知識獲得システムにおける一貫性保持について

69－4

宮地　泰造　　岡部　進　　北上　始　　古川康一
財団法人　新世代コンピュータ技術開発機構

1．はじめに

知識獲得システム（仮に KASと呼ぶ）が人間にとってより良い助手となるためには、KAS は利用者の目的・意図を反映しつつ知識を同化・管理できなければならない。そのためには、利用者および KAS管理者が、その目的・意図をKAS に対して表現できることが不可欠である。本稿では、保全性制約(Integrity Constraints, 以下ICs と略記する) [1] を用いた KASにおける意味表現方法を示し、Prologにより KASにおける知識の同化・管理機能をインプリメントしたので報告する。

2．知識獲得システムの意味表現

実世界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' をも引起こす。このオブジェクトの価値はその変化やアクションが起きる時に極大となる。KAS では、種々のオブジェクトを ICsに従い同化・管理することにより、実世界を部分的に管理することができる。

一般に、KAS の知識ベース(KB)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、KB内に複数個の Worldが存在し、オブジェクトも異なる意味・側面を有す。 World の具体的な性質はオブジェクトの意味やオブジェクト間の関係に依存する。 そこで、World の具体的な規定は、オブジェクトやオブジェクト間の関係，アクションや変化がおきるシーン・条件やその重要度を叙述してやればよい。 すなわち、オブジェクトが満足すべき必要条件を各々のWorld に対して規定してやればよい。

オブジェクトの叙述には、大別して、静的制約(Static Integrity Constraint:略してSIC)、推移的制約(Transitive Integrity Constraint: 略してTIC)と動的制約(Dynamic Integrity Constraint:略してDIC)によるものがある。

静的制約(SIC) は、オブジェクトの必要条件を叙述し、KB中の各World に対して規定される。 このSIC を満足するオブジェクトだけがDBの目的に矛盾しないとみなされ、KBに同化 (assimilate) 、異化(dissimilate) される資格がある。 ここで、SIC を満足するとは" not(SIC)" が対象とされているworld 内で証明されないことを言う。

推移的制約(TIC) はSIC を満足するオブジェクトがあるWorld:W1内に知識の同化される場合にW1内、KB内に矛盾が起きないための制約であり、様々なシーンにおいて必要な調節処理（アクション）を叙述する。 また、TIC は関係のあるシーンのシーケンスも叙述可能であるため、重要なシーンのシーケンスとそのシーン上における調節処理を複合して叙述できる。 ここで、TIC が満足されるとはTICの個々の調節処理がSIC を満足する場合に実行完了されるときを言う。 すなわち、オブジェクトの存在のための制約チェックと消滅のための制約チェックが満足されて調節処理が行われたときである。

動的制約(DIC) はKB内の部分世界全体が変化を起こし、全く新しい部分世界に変身する場合の制約である。例えば、1年ごとに給与や予算が数％アップする例がある。すなわち、ある世界の全員の給与や予算がアップして、全く新しい世界に変身するのである。この様な部分世界全体が全く新しい部分世界に変身する過程においては、一般に、静的制約(SIC) は成立しなくなる。よって、SIC のチェックを一時的に停止する必要があり、この点がDIC とTIC との異なる点である。

3．知識ベースにおける知識同化と無矛盾性

知識同化の処理では、証明可能性，無矛盾性，冗長性，独立性の検査をこの順に行う[2] 。知識同化の過程においては、KB内に矛盾が起きないようにKBを管理することが重要である。 ICs (SIC,TIC,DIC)を用いて設計されたKBにおいては、無矛盾性の定義をより詳細に行うことができる。" KBが無矛盾である" とはKB内に矛盾が検出されないことを言い、 (A)を満足するときを言う。ここで、demo(W,G) は、論理式における" W⊢ G " に対応し、demop(W,G)は Wから G で示された変化の前後状態が証明されることを意味する。

```
not( demo(World,not(SICs))),  demop(World,TICs),
demop(World,DICs))                          ・・・ (A)
```

4．知識獲得システムの知識の同化・管理

オブジェクトの意味やオブジェクト間の関係等を定義されたKB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。 本章では、KBの知識同化における一貫性検査の過程を説明する。 一貫性検査の過程は基本的には次の３つである。

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World 内が無矛盾になるように調整する。

3）新知識が追加されたDB内が無矛盾になるように調整する。

Fig. 1のKB1 外のノードは利用者の知識の同化要求を表し、利用者が意識できる変化である。それに対して、KB1 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。 また、利用者はKB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、常識がKB内に自然に追加されていくことになる。

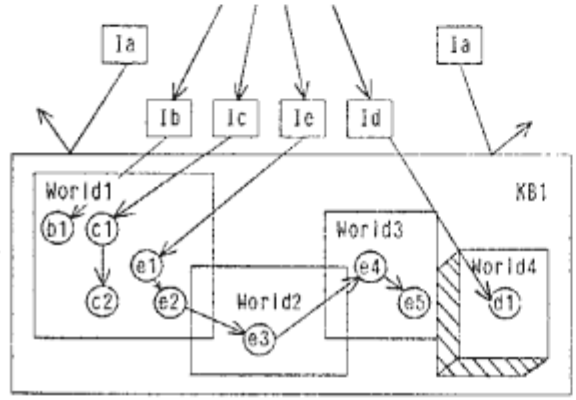

Fig. 1 Assimilation Management of Logic Databases

5. 知識同化のPROLOGプログラム

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 CWA[3] を仮定する。 3)無矛盾性は3章の (A)の意味である。知識同化における TICの検査プログラムの核の部分はFig. 2のとおりである。 PROLOGを使うことにより、 ICs(SIC,TIC,DIC) のシンタックスに対応づけながら宣言的な記述を容易にインプリメントできる。

