知識工学と人工知能　36－8
（１９８４．９．１９）

Ｃｏｎｓｔｒａｉｎｔに基づく論理データベース管理について

宮地 泰造　國藤 進　古川 康一　北上 始
（財）　新世代コンピュータ技術開発機構

１．はじめに

DBシステムが人間にとってより良いアシスタントになるためには更に知的な機能が有用であり、演繹的な質問応答機能等の知的検索・更新機能を持つDBシステムを〝論理DBシステム〟と呼ぶ[M84]。そのようなシステムの重要な知的機能の１つとして、利用者の目的・意図を反映しつつ一貫性を保持した状態でデータや知識を収集・管理する機能を挙げることができる。　この機能によると、利用者やデータベース管理者の目的・意図が論理DBシステム（DBS1）に明示されることにより、一般の利用者やDBS1以外の論理DBシステムの利用者はDBS1に対してその内容を容易に問い合わすことができ、DBS1の別の分野への応用も可能になる。更に、論理DB管理システムが指示された目的・意図に従って、知識を同化し（assimilate)[BK82]、一貫性を保持しつつ知識を管理することにより、利用者は必要な知識を他のDBシステムから容易に入手可能となる。ここでは、人間は実世界の個々の対象物についての自分の意図を論理DBシステムに対して宣言的に記述すればよく、その後は、論理DBシステムによる知識の同化・管理が適切に行なわれているか否かをチェックしていればよい。　以上のような知的な機能を実現するためには、実世界における一貫性や実世界に存在する個々の対象物の由来・意味・機能を利用者の目的・意図に沿って論理DB内に記述できることが重要であるが、その記述事項の研究は十分には行われていない。関係する研究として"Constraint"に関する研究がいくつかの応用分野で行われており[Ca76,NG78,SS80,BP83]、"Constraint"を用いた意味表現は有効であると考えられる。

また、このような知的な論理DBシステムに適した知識同化の手法[M84]の研究も必要である。

本稿では、第２章では、まず実世界を論理DBに反映するために必要な意味記述能力について、論理DBにおいて必要とされる"Constraint"の意味記述能力を明確にすることにより議論する。さらに、"Constraint"を対象(Object)とする対象指向（Object Oriented)にもとずいた論理DBシステムにおける知識同化の手法を提案し、知識同化の過程を議論する。　第３章では、宣言的表現を許す論理型言語Prologによる知識同化論理データベース・システムのインプリメントを行ったので報告する。

関連する研究には、知識の同化処理においてPrologと論理DBとが相性が良いことを示した研究[M84]、Prologによる否定的知識の処理系の容易な実現法[SM84]、知識の調節(accommodation)の処理の研究も試みられている[Ki84]。その他、Integrity Constraintの研究がCadiou[Ca76],Nicolas[NG78]などにより行われている。DBのモデルの研究には、[Co70],[Ch76],[S81],[HM81]がある。

２．　論理データベースにおける意味・相互関係表現

実生活に存在する様々の事物を 'オブジェクト' と呼ぶことにする。オブジェクトは時々刻々と変化し 'アクション' をも起こす。また、オブジェクトはアクション自身でもある。オブジェクトの意味やオブジェクト間の関係はその変化やアクションが起きる時にその重要性を発揮し、アクションの価値評価に従いその価値が決定される。よって、オブジェクトの由来や意味をアクションや変化がおきる時のシーン・条件や重要度とともに記述することが非常に重要となる。また、この記述により、個々のオブジェクトやデータは意味・由来を持つことができ、１個の知識としてより広い分野に利用可能となる。

最初に、オブジェクトの記述事項を論理DBにおける"Constraint"を分析することにより明確化して、実世界を論理DBに反映する方法を検討する。

２．１　Ｃｏｎｓｔｒａｉｎｔの規定内容

実世界のConstraintは、実世界におけるオブジェクトの存在やアクションの正当性を規定する役割を有している。

実世界におけるオブジェクトやアクションは、オブジェクト・レベルの言語により記述可能であり、“オブジェクト世界の知識（オブジェクト知識）”と呼ぶ。それに対し、実世界のConstraintはオブジェクト知識やオブジェクト知識間に成立する必要条件を規定し、“メタ知識”と呼ぶ。

２．１．１　Ｃｏｎｓｔｒａｉｎｔの定義

Constraintは、アクションをも含むオブジェクト知識やオブジェクト知識間の正当性を規定する叙述であるので、アクションが起きるときの状態の遷移を記述できなければならない。　逆に、状態の遷移による記述は、手続きによる記述に比較して、宣言的に行えるだけでなく理解しやすいという長所があるので、“基本Constraint”は状態の遷移に伴う必要条件を記述できる次の形式で表現する。

```
<Constraint>::=<Pre Conditions>,
    {<Pre State>:<Pre State Descriptions>  ->>
     <Post State>:<Post State Descriptions> },
     <Post Conditions>
```

