論システムには，さまざまな知識が用いられる．とくに知識が不十分な場合でも適切な推論が必要である．著者らは，このような不十分な知識のもとでの推論を計算機上に実現するために知識記述のためのフレームの概念[14]にもとづいた推論システム FDRS (Frame based Default Reasoning System) を次の点を考慮して試作した．1)推論結果の一貫性をチェックするための機構を有すること．2)ダイナミックに，ルールを構築および実行するためのルール適用に関するルール制御構造を有すること．3)推論プロセスをシステム自身が利用できるように data dependency を有した事実の記述方法を実現すること．FDRS は不十分な知識のもとで人間が行う推論の一部分を実現しているのにすぎないが，知識を柔軟に記述し利用できる能力は上記の機能の有効性を示している．

1. まえがき

近年，1980年前後，知識の利用について，その重要性がクローズアップされてきた[13),14)]．この知識の利用においては推論システムに不十分な知識のもとでの推論能力をもたせることが重要課題である[1),4)]．不十分な知識のもとでの推論を実現するためには「Pが無矛盾ならばQと帰結せよ」といった非単調な推論ルールが必要になる．非単調なルールを実際に計算機上に実装するためには信念の翻意を扱うなどのいろいろな問題点が生じる[6),13)]．たとえば，Doyle[3)] は非単調なルールを扱う問題解決システムを提案している．そのシステムは非単調な推論ルールを効果的に扱うために信念を扱う機構をシステムのなかで推論部から独立させている．この機構は TMS (Truth Maintenance System) と呼ばれる．TMS の理論はまだ十分にはうまくいっていないが，非単調な推論を扱う著者らのシステムに重要な一連のアイデアやテクニックを提示している．一方非単調な推論ルールを論理的に扱う研究が Reiter[9)] や McDermott[7),8)] らによってなされている．Reiter は非単調な推論ルールを「もし与えられた知識ベースからある情報が推論されなければ…と帰結せよ」として形式化した．この Reiter の形式化にはどのようにしたら非単調な推論ルールを計算機上に具体的に実現していくかということは述べられていないし，互いに影響しあう非単調な推論ルールの状態をも十分には定義していない．しかしながら，知識が不完全なために明示されない事柄をデフォルト値として扱うこの Reiter のデフォルト推論のための論理は，そのデフォルトルールの形式化が簡明に記述されており，計算機上に実現しやすく，著者らのシステムのフレームワークとなっている[11),12)]．

著者らは，不十分な知識のもとでの推論を実現するために以上の人工知能研究の概観からフレームをベースにした FDRS (Frame based Default Reasoning System) を開発した．FDRS は TMS の概念を導入した非単調推論システムであり，フレームをベースにした非単調推論ルール記述言語である．本論文では，このシステムの概要とその有効性について報告する．

2. FDRS の概要

FDRS は Reiter のデフォルト理論の一対の組である (D, W) におけるデフォルトルールの集合であるDを推論規則構成部で実現し，wff の集合である W を事実構成部で実現する．デフォルトルールは一般に次の形をしている．

$$\frac{\alpha(\vec{X}) : M\ \beta(\vec{X})}{\gamma(\vec{X})} \qquad (1)$$

ここで $\alpha(\vec{X}), \beta(\vec{X}), \gamma(\vec{X})$ はすべて第1階の式であり，$\vec{X}$ は X_1 から X_m までの自由変数を示す．式(1)において $\alpha(\vec{X})$ はルールの必要条件であり $\beta(\vec{X})$ は一貫性をチェックされるものであり，$\gamma(\vec{X})$ はルールの結論である．そして M は一貫性をチェックするオペレータである．このデフォルトルールは次のように解釈される．

「$\alpha(\vec{X})$ が信じられており，$\beta(\vec{X})$ が無矛盾ならば $\gamma(\vec{X})$ と帰結してもよい．」

† An Approach to Design of Default Reasoning Systems in Knowledge Base by TORAMATSU SHINTANI and FUMIO MIZOGUCHI (Department of Industrial Administration, Science University of Tokyo).

†† 東京理科大学理工学部経営工学科

FDRS では式(1)で示されるルール形式よりも次の式(2)に示されるノーマルデフォルトルールをその簡明さのためルール形式の基本形態とする.

$$\frac{\alpha(\vec{X}):M\ \gamma(\vec{X})}{\gamma(\vec{X})} \qquad (2)$$

式(2)は式(1)とちがい一貫性がチェックされるものと結論が同じものである. 式(2)を用いると, たとえば鳥が一般に飛ぶという属性は次の式(3)で示せる.

$$\frac{\mathrm{BIRD}(X):M\ \ \mathrm{FLY}(X)}{\mathrm{FLY}(X)} \qquad (3)$$

式(3)は, 以下の内容を表現している.

「X がもし鳥で, X が飛べるという仮定が無矛盾ならば X は飛べると推論する.」

この式(3)は FDRS で以下のように記述する.

```
(-* (:K (BIRD :X))
            (THEN (ASSUME (FLY :X))))
```

FDRS において式(2)で示されるデフォルトルールは, 二つの関数 (assert and assume) と推論部で実現する.

一般に式(2)における $\alpha(\vec{X})$ は関数 assert で生成し, $\beta(\vec{X})$ は関数 assume で生成する. 関数 assert は事実を主張する際に用いられ, 関数 assume は不確かな事実を主張する際に用いる (これら関数の詳細は後章で述べる). FDRS では以上のようなデフォルトルールを実現するために, 以下の主要機能がある.

1° data dependency 保持機能
2° 知識ベース一貫性保持機能
3° ルール適用に関する機能

2.1 data dependency 保持機能[2),5)]

FDRSで扱う個々の事実はノードで表現する. ノードはノードが示す事実とその事実が導かれた経過を記録する data dependency とで構成される. **図1**でこの関係を示す. 図1は A-00015 なるノードが, このノードが表す (IS THE BLOCK A HEAVY) という事実と (F-00014 A-00013) なる data dependency とで構成されていることを示している. この data dependency は A-00015 なるノードが F-00014 と A-00013 というノードから導かれていることを意味する. data dependency は本システムにおいて以下の機能を実現するための有効な情報である.

1) 推論プロセスや推論によって導き出された結果の説明を引き出すことができる.
2) 新しい知識が知識ベースに付加されたり, ある知識が知識ベースから削除された場合その知識に関与する前知識 (preknowledge) を変更することができる.
3) 矛盾した結果を導き出すのに起因した都合の悪い仮定を見つけることができる.

以上の機能を達成するには data dependency が循環構造になると都合が悪い. この循環構造はたとえば, 以下のような推論ルール群があると生じる.

1° IF (IS :X DOG)
THEN (IS :X MAMMAL)
2° IF (IS :X MAMMAL)
THEN (IS :X ANIMAL)
3° IF (IS :X ANIMAL)
THEN (IS :X MAMMAL)

ここで :X は自由変数である. いまここで (IS POCHI DOG) を与えると, まず1°のルールが起動され :X に POCHI が代入され (IS POCHI MAMMAL) が得られる. すると2°のルールから1°のルールを適用した場合と同様に, (IS POCHI ANIMAL) が得られる. 引き続き3°のルールが適用され (IS POCHI MAMMAL) が得られる. この結果, この時点で**図2**で示すようなルールにおける循環構造ができる. このことは, data dependency における循環構造も

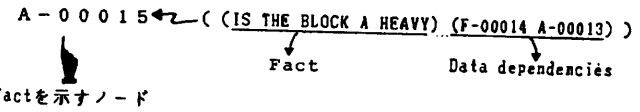

図 1 FDRS における事実の表現
Fig. 1 Representation of facts in FDRS

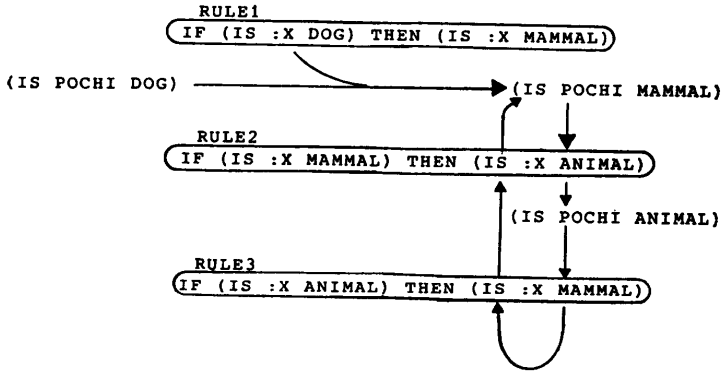

図 2 ルールにおける循環構造
Fig. 2 Circular struture in rules.

できることを示す．以上のような循環構造は生成されるノードの示す事実がすでに存在していた場合，新たにノードを生成しないことにより，システム内に生じることを回避する．上記の例では3°のルールが適用された場合，(IS POCHI MAMMAL) を示すノードを生成しないことで循環構造ができることを回避する．

2.2 知識ベース一貫性保持機能

非単調な推論を実現するためには非単調な推論ルールが適用され，導き出された事実が知識ベースに対して矛盾を生じさせた場合，その矛盾を解決するための知識ベース一貫性保持機能が必要である．このような知識ベース一貫性保持機能を FDRS では KGC (Knowledge base Garbage Collector) で実現する．KGC は手続き的付加としてシステム内に存在し，矛盾が生じた場合に，そのつど活性化され矛盾の解決を図る．

本システムにおける矛盾は矛盾を規定した推論ルールが起動して生じる．そして矛盾の解決とは，その推論ルールの前提条件をチェックし修正することにより知識ベースの一貫性を保持することである．