ここで、<Pre Condition>,<Pre State Description>,<Post State Description>, そして<Post Condition>は、DEC-10 Prolog と同等の機能を有する論理型プログラミング言語のゴール列で記述される。 基本Constraintの手続き的な意味は以下の通りである。

“あるアクションが対応する<Constraint>を満足する”とは、以下の処理が対象世界において終了したときをいう。

1) <Pre Conditions>が満足されているか検査する。
2) 1)が満足されたときにPre-State を決定するために<Pre State Descriptions>を評価する。
3) <Pre State Descriptions>の評価により決定された<Pre State> をフェッチする。
4) Post-Stateを決定するために<Post State Description>を評価する。
5) <Post State>を生成して、<Pre State> を<Post State>に置換える。
6) <Post Conditions> が新しい対象世界で満足されているかを検査する。

つぎに、対象知識や対象知識間の正当性を規定する"Object Constrasint"(OC)は、基本Constraintである<Constraint>を用いて次のように定義できる。

```
<OC>::= <Pre Constraints>'('<Constraint>')'
        <Post Constraints>
<Pre Constraints>  ::= nil  | <OC>
<Post Constraints> ::= nil  | <OC>        …(S1)
```

ここで、<Pre Constraints> はある<Constraint>に先行してチェックされるConstraints に関する必要条件の記述であり、<Post Constraints>はある<Constraint>に後続するConstraints に関する必要条件の記述である。 よって、“<OC>が満足される”とは、ある<Constraint>に先行するConstraintsの条件が満足されて当該Constraintが満足され、後続するConstraints の条件が満足されることである。

この<Constraint>の前後に必要な制約条件の記述を許すことにより、対象知識間の関係が記述可能になる。

(例 1) ある従業員のランクがA から管理者(MC)に昇進し、給与も上がる(a) 。この地位の昇進に伴い権限が高くなる(b) 。更に、電話等の所持品が向上する(c) 。ここで、a,b,そしてc 各々に対してPre-Comnstraint,Constraint,Post-Constraintが規定される (3.2(B)参照) 。

2．1．2 Constraintの規定内容の分析

Constraintの概念は次の２つに大別できる。

a)存在制約 (Existencial Constraint(EC))
オブジェクトが対象世界に存在しても対象世界に矛盾が起きないための必要条件の規定。

b)動作制約 (Action Constraint (AC))
オブジェクトが状態の推移する対象世界に存在するために対象世界全体で満足されるべき状態の推移に関する必要条件の規定。

(１) 存在制約 (Existencial Constraint(EC))

存在制約は、オブジェクトが対象世界に存在しても対象世界に矛盾が起きないための必要条件の規定であり、オブジェクトが対象世界に単に静的に存在するための必要条件であるので、次のように定義される。

```
<EC>::= {<State>:<State Description>},
                <Consistency Conditions>
```

ECでは、オブジェクトが対象世界に存在する状態の記述<State> が重要であり、Constraintの必要条件は対象世界に矛盾が起きないための必要条件:<Consistency Conditions> となる。ECは<Pre Condition>,<Pre State> に独立な基本Constraintであるとみなせる。ECの具体的な機能は、対象世界の枠組みを定義する必要条件を与えるとともに対象世界の静的な状態を規定することである。 ECの概念はデータベース理論における“一貫性制約(Integrity Constraint)”と同一である。

(２) 動作制約 (Action Constraint (AC))

動作制約は<OC>により記述できる。動作制約は、以下に示す４つの異なる側面を有す。

(a) 推移性制約(Transition Constraint(TrC))

一般に、対象世界における２個以上のアクションは時間的に順序づけられて生起する。推移性制約は、この順番に起るアクションの順序に関する正当性を規定する。対象世界に存在するアクションが他のConstraintと時間的順序に独立である場合、ACは<Constraint>と同一になる。

(例 2) 推移的制約の例としては例 1がある。

(b) 依存性制約(Dependency Constraint(DeC))

対象世界に２個以上のアクションが存在する場合、アクションは他のアクションにより新たに決定された具体値(Instance)に依存して動作が決定されることがある。依存性制約は、このように他のアクションにより新たに決定され

た具体値に依存して動作が決定されるアクションの依存関係の正当性を規定する。
（例 3）車を買い替える時、変速機がマニュアルからオートマチックに変ったことにより、車の操作方法が変る。