KGC は矛盾を解決するために矛盾に起因した都合の悪いデフォルト値を見つけ，このデフォルト値を否定する．本システムは KGC が，都合の悪いデフォルト値を見つけるための情報としてノードに事実ノードと仮定ノードの区別をノード生成の際，自動的に決定する．

仮定ノードはノード番号を示す数字の先頭に "A-" が付記される．たとえば仮定ノードとは A-00015 とか，A-00010 などである．仮定ノードはデフォルト値により直接，もしくは間接的に依存して生成されたノードを示す．つまり，仮定ノードの data dependency を逐次たどることにより，KGC はデフォルト値を見つけることができる（**図3** 参照）．図3においてノードAの data dependency はノードBとノードCであり，ノードCの data dependency はノードDとノードEとデフォルト値 F である．このことは，ノードAからデフォルト値 F まで一連のパスができていることを示している．一方，事実ノードはデフォルト値に依存しないノードでノード番号の先頭に "F-" が付記される．たとえば事実ノードとは F-001 とか F-120 などである．この事実ノードの data dependency には仮定ノードは包含しないのでデフォルト値までのパスは存在しない．そこで KGC は，矛盾に起因した都合の悪いデフォルト値を見つけることを仮定ノードのみに着目し，その data dependency をたどることにより達成する．ノード番号は，ノードがシステムにより推論などで自動的もしくは入力により新たに生成された場合，LISP の基本関数 gensym により自動的に決定される．このノード番号は，大きいほど新たに生成されたノードを意味する．

FDRS では最新の事実を常に疑わしいものとして扱う．それで KGC は矛盾が生じた場合，矛盾の原因と

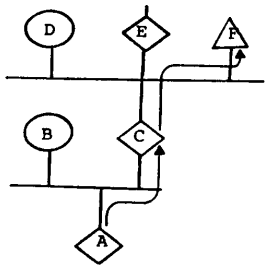

図 3 デフォルト値の探索
Fig. 3 Search of a default value.

```
(DE KGC(CONTRADICTION)
    (MARK CONTRADICTION)
    (OR (COND ((IS-DEFAULT CONTRADICTION)
               (CHECK-DEFAULT CONTRADICTION))
              (T (KGC2 (PICK-ASSUMPTION CONTRADICTION))) )
        (UNMARK CONTRADICTION) ) )

(DE KGC2(L)
    (COND ((NULL L) NIL)
          ((KGC (CAR L)) )
          (T (KGC2 (CDR L))) ))

(DE CHECK-DEFAULT(X)
    (COND ((MAKE-NOT X)(PATH-DELETE X)(END))
          (T NIL) ))

(DE MAKE-NOT(Y)
    (ASSERT2 (LIST 'NOT Y))
    (COND ((EXIST-CONTRADICTION) NIL)
          ( T T ) ))
```

(MARK X) : 関数MARKはノードX とこのノードX に依存したノードにマーク* をつける。

(UNMARK X) : 関数UNMARKはノードX とこのノードX に依存したノードのマーク* を消す。

(IS-DEFAULT X) : 関数IS-DEFAULTはX がデフォルト値なら値としてT を返し、さもなければNIL を返す。

(PICK-ASSUMPTION X) : 関数PICK-ASSUMPTION はノードX のDATA DEPENDENCY から仮定ノードをピックアップして、そのリストを値として返す。このリストの中の仮定ノードはノード番号が大きい順に並んでいる。

(EXIST-CONTRADICTION) : 知識ベースに矛盾が存在すればT さもなければNIL の値を返す。

(PATH-DELETE X) : 関数PATH-DELETE はノードX とこのノードに依存したノードを消す。

(END) : 関数END はKGC を終了させる。

(ASSERT2 X) : 関数ASSERT2 はX を評価し、その結果を主張する。（主張事実は知識ベースに保存される。）

図 4 KGC の概略
Fig. 4 A rough sketch of KGC.

なった事実を示すノードの data dependency のなかでノード番号がいちばん大きい仮定ノードからバックトラッキングをはじめ都合の悪いデフォルト値を見つける．そして最初に見つけたデフォルト値の反対命題を主張する．

このデフォルト値を見つけるためにたどった仮定ノードにはマークがつけられる．マークづけされたノードは推論部から無視され，デフォルト値の反対命題を主張したことによる推論連鎖には用いられない．推論連鎖とは，ある事実が知識ベースに加えられたことにより次々と行われる一連の推論を示す．

着目したデフォルト値の反対命題を主張することにより推論連鎖がおこり，その結果，再び新たな矛盾が生じたなら，マークづけされたノードのマークを消去し新たに仮定ノードにマークづけしながらバックトラッキングを行い，次のデフォルト値を見いだし，その否定を主張する．KGC は当初の矛盾が解決されるまで当初の矛盾に関与した複数のデフォルト値のなかの一つのデフォルト値を次々に見つけ，これらのデフォルト値のチェックを行う．KGC の概略を LISP で表現すると**図4**のようになる．図4に示すように KGC は再帰的に用いられ，デフォルト値を見つけたなら(関数 is-default を用いる)，そのデフォルト値の否定を主張し（関数 make-not を用いる)，推論連鎖によって新たな矛盾が生じなければ(関数 exist-contradiction を用いる）着目しているデフォルト値を都合悪いものとして修正し，このデフォルト値に依存していた仮定ノードをすべて消去し（関数 path-delete を用いる)，関数 end により終了する．矛盾が新たに生じれば，関数 KGC が再帰的に呼ばれ，当初の矛盾を解決するための他の可能性を調べる．KGC では，デフォルト値を見つけるための探索が縦型になっている．したがってあるところまで探索してデフォルト値を見つけ，このデフォルト値の否定を主張することにより当初の矛盾が解決しなかった場合，途中まで戻り，他の可能性を調べていく自動バックトラッキングが行われる．

その結果，矛盾が解決されるまで，当初の矛盾に関与したあらゆるデフォルト値が調べられる．

図5は KGC がノード名 A-12 なるデフォルト値をバックトラッキングにより見つけたことを示す．途中のノード名 A-14 なるデフォルト値はすでにチェックが終了し，当初の矛盾を生じさせたデフォルト値でないことを示す．図5に示すようにバックトラッキングの際仮定ノード A-40 の data dependency (F-20 と A-30 と A-35) をたどる場合には，ノード番号がいちばん大きい仮定ノード A-35 が，仮定ノード A-30 に先がけてチェックされる．

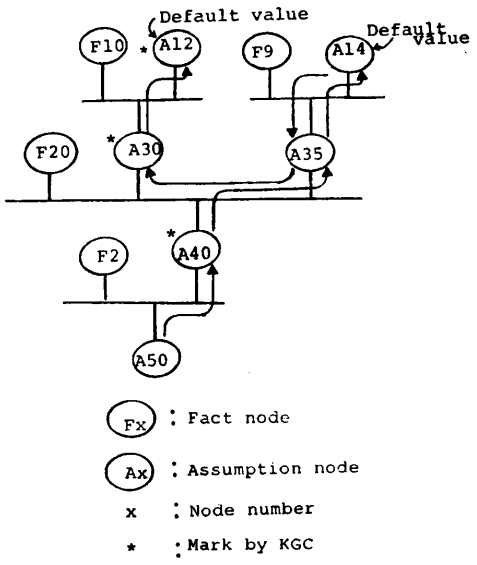

図 5 KGC の行動
Fig. 5 Action of KGC.

2.2.1 KGC の限界

本システムにおいて矛盾は不適切なデフォルト値が原因で生じる．それで，前節で述べた KGC の機能で矛盾の解決を図る．KGC は常に矛盾が演繹されたときに起動され，その時点での知識ベースの一貫性を保持する．矛盾の演繹は，たとえば（簡便に記すと）以下のような推論ルールで達成する．

IF (A & (NOT A)) THEN
(ASSERT CONTRADICTION)

このルールの前提が矛盾を生じさせた条件であり，結論が，KGC を活性化させるためのキーワードである．KGC は活性化されると KGC を活性化させたルールの前提を調べ，その前提に関与するデフォルト値をチェックする．KGC が起動し，最新の知識ベースの一貫性が保持されても，知識ベースには，以下の二つの潜在的矛盾が存在する場合がある．

1) 矛盾を演繹する推論ルールがないので，KGC が起動されずに存続する矛盾
2) 都合の悪いデフォルト値によって将来，推論連鎖の結果，生じる矛盾

後者 2) の潜在的矛盾は知識ベースにデフォルト値が含まれる限り存在する．KGC は一般に後者 2) のデフォルト値による矛盾を解決するために起動され，常

に最新の知識ベースの一貫性の保持に努める．KGCは矛盾を解決するために矛盾に関与したデフォルト値を次々にチェックする．このチェックはデフォルト値の否定を主張することにより新たな矛盾が生じなくなるまで続けられる．これらデフォルト値すべてをチェックすることによって矛盾が解決できなかった場合，KGCは矛盾の解決を自動的に行うことができずユーザにいままでチェックした当初の矛盾に関与するデフォルト値のリストを上げて，矛盾解決のための処理を終了する．その場合，ユーザ自身が都合の悪いルールをルールエディタで修正し，その後に都合の悪いデフォルト値やノードを消去することにより知識ベースの一貫性を保持する．

以上のようにFDRSにおける知識ベース一貫性保持機能はKGCによって達成したが，まだ不十分である．しかしながら，KGCは非単調な推論ルールを実現するための有効な基本的機能である．

2.3 ルール適用に関する機能

FDRSにおいてはルールを適用する際にどのルールが優先的に用いられるべきかというルール制御構造がある．この構造は，どのルールが優先的に用いられるべきかというルール間の依存関係を表現するので，この制御構造をrule dependencyと呼ぶ．rule dependencyはルールの優先度に着目して，システムが自動的に決定する．ルールの優先度は以下の項目に依存する．