(c) クラス制約(Class Constraint(CIC))
対象世界に存在するクラスや仮想クラスの全ての要素に同時に有効となるConstraintを“クラス制約”と呼ぶ。クラス制約は、対象となるクラスの個々の要素に適応されるのではなく、クラスに対して適用される点において推移性制約とは異なる。
（例 4）ランクＡのある従業員の給与が10% 上がるときは、ランクＡの全ての従業員の給与が10% 上がる

(d) 時間制約(Time Constraint(TiC))
時間制約は時刻や相対的な時間に関する正当性を規定する。
（例 5）Constraint(c2)はConstraint(c1)が適用された３分後に適用される。
（例 6）Constraint(ce8) は平日の８時に毎日適用される。

２．２　論理データベースにおける複合世界と意味表現

実世界を論理DBに反映するために、本節では両者の対応を考察する。一般に、DBは複数個の利用目的を有しているので個々の利用目的に対応して１つの世界を有し、これを〝単位世界〟と呼ぶ。また、１個以上の単位世界の集合を〝複合世界〟と呼ぶ。同一の性質を複数個の単位世界が共有するときに、その性質に対して複合世界が形成される。複合世界は部分実世界を表現する。DBは複合世界の集合体であり、利用者が対象としている実世界を表現する(Fig.2.1参照)。実世界のオブジェクトはDB内では〝オブジェクト像(Reflected Object 略して"RO")〟と呼ぶ。一般に、ROは異なる単位世界に対して異なる意味・側面を有し、単位世界の具体的な性質はROの意味やRO間の関係に依存する。よって、単位世界の具体的な規定は、ROが満足すべき必要条件を各々の単位世界に対して規定してやればよい。

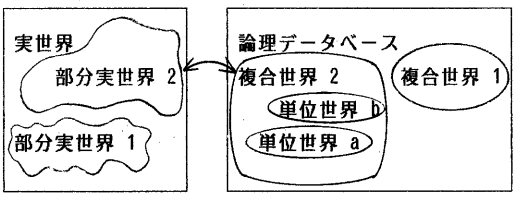

Fig. 2.1　実世界と知識ベース内の世界

２．３　制約意味モデル

論理データベースに実世界の意味を反映するためには、Constraintを導入したデータベース・モデルが有効である。
本節では、部分実世界を論理DB中の複合世界に反映するために、論理DB中の関係(Relation)及び関係間の意味をConstraint(OC)により表現するデータベース・モデル：制約意味モデル（Constraint based Semantic Model for Logic Databases 略して"CSM")を提案する。CSM は複数の関係における静的および動的な意味を対象指向的に表現可能にするモデルである。

［制約意味モデルの定義］
D1,D2,…,Dn(N>0)を定義域(Domain)とする度数ｎの関係Ｒは直積×{Di ： i=1,2, …,n} の部分集合として定義される。この関係Ｒを解釈モデルとして、一般に論理データベースＬが一階ホーン論理のwell-formed formula を用いて定義される。この様な論理データベースの集合{L1,L2, …,Lp}に対して、２項関係：< {L1,L2, …,Lp}, {OC1,OC2, …,OCq} > (ここに、OCi は 2.1.1の(S1)で記述され、その解釈モデルは上述の関係Ｒに帰着される)によって論理データベースの制約意味モデルが規定されるものとする。

CSM では大別して次の２種類の意味表現が行える。
1)論理DB中における複数関係間の静的な意味・関連と複数関係間に起きる動的な意味・変化を、シーン，アクション，シーンのシーケンス等の叙述としてOCに記述することにより、論理DB中の各複合世界における必要条件を宣言的に叙述することができる。
2)論理DB内の単位世界，複合世界間に起きる動的な意味・変化をOCにより叙述できる。

CSM により、実世界に存在する多様な意味・関連を実世界の変化に対応づけて表現できる理由としては次の４つの機能の実現が上げられる。
1)メタ知識(OC)の宣言的表現が可能である。
2)個々にOCを記述でき、OCs を追加することにより管理できる実世界を容易に拡張できる。
3)Extension とIntension とを自由に組合せて意味ネットワークが表現できる。
4)種々の抽象化のレベルにおける概念の記述が、リレーション，属性，インスタンスを用いて表現できる(任意のオブジェクトを関係名，属性名，インスタンスのどれとしても記述できるという理由による）。