1) ルールパターン（ルールが起動するための条件で，合鍵の役割をもつ文字列）に変数をより多く含むものほど優先度が小さい．
2) 前提となるルールパターンの数が多いほど優先度が大きい．
3) 頻繁に用いられるルールほど優先度が大きい．

rule dependencyには上の1)～3)までの情報を記録するための二つのrule dependencyがある．一つは，FDRSの推論規則構成部の全体の構造を決定するGRD (General Rule Dependency) である．GRDは推論規則構成部を複数のルールからなる一つのリストと見なし，個々のルールを適用度の高い順にならべたものである．このGRDの先頭から順に適用できるルールを検索することによって，適用度の高いルールが最初にチェックされる．そして，適用されたルールは常にGRDの先頭に移動される．GRDにより頻繁に用いられるルールほど優先的に用いることができる．

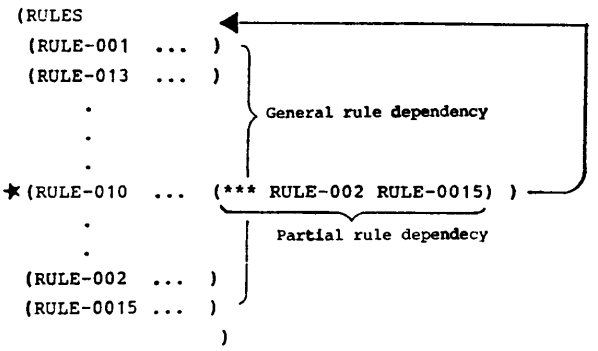

図 6 GRD と PRD
Fig. 6 GRD and PRD.

他の一つはPRD (Partial Rule Dependency) で，おのおののルールが含んでいるルールパターン同士の類似性を考慮し，ひとまとめにして，そのグループの中から最も優先度の高いルールから適用しようとするものである．PRDの実装は上述のルール優先度決定のための項目で1)と2)を実現する．PRDは各ルールのなかに一つのリストとして与えられる．このリストのなかにはルール名が優先度の小さい順に並ぶ．たとえばPRDは以下のように与えられる．

(RULE-0012 IF A THEN B (*** RULE-0010))

ここで***とRULE-0010とでなるリストがPRDである．このPRDによりルール名RULE-0012のルールの前提条件Aが満足してもすぐに適用されず，ルール名RULE-0010のルールがまずチェックされる．そして，その適用が失敗した後に***が示すルール名RULE-0012のルールが適用される．PRDのなかで***はこのPRDを保有しているルール名を示す．GRDとPRDの関係を**図6**で示す．図6ではルール名RULE-010なるルールが図で示されたようなPRDを有していることを示す．RULE-010および，そのPRDに記録されたルール名RULE-002, RULE-0015のルールが一つでも適用されるとルール名RULE-010のルールは，GRDの先頭に移動される．その結果，ルール名RULE-010でグループ化されたルール群は，他のルールより優先的にチェックが行われる．

rule dependencyの実装により，本システムではシステムの実行中にルールの構築と利用ができるダイナミックなルール記述が可能である．

3. FDRS の構成

FDRSは，いくつかのスロットの集合からなるフ

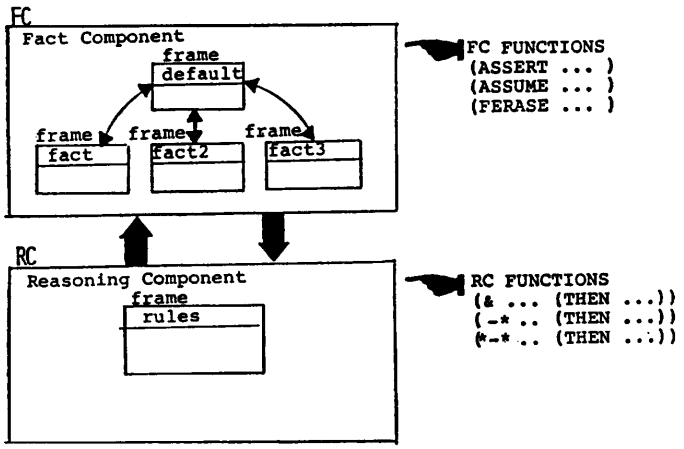

図 7 FC と RC 間の情報交換
Fig. 7 Communication between FC and RC.

```
(Frame-name
   (AKO (VALUE (DEFAULT)))
   (DD-name1 (Node-name (Statement)
                        ((Data dependencies))))
      :
      :
      :
   (DD-nameN (Node-name (Statement)
                        ((Data dependencies)))) )
```

```
⟨Frame-name⟩::= FACT | FACT2 | FACT3
⟨DD-name⟩    ::= ⟨name of Data dependencies⟩
⟨Node-name⟩  ::= F-0001 | ... | A-0008 | ...
```

図 8 FC のシンタックス
Fig. 8 Syntax of FC.

```
(RULES
   (RULE-pattern-list1 (PM-method ((Conclusion))
                                  ((Rule dependencies))))
      :
      :
      :
   (RULE-pattern-listN (PM-method ((Conclusion))
                                  ((Rule dependencies)))))
```

```
⟨RULE-pattern-list⟩::= (⟨Rule-pattern1⟩ ··· ⟨Rule-patternN⟩)
⟨Rule-pattern⟩     ::= ( ⟨node-pattern⟩⟨pattern⟩)
⟨pattern⟩          ::= ⟨ルールが起動するための条件で合鍵の役割を持つ文字列⟩
⟨node-pattern⟩     ::= :⟨文字列⟩
⟨PM-method⟩        ::= & | -* | *-*
⟨Conclusion⟩       ::=⟨LISP Expression⟩
```

図 9 RC のシンタックス
Fig. 9 Syntax of RC

レームを記述の基本としている．フレームの形式としては，FRL[10] の表現を基盤にする．本システムは二つの構成要素からなっている．一つは事実を示すノード群を記録するためのもので FC (Fact Component) と呼ぶ．他の一つは推論ルールを記録するための構成部で RC (Reasoning Component) と呼ぶ．FC と RC の関係は**図 7** で示す．FC は四つのフレーム (フレーム名 DEFAULT, FACT, FACT 2, FACT 3) で構成され，そのなかで DEFAULT が他の三つのフレームの上位フレームとなる．フレーム DEFAULT には手続き的付加が記述されている．この手続き的付加には FC で矛盾が生じた場合に起動する KGC がある．FC において，常に用いられるフレームは FACT であり，FACT 2 と FACT 3 は KGC が用いる．RC はフレーム RULES で構成する．各コンポーネントは固有の関数で操作される．FC では関数 assert, assume で新たに事実を付加し，関数 ferase で FC 中の事実を消す．FC 関数実行後には RC 中のルールがテストされる．また RC では，RC 関数でルールを記述する．新たにルールが RC に付加されると，そのルールが適用可能な場合，起動し，FC にその結果を記録する．FC と RC は以上のようにシステムの実行にともない情報を交換する．FC と RC の中で用いられるフレームの表現形式はそれぞれ**図 8** と**図 9** に示す．図 8 で第 1 スロット名の AKO は，この値の上位フレーム DEFAULT とリンクを生成するために用いる．

3.1 RC 関数

RC 関数には前提と結論があり，プロダクションシステムにおけるプロダクションとして見ることができる．RC 関数のシンタックスは**図 10** で示す．図 10 で複数のルールパターンが前提であり，THEN 以降が結論である．RC 関数は評価された後に RC にルールとして保存される．

ルールパターンには変数が含まれる．変数は一つのルールの中で固有の局所変数である．変数はパターンマッチングの

```
(PM-method rule-pattern1 rule-pattern2…rule-patternN
                    (THEN conclusion ))

    ⟨PM-method⟩::= & | -* | *-*
    ⟨rule-patterni⟩::=⟨ith rule pattern as a
                        trigger fact⟩
```

図 10 RC 関数のシンタックス
Fig. 10 Syntax of RC functions.

```
(FC-function  ( statement ) (DD-name Data dependency  ))
                                        DD-set
    ⟨FC-function⟩::= ASSERT | ASSUME | FERASE
    ⟨DD-name⟩     ::=⟨A name of Data dependency⟩
    ⟨statement⟩   ::=⟨a fact for assertion⟩
```

図 11 FC 関数のシンタックス
Fig. 11 Syntax of FC functions.

```
(& (:K (IS THE BLOCK :X RED))
   (THEN (ASSERT (IS THE BLOCK :X HEAVY)
                 (DEDUCTION-BY :K)) ))
```

図 12 RC 関数と FC 関数
Fig. 12 Example of RC and FC function.

```
⟨RUN⟩ …①

 NOW INITIALIZING !
 ◆◆◆ OK ◆◆◆
◆◆ RULE ---②

 FILE NUMBER PLEASE !
◆◆ 1

" RULE OF BLOCK WORLD 81/12/11 "
⟨VERSION 0⟩
 FILE IS LOADED
◆◆ ⟨ASSUME ⟨IS THE BLOCK A RED⟩ ⟨PREMISE⟩ --------③

F-00011 ⟨ASSUMED ⟨IS THE BLOCK A RED⟩⟩ ⟨PREMISE⟩
A-00012⟨NOT ⟨IS THE BLOCK A RED⟩⟩
A-00013⟨IS THE BLOCK A RED⟩

          Prompting
◆◆ ⟨ASSERT ⟨ARE ALL RED BLOCKS HEAVY⟩ ⟨PREMISE⟩ ---④

F-00014 ⟨ARE ALL RED BLOCKS HEAVY⟩ ⟨PREMISE⟩
A-00015 ⟨IS THE BLOCK A HEAVY⟩ ⟨DEDUCTION F-00014 A-00013⟩
                                          ……⑤

◆◆ ⟨ASSERT ⟨NOT ⟨IS THE BLOCK A HEAVY⟩⟩ ⟨PREMISE⟩ ---⑥

F-00016 ⟨NOT ⟨IS THE BLOCK A HEAVY⟩⟩ ⟨PREMISE⟩
A-00017 ⟨? A-00015⟩ ⟨CONTRADICTION A-00015⟩ ……⑦
A-00012 ⟨NOT ⟨IS THE BLOCK A RED⟩⟩ ⟨NOGOOD A-00017⟩
A-00018 ⟨IS THE BLOCK A BLUE⟩ ⟨DEDUCTION A-00012⟩

 END OF COMPULATION
```

図 13 FDRS の実行例
Fig. 13 An example using FDRS.