Fig.2.2 は、SRq が複数関係間の関連を、属性，Instance間の叙述により規定していることを示す概念図である。

つぎに、従来の研究におけるConstraintの記述能力と意

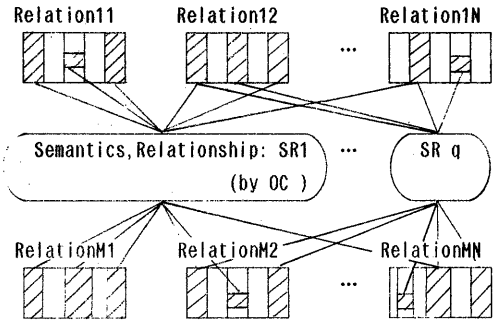

Fig. 2.2 Representation of semantics and relationship by OC(EC,AC)

味制約モデルにおけるConstraintの記述能力との比較を示す。 従来のDBにおけるConstraintの研究としては"Integrity Constraint","Integrity Checking","Trigger" がある。Nicolas らによる"Integrity Checking"の研究[NG78]では"State Law" や"Transition Law"の研究として存在制約(EC)や推移性制約(TrC) の研究が行われている。"Trigger" の研究の一部は"Time Constraint(TiC)"の研究としておこなわれていた[Ca76]。意味制約モデルはConstraintに基づくので、実世界を表現するために必要な記述能力として 2.1節で明確化したEC,AC(TrC,DeC,CIC,TiC)を有すことができる。しかし、Prologで実現する意味制約モデルでは、既存Prolog処理系の機能の制限により時間制約(TiC) を実現できない。(Fig. 2.3 参照)

Constraintの制約内容					従来の研究とCSM
EC	TrC	DeC	CIC	TiC	
○	*	*	*	*	State Law
*	○	*	*	*	Transition Law
*	*	○	*	○	Trigger
○	○	○	○	○	CSM
○	○	○	○	*	PrologでCSM

Fig. 2.3 従来の研究とConstraintの制約内容

2．4 無矛盾性

知識同化の過程において、DB内に矛盾が起きないようにDBを管理することが重要である。 CSMを用いて設計されたDBにおいては、クラークの条件(Negation as Failureの健全性が保証されるための十分条件) 下[Cl78]で無矛盾性の定義をより詳細に行うことができる。"DBが無矛盾である"とはDB内に矛盾が検出されないことを言い、次の条件を満足するときを言う。 ここで、demo(W,G) は、論理学における"W⊢G "に対応し、W から Gが1階ホーン論理の意味で証明されることを意味する。

```
no_contradiction(Compound_world,ROs) ->
  demo(Finished_Constraint_List, Pre_Constraints),
  demo(Compound_world, Pre_Conditions),
  demo(Compound_world, Pre_State_Descriptions),
  demo(Compound_world, Pre_State),
  demo(Compound_world, Post_State_Descriptions),
  substitute_state(Compound_world, Pre_State,
                                        Post_State),
  demo(Compound_world, Post_Conditions),
  no_contradiction(Following_Compound_world,
                                Following_Constraint),
  not(demo(Compound_world, not(EC))).
```

< 無矛盾性の検査 >

一般に、論理DBの Consistencyを規定する Constraint (OC(AC,EC)) の間には Consistency検査に関する従属性が存在し、これを〝制約従属性：Consistency Checking Dependencies(略して、CCD)〟と呼ぶ。 すなわち、一般には、新知識がデータベースに入れられるとき、最初にチェックされるべきAC(a) が規定されていて、AC(a) には続いてチェックされるべきAC(b) の列が指定されている。ここでは、AC(a) のチェックが要求されることによりAC(b) のチェックが要求されるので、これを"AC(b) はAC(a) に従属する"と呼び、"AC(a) => AC(b) "と表わすことにする。複数個のOCにより記述されるConstraintは木を生成する。そのチェックはdepth first で管理され、ECは、ACで列挙された個々のアクションについて、対応するACがチェックされる後にチェックされる (Fig.2.4 参照) 。 さらに、利用者は、追加したOCi が論理DBにおいていかなる意味を有し他のOCs といかなる関係にあるかの明確な把握を、 CCDを眺めることにより行える。また、OCi の実世界における妥当性の評価を CCDを用いて行える。

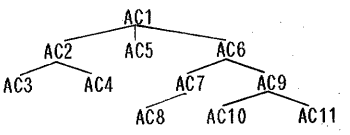

Fig. 2.4 Dependency tree of ICs

2．5 論理データベースにおける知識同化の管理

制約意味モデル (Constraint based Semantic Model: CSM) によりオブジェクトの意味やオブジェクト間の関連(Relationship)を定義された論理データベースにおいては、入力された知識は勿論、その関連する知識も利用者の意図

に従って自動的にDBに同化することができる。本節では、論理データベースにおける知識同化の過程を説明する。知識同化の過程は基本的には次の３通りである。

１）DBに追加すると必ず矛盾を起こす新知識を検出して同化の対象から除く。

２）新知識が追加された複合世界内が無矛盾になるように複合世界内の知識を調整する。

３）新知識が追加されたDB（複合世界の集合体）が無矛盾になるようにそのDBを調整する。

さらに詳細化すると、基本的には次の10通りがあり（ここで、ｎ）は nx)に詳細化される）、その組合された過程も可能である(Fig.2.5参照）。

1a）新知識がDBに矛盾することが検出される(EC による検査）ために新知識はDBに追加されない（Ia）。

2a）新知識がDBに同化され、DB内にそれ以外の変化が起きない（Ib）。

2b）新知識がDBに同化されて、同一複合世界内に推移的な変化が起きる（Ic）。

2c）新知識がDBに同化されて、同一複合世界内のあるクラスの全ての要素が変化する(Id)。

2d）新知識がDBに同化された後、同一複合世界内で指定された時刻に他の変化が起る。

2e）新知識がDBに同化された後、同一複合世界内で2b～2dの複合した変化が起る。

3b）～3e）新知識がDBに同化されて、同一DB内に2b～2eに対応する推移的，クラス的，時間的，複合的な変化が起きる（Ie）。

Fig.2.5 のDatabase1 外のノードは利用者の知識の同化要求を表わし、利用者が意識できる変化である。それに対して、Database1 内における各ノードは、１つの変化またはアクションに対応し、ノード間の矢印は変化・アクションの推移関係を表わす。ノード間の矢印で表現できる推移においては必ず1)の検査が行われる。ここで、利用者は知識の同化によるDB内の変化について必要に応じて確認すればよい。これにより、常識を形成する知識もDB内に自然に追加されていく。

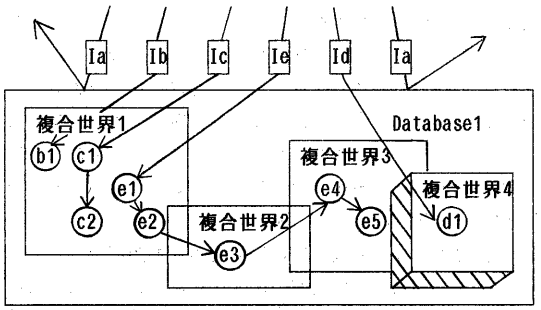

Fig. 2.5 Assimilation management of logic databases

２．６ 制約意味モデル・データベース・システムの特徴

CSM データベース・システムは、人間の知的活動や知識管理を支援する為に利用者の目的・意図を反映する機能を有している。本節では、知識獲得機能および論理DBシステムの設計とライフサイクルの管理機能について議論する。また、この機能の実現のためには、利用者が興味の対象物に対してその意図を示すことを少量の宣言的な叙述で容易に行うことができる必要があるが、CSM データベース・システムでは、利用者はConstraint(OC)を用いて個々の対象物ごとに意味を記述することによりこれを行える。

（１）知識獲得機能

CSM データベース・システムでは、利用者は、知識の意味を利用用途に対応させてConstraint(OC)を用いて記述するだけでよい。知識の獲得は、DBシステムが利用者の意図に従ってあたかも自動的に行う。(Fig. 2.6 参照) その基本は、知識の追加のみならず除去も同じConstraintの枠組みの中で管理できることにある。DBシステムに獲得される知識の中には、人間が無意識の内に学習する知識やその他の雑事に妨害されて学習し忘れる知識も含まれる。これらの知識の中には、常識を形成するために利用される知識もある。このような学習が可能になる理由は、１つの知識の同化により励起される学習が複数のConstraint(OC)により次々に規定されていることによる。

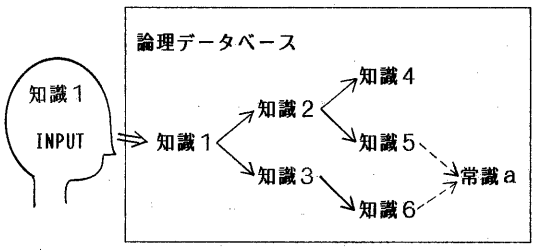

Fig.2.6 Semi-automatic assimilation of knowledge and common sense

（２） データベースの設計とライフサイクルの管理機能

DBの設計においてDB管理者のみならず利用者がDBシステム(DBS) 設計・DBS 管理・応用プログラミング作成を同時に行える機能が、知的活動の支援システムには有効である。利用者は論理DB内のオブジェクトの意味をConstraint(OC)を用いて記述すればよい。(Fig. 2.7参照)　また、ライフサイクルの管理においては、この利用者の意図やデータベースの内容の記述を見ることができることにより DBS再設計・管理の評価が容易になる。