際に結合および評価 (binding and evaluation) の処理が行われる．パターンマッチングの際に，最初に出現した変数は結合変数として用いられ，パターンの結合が行われる．次にこの変数が出現した場合は，評価変数として用いられて，変数の評価が行われる．本システムではアトムの先頭に：が付記されたものが変数を表す．たとえば :K や :NAME などは変数である．

THEN 以降の結論部は以下のように示せる．

(THEN S1 S2 S3…S*n*)

ここで S*i* は LISP における S 表現を示す．THEN 以降の S*i* は左から右に順次，評価される（これは LISP における関数 PROGN に匹敵する）．また，S*i* で構成されたリストが結論部にあると，このリストは LISP における COND 文に匹敵する．たとえば以下のような THEN 部があるとする．

(THEN S1 S2 S3…(Sa Sb Sc)…S*n*)

このとき (Sa Sb Sc) の部分は，次の COND 文と同じ意味である．

(COND (Sa Sb Sc))

RC 関数は図 10 で示す PM-method を意味する．RC 関数は以下の形式のルールを記述する．

1° AND 結合ルール
2° OR 結合ルール
3° 順序結合ルール

1° のルールは PM-method として & を用いる．このルールは前提となるルールパターンがすべて満足すれば（知識ベース上の事実とマッチングが成功すること），ルールが起動する．2° のルールは PM-method として *-* を用いて前提となるルールパターン群の一つでも満足すればルールが起動する．3° のルールは PM-method として -* を用いる．このルールはルールパターンが並んでいる順にマッチングされていき，すべてのルールパターンが知識ベースにおける事実とマッチングが成功すると起動する．この順序結合ルールはルールパターンの順が重要になる．それで，このルール記述は事実の順序をルール化する場合に有効である（事実の順序はノード番号で参照する）．

3.2 FC 関数

FC は FC 関数 assert, assume, ferase で操作する．関数 assert は本システムで一般に事実を主張する際に用いる．関数 assume はデフォルト値を主張する際に用いられ，後に不都合が生じたなら KGC によりチェックされる．FC 関数のシンタッ

クスは**図11**で示す．図11で示したDD-setはFC関数がルールの結論部で用いられたときとくに有効である．DD-setはルールが起動し，FC関数が評価されたときにRCがFCへ送る情報である．この情報はFC関数が評価された（ルールが起動された）前提条件を示し，FC関数によって主張された事実のdata dependencyを表す．たとえばFC関数はルールのなかで(RC関数のなかで)**図12**のように用いる．図12でFC関数assertのDD-setは(DEDUCTION-BY :K)である．図12で示すルールが起動した結果，FCにはこのルールの結論部の関数assertで主張された事実とそのdata dependencyが知らされる．この場合，data dependencyはルールの前提となった事実のノード名:Kである．:Kは変数であり，パターンマッチングの後に実際のノード名に置換される．また，このdata dependencyの名前にはDEDUCTION-BYが用いられる．

```
(RULES
   ((:M (NOT (IS THE BLOCK A RED)))
    (-* (((ASSERT (IS THE BLOCK A BLUE)
                  (DEDUCTION :M))))
        ((***))))
   (((:N (ARE ALL RED BLOCKS HEAVY))
     (:M (IS THE BLOCK :X RED)))
    (& (((ASSERT (IS THE BLOCK :X HEAVY)
                 (DEDUCTION :N :M))))
       ((***))))
   (((:N (NOT :X))
     (:M :X))
    (-* (((ASSERT (? :M)
                  (CONTRADICTION :M))))
        ((***)))))
```

図14 RCの例

Fig. 14 Example of RC.