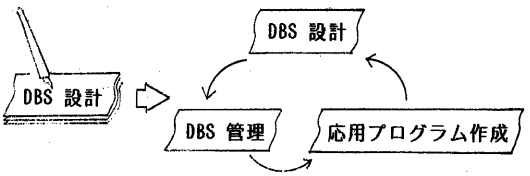

Fig.2.7 Unifying DB design, management and making application programs

3. 制約意味モデルにおける知識同化の実現

制約意味モデル(CSM) によりオブジェクトの意味やオブジェクト間の意味・関連(Relationship)を定義された論理DBにおいては、利用者の意図に従って、入力知識に関連する知識を自動的にDBに同化することができる。 本章では、論理データベースにおいて CSMを、論理型プログラミング言語：Prologにより実現する方法について述べる。

3．1 意味表現のシンタックス

論理データベースにおいて、知識は、ファクト(extension),ルール(intension) そしてConstraints(OC(EC,AC))により表現する (2.2 節参照) 。 Constraints はオブジェクトの意味やオブジェクト間の関連を表現するメタ知識であるので、オブジェクトである他の２つとは区別し、異なるシンタックスで表現する。

OCの表現において、以下の３点が重要である。

ａ） 各オブジェクト毎に別々に表現できること。
ｂ） 宣言的な表現形式が利用できる。
ｃ） 手続き的な解釈ができること。
ｄ） 表現のための記述量が少ないこと。

その理由は以下の通りである。

a)により、OCの追加，変更，削除が容易になる。
b)により、OCの読解が非常に容易になるとともに叙述時の誤りを容易に検出できる。
c)により、OCの実行時の動作の解釈が容易にできる。
d)により、OCの記述の繁雑さを減少させる。

（Ａ）存在制約(EC)のシンタックス

ECは" check_EC "のワクにおいて、対象複合世界，対象オブジェクト，矛盾検出時のメッセージとともに表記することにより定義する。

check_EC(複合世界，オブジェクト，EC，' 矛盾検出時のメッセージ')
check ECにおいてECは次のシンタックス[M84] で記述する。

```
<ECs> ::= <EC>, <ECs> |<EC>;<ECs> |<EC>
<EC>::= <Ls>  --> <L>
<Ls>::= <L> , <Ls>|<L> ;<Ls>| not(<Ls>)|<L>
<L> ::= not(<L>)|<I>
<I> ::= <Prologのゴール>
```

（Ｂ） 動作制約(AC)のシンタックス

動作制約(AC)は次のように"check AC"のワクの中に記述する。 AC内の詳細な叙述は、Prologのgoal列で行う。

```
check_AC( AC_ID,
    Input,
    [[Necessary_Actions], [Banned_Actions]],
    [[World_Name, Class_Constraint_Attributes,
      Pre_State_Descriptions,   Pre_State ->>
        Post_State,  Post_State_Descriptions] |ACR],
    [[Following_World_Name, Following_Actions]|FAR],
    [[World_Name' , Post_Conditions]|WPR],
    Importance)
```

check AC中の最初の項はACの識別子(AC ID),２番目の項が入力された新知識である。 尚、知識の更新を要求するためには入力の形式を"update('Old_Tuple','New_Tuple')" として旧リレーションと新リレーションとを記述すればよい。 知識の異化(dissimilation) を要求したいときには"remove('Tuple')"の形式で記述すればよい。

３番目の項は、既に起きた変化・アクションに関する必要条件のListと禁止条件のListであり、対象としているアクションが起きるまでのシーンの推移過程をConstraint名(Constraint への入力) 列により条件づける。

４番目の項はアクションの本体を記述するListであり、そのなかには a) 単位世界名List, b)クラス制約の対象になる属性(List), c)アクション前の環境の記述(List), d)アクション前におけるアクションの対象の状態記述(List), e)アクション後におけるアクションの対象の状態記述(List), f)アクション後に満足されなければならない条件の詳細(List)を表現できる。 ここでは、アクションの起きるシーンがcheck ACの３番目の項，４番目の項のa),c)により規定できる。 CICの記述を可能にしているのは４番目の項のb)であり、b)に記述された属性はクラス制約の対象になる。 具体的なアクション・変化は上述のd),e)により規定でき、適当なアクション・変化に必要な条件の詳細をf)で与えることができる。

５番目の項は、引続き連鎖的に起るべき知識の同化オブジェクトをConstraint名列により示す(List)。これにより、アクションの連鎖発生を記述でき、シーンの推移も表現で

きる。

６番目の項は、最終的に満足されなければならないDB内の状態を記述する。これにより、適当なアクション・変化が完了したか否かの判定を与えることができる。

７番目の項は、重要な新知識か否かを示すためにある。すなわち、単なる常識の追加ではなく、重要な新知識が同化された場合にその由来を残し、意味を後で明確に把握することができる。由来は、" sys_memory(ID,由来) " の形式で記憶する。

３．２　知識の同化のPROLOGプログラムと実行例

本節で示す知識同化プログラムの前提条件は次の３つである。 1)入力は外延(facts) である， 2)DBはClark の条件(Negation as Failureの十分条件) を仮定する， 3) Consistency は 2.4節の意味である。

知識同化プログラムは、Prologが宣言的表現が可能であるので、ECやACのシンタックスに対応づけながらPrologを使って容易にインプリメントできる。

知識同化の無矛盾性検査において必要な検査機能を、EC, ACの適用例とともに以下に説明する。

（Ａ） EC適用例 … 知識ベースに矛盾する新知識は同化しないという機能が必要。 『Ｈ病院で、山田則夫さん・由美子さん夫妻に子供（陽子さん）が誕生しました。病院側では、陽子さんを２人の子供として登録します。このとき、遺伝学的に正しいことの検査も行っています。』

実行結果では、遺伝学的に誤っていることが検出されている（「（遺伝学）のメンデル博士が“Ｎｏ”と言っている。」という矛盾検出時のメッセージが出力されている）。これは、血液型がＡ型とＯ型の夫婦からＢ型の子供は生まれないことが検出されているのである（Fig. 3.1参照）。

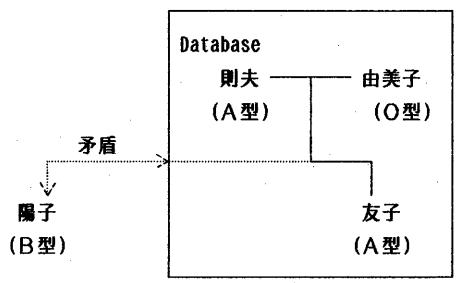

Fig. 3.1　存在制約による矛盾性の検査

－実行結果－

```
?-assimilate([parent], father(youko,norio)).
---Input conflicts with the Integrity Constraint !!
   Dr. Gregor Johann Mendel says " NO ! "
```

ここで、矛盾性を検出したECは次の形式をとる。

－ECの形式－

```
check_EC( [parent], father(X,F),
  (blood_type(F,FT),married(F,M),blood_type(M,MT),
    genes_match(FT,MT,CBT),blood_type(X,BT)
    --> member(BT,CBT)),
  'Dr. Gregor Johann Mendel says " NO ! "', ).
```

ここでは、父親の血液型，母親，母親の血液型，子供の血液型，両親に対する子供の血液型の候補を導出して、子供の血液型がその候補の中に入っていることを関係“father”に対するECとしている。

（Ｂ） AC適用例

従業員管理データベース中に、リレーション：employee (E#,ENAME,Rank,SAL,DEPT),aveSAL(Rank,AVERAGE Salary), rate(DEPT,Rate) が存在し、山田さんはランク(A) の従業員である”という知識も存在する。その他に、『ある従業員のランクがランク(A) から管理者(MC)に昇進する場合には、昇進に伴い給与がSMC:SMC=MCの平均給与×部門別レートになり、ランクと給与のアップとに伴って権利が拡大する』というACが規定されている。 ここで、”山田さんが担当者(A) から管理者(MC)に昇進する”という新知識をデータベースに入れると、山田さんの地位が管理者(MC)に更新されるだけでなく、山田さんの給与が SMC円に更新され、所属部署から権利MC（RoomX1への入室許可権を含む）も与えられる。

ここでは、山田さんの地位，権利が上がる場合に、対応するWorld 内においてConsistency が保持されるために不可欠なアクションを、管理システムが自動的に起こしている。 利用者は、更新結果を確認するだけでよい。

－実行結果－

```
?-assimilate([employees],rank_up(Rank,emp(EN,n_yamada,Rank,SAL,DEPT),mc)).
-- New Knowledge is emp(4,n yamada,mc,1176,researcher)
-- New Knowledge is authority(mc,4,n_yamada,researcher)
```