4. 実行例

図13にFDRSの実行例を示す．図13の中で番号が示す行はそれぞれ以下の意味がある．

①FDRSの起動，②RCの呼び出し，③デフォルト値の主張，④事実の主張，⑤RCからのメッセージ，⑥事実の主張，⑦KGCを起動させる矛盾の発生

図14は図13で用いられたRCである．

図15はFDRSにおける説明機能を示した．説明機能は，着目した事実のdata dependencyを次々に調べることで達成する．図15の中で番号が示す行はそれぞれ以下の意味がある．

① ノード名 A-00015 が出現した理由を問う．

② ①で得られた説明のなかでさらにP-1という項目の説明を問う．

③ ①で得られた説明のなかのP-2で示されるルールの内容をさらに詳細に見る．

5. インプリメンテーション

FDRSは東京大学計算センターのM200H上で起動するLISPF3を用いて記述されている．

LISPF3はFORTRAN(約6,000 steps)で記述されたLISPインタプリタであり，INTER LISPのサブセットになっている．FDRSは約2,000ステップのLISPでコーディングされ，起動する際に作業領域を除いて約30kcellのLISP領域が必要である．著者らの主目的は非単調な推論を計算機上で実現し，その有効性を確認することにあるので，システムの実行効率については現在，特別な考慮を払っていない．

6. むすび

計算機による推論システムの実現には，さまざまな視点からのアプローチが必要である．本論文では人間が行う不十分な知識のもと

```
** ? A-00015 ---①

 EXPLANATION : A-00015 IS ((IS THE BLOCK A HEAVY) (F-00014 A-00013))
     P-1: (IS THE BLOCK A RED)
     P-2:  RULE : (IF (:N (ARE ALL RED BLOCKS HEAVY))
                      (THEN (IF ... ...)))
     P-3: (ARE ALL RED BLOCKS HEAVY)
 FURTHER EXPLANATION ?
** P-1 ---②

     P-1-1: ((NOT (IS THE BLOCK A RED)) (F-00011))
     P-1-2: ((ASSUMED (IS THE BLOCK A RED)))
 FURTHER EXPLANATION ?
** P-3

 THIS IS PREMISE !
 FURTHER EXPLANATION ?
** P-2 ---③

 THIS IS FOLLOWING RULE EXPRESSION :
(IF (:N (ARE ALL RED BLOCKS HEAVY))
    (THEN (IF (:M (IS THE BLOCK :XXX RED))
              (THEN (ASSERT (IS THE BLOCK :XXX HEAVY)
                            (DEDUCTION :N :M))))
 FURTHER EXPLANATION ?
** P-1-1

     P-1-1-1: ((ASSUMED (IS THE BLOCK A RED)))
 FURTHER EXPLANATION ?
** NO
```

図15 FDRSにおける説明機能

Fig. 15 Function to generate explanations in FDRS.

での推論に着目し，どのようにしたら，このような推論を計算機上に実現できるかという問題を論じ，フレームをベースにした推論システムである FDRS を試作した．FDRS は人間が行う不十分な知識のもとでの推論の一部分を扱ったにすぎないが，知識を柔軟に記述し，利用できる FDRS の機能は，知識ベースにおけるコンサルテーションシステムの構築の際，基盤のシステムになりえることを示している．

参考文献

1) Bobrow, D. G. and Winograd, T.: An Overview of KRL: A Knowledge Representation Language, *Cognitive Sci.*, Vol. 1, No. 1, pp. 3-48 (1977).
2) Charniak, E., Riesbeck, C. K. and McDermott, D. V.: *Artificial Intelligence Programming*, pp. 193-226, LEA Publishers, New Jersey (1980).
3) Doyle, J.: Truth Maintenance System for Problem Solving, MIT, AI-TR-419 (1978).
4) Fikes, R. E.: Odyssey: A Knowledge-Based Assistant, *Artif. Intell.*, Vol. 1, No. 16, pp. 331-336 (1981).
5) Kleer, D., Doyle, J., Steel, G. L. and Sussman, G. J.: Explicit Control of Reasoning, AI Memo, No. 427, MIT (1977).
6) 松本裕治：Default Reasoning と非単調論理，電子通信学会誌，Vol. 64, No. 3, pp. 313-316 (1981).
7) McDermott, Đ. V. and Doyle, J.: Non-Monotonic Logic 1, AI Memo, No. 486, MIT (1978).
8) McDermott, D. V.: Non-Monotonic Logic 2, Research Report, No. 174, Yale Univ. of Computer Science (1980).
9) Reiter, R.: A Logic for Default Reasoning, *Artif. Intell.*, Vol. 13, No. 1, 2 (1980).
10) Roberts, R. B. and Goldstein, I. P.: The FRL Primer, AI Memo, No. 408, MIT (1977).
11) 新谷虎松，溝口文雄：知識ベースにおけるデフォルト推論システムの試作，情報処理学会第23回全国大会講演集，pp. 847-848 (1981).
12) 新谷虎松，溝口文雄：知識ベースにおけるデフォルト推論システムの試作(その2)，情報処理学会第24回全国大会講演集，pp. 881-882 (1982).
13) 田中穂積他：自然言語処理技術と言語理論（第5章）問題解決と推論—Default Reasoning，電子技術総合研究所調査報告，No. 205, pp. 98-107 (1981).
14) Minsky, M.: A Framework for Representing Knowledge, in Winston, P. (ed.), The Psychology of Computer Vision, pp. 211-277, McGraw-Hill, New York (1975).

(昭和57年5月10日受付)
(昭和58年3月11日採録)