-AC の形式 -

```
check_AC(4,
```

```
  rank_up(RANK, emp(ENO,ENAME,a,SAL,DEPT),mc),
  [],
  [[[employees],[],[dept(DN,DEPT,DeptRate),
                    aveSAL(mc,S_MC)],
    [emp(EN,Name,a,SALpre,DEPT)]->>
    [emp(EN,Name,mc,SALpos,DEPT)],
    [(SALpos is S_MC * DeptRate / 100)] ]],
  [[[employees],[authority_check(EN,Ename,mc)]]],
  [], 1)
check_AC(5,  authority_check(EN,Ename,mc),
  [[[[employees],rank_up(_,_,mc)]],[]],
  [[[employees], [],
    [emp(EN,Ename,mc,SAL,researcher),   check_
      authority1(EN,Ename,mc,SAL,researcher)],
    [] ->>
    [authority(mc,EN,Ename,SAL,researcher)], [] ]],
  [], [], 1)
```

4. まとめ

利用者の目的・意図を反映させながらデータや知識を同化・管理できる論理データベース・システムの実現を目指して、制約意味モデル(CSM) ・論理DBシステムを提案した。CSM 論理DBシステムでは、利用者は、知識ベースの意味を宣言的に個々のオブジェクトについて利用されるシーンとともに叙述しておくことができ、後のデータや知識の同化・管理は論理DBシステムが行ってくれる。CSM の表現を用いれば、DBの設計・管理・応用プログラムの作成をDBの設計だけでほとんど行えるため、新たな目的・意味が発生すれば容易に追加・修正可能である。 また、DSM によると、複数個のリレーション，属性，インスタンス間の意味や関連性を自由に表現できる。 知識の同化・管理の機能は、OC(EC,AC) 間にトリー状をなして存在する従属性をdepth first に辿りながら個々のICの処理を片付けていく方法を採る。これにより、知識の同化・管理機能は同化を拒否すべき新知識を検出したり、追加した新知識に関連するConsisitencyを保持するための処理を行える。

今後の課題は次のとおりである。

- ICs の記述力を向上させること。
- 利用性を高めること。

－謝　辞－

本研究の機会を与えてくださった（財）新世代コンピュータ技術開発機構　研究所　渕一博所長ならびに熱心に討論して戴いた同研究所　近藤浩康研究員と第２研究室の皆様に深謝致します。

［参考文献］

［BK82］K.A.Bowen and R.A.Kowalski; “Amalgamating Language and Metalanguage in Logic Programming,” Logic Programming (K.L.Clark and S.-A.Taernlund eds.), Academic Press, pp.153-172, 1982.

［BP80］J.Barwise and J.Perry; “Situation and Attitudes,” MIT Press,1983.

［Ca76］J.M.Cadiou; “On Semantic Issues in the Relational Model of Data, ” Math. Found. Comput. Sei. Mazmkiewiez. Vol.45, Berlin H eidelberg New York: Springer, 1976.

［Co70］E.F.Codd; “A Relational Model of Data for Large Shared Data Banks, ” Comm. ACM 13,6, pp.377-387, Jun. 1970.

［Ch76］P.P.Chen; “The Entity-Relationship Model － Toward a Unified View of Data, ” ACM TODS, Vol1, No.1, Mar. 1976.

［Ki84］H.Kitakami, S.Kunifuji,T.Miyachi and K.Furukawa; “A Methodology of Knowledge Acquisition System,” Proceedings of 1984 International Symposium on Logic Programming,Atlantic City, pp.131-142, Feb. 1984.

［HM81］M.Hammer and D.McLeod; “Databsase Description with SDM: A Semsantic Database Model, ” ACM TODS, Vol6, No.3, Sep. 1981.

［M84 ］T.Miyachi,S.Kunifuji,H.Kitakami,A.Takeuchi and H.Yokota; “A Knowledge Assimilation Method for Logic Databases,” Proceedings of 1984 International Symposium on Logic Programming,Atlantic City, pp.118-125, Feb. 6-9, 1984.

［NG78］J.M.Nicolas and H.Gallaire; "Data Bases: Theory vs. Interpretation,"Logic and Data Bases(H.Gallaire and J.Minker, Eds.), Plenum Press, New York London,pp.34 -54, 1978.

［R78 ］R.Reiter; “On Closed World Databases,” in Logic and Data Bases(H.Gallaire and J.Minker,Eds.), Plenum Press, New York, London, pp.55-76, 1978.

［S81 ］D.W.Shipman; “The Functional Data Model and the Data Language DAPLEX, ” ACM TODS, Vol6, No.1, Mar. 1981.

［SM84］K.Sakai and T.Miyachi; “Incorporating Naive Negation into Prolog, ” Proceedings of Logic and Conference, Monash Univ. Jan., 1984.

［SS80］G.Sussman and G.Steele; “CONSTRAINT-A Language for Expressing Almost-Hierarchical Descriptions,” Artificial Intelligence 14, pp.1-39, 1980.