<実行例>

従業員管理知識ベース中に、リレーション：emp (E#,ENAME,SAL,RANK) が存在し、"山田さんはランク(A) の従業員である" という知識が存在する。この他に知識ベースには、『ランク(A) の従業員の給与がX%アップしたら、他のランク(A) 従業員の給与も全てX%アップしなければならない』という TIC(12)が規定されている。ここで、知識ベースに"山田さんの給与を10%アップする" を入力する。

この場合、1人のランク(A) 従業員の給与の更新が、ランク(A) の従業員全員の給与の更新を引起こし、従業員給与の世界全体を新しく塗りかえている。これにより、従業員給与に関する常識、例えば従業員給与の平均も変ることになる(Fig. 1 のId参照)。

－実行結果－

```
| ?- assimilate([current_kb],raise(SAL,emp(EN,n_yamada,a,SAL,DEPT),10),[A]).

--- New Knowledge is  emp(8,i_tanaka,a,605,development)
--- New Knowledge is  emp(4,n_yamada,a,770,investigation)
--- New Knowledge is  emp(6,s_suzuki,a,660,development)

--- TIC is current_kb(35,[],38,check_tic(12,raise(_38,emp(_66,n_yamada,a,_38,_10
6),10),[[],[rank_up(a,emp(_564,_565,a,_566,_106),_567)]],[[current_kb,[_545],[],
[emp(_564,_565,a,_566,_106)]->>[emp(_564,_565,a,_569,_106)],[_569 is _566*(100+1
0)/100]]],[],[[current_kb,[true]]],1,(_143)))

--- Transitive Integrity Constraints are checked !!
```

ここで使われている TICはつぎに示すとおりである。

－TICの記述－

```
check_tic(1,
          raise(SAL,employee(ENO,[NAME, a,SAL),RATE),
          [[],[]],
          [[Worldname,  [EN],  [],
           [employee(EN , NAME,SALpre, a)]
           ->> [employee(EN, NAME, SALpos, a)],
           [(SALpos is SALpre*(100+RATE)/100)]],
          [],   [],
          1,  employee))
```

```
assim_TIC(Wname,Input,View,Solve_pass,RES_IC):-
     proof_rec(Wname,[check_tic(TIC_ID,
                       Input,
                       Log_check_list,  % Scene sequence check of IC %
                       [[Wname, Universal_var, Pre_conds,
                             Pre_acts ->> Pos_acts, Pos_act_conds]|ICR],
                       Dependency,
                       Pos_conds,  % IC check for side efect %
                       Importance, View) ],
               RES_IC),
        demo_log_check(Solve_pass,Log_check_list),
        demo_pre(Wname,Pre_conds,V,OriginST,View),
       (Universal_var = [],
         demo_pre_act(Wname,Pre_acts,V,OriginST,View),
         demo_pos(Wname,Pos_act_conds,V,OriginST,View),
         demo_pos_act(Wname,Pos_acts,V,OriginST,View) ;
        ((u_demo_pre_act(Wname,Pre_acts,V,OriginST,View),
          u_demo_pos(Wname,Pos_act_conds,V,OriginST,View),
          u_demo_pos_act(Wname,Pos_acts,V,OriginST,View),fail);true)),
        demo_dpnd_check(Wname,Dependency,V, OriginST,View,Solve_pass),
        demo_pos_conds(Pos_conds,V,OriginST,View),
        demo_importance(Wname,Importance,RES_IC), ttynl,ttynl,
        display('--- Transitive Integrity Constraints are checked !!'),
        ttynl,ttynl,ttynl.
assim_TIC(Wname,update(A,NewA),View,Solve_pass,RES_IC):-
       (delete(Wname,A), add(Wname,NewA),ttynl,
        display('----- The Update is done done !!'),ttynl
;ttynl, display('----- The Update is fail, because there is no object !!')).
assim_TIC(Wname,remove(A),View,Solve_pass,RES_IC):-
       (delete(Wname,A) ;ttynl,display('----- The removal is failed !!')).
assim_TIC(Wname,Input,View,Solve_pass,RES_IC):-
        add(Wname,Input),
        display('----- A New Knowledge is assimilated !!'),ttynl.
```

Fig.2 知識同化における TICの検査プログラム

6. ま と め

知識利用者が目的・意図を知識獲得システムに対して表現し、KAS はその目的・意図に従って知識の同化・管理を行う為の方法として、ICs を用いる方法を示した。今後の課題として、次の2つがある。

- TIC の表現力を向上させる。
- 利用性を高める。

［参考文献］

[1] J.Nicolas, H.Gallaire; "Data Base : Theory vs. Interpretation," in Logic and Data Bases, Plenum Press, 1978.

[2] T. Miyachi et al., "A Knowledge Assimilation Method for Logic Databases," in Proceedings of 1984 International Symposium on Logic Programming.

[3] R. Reiter; "On Closed World Databases," in Logic and Databases, 1978.

データフロー方式ＰＲＯＬＯＧマシンのコンパイラについて

益田　嘉直　　伊藤　徳義　　清水　肇
（財団法人　新世代コンピュータ技術開発機構）

１．はじめに

PROLOGは推論の基本機能を内蔵しており、PROLOGの実行過程と並列性を自然な形で実現できるデータフローの概念は相互に密接な関係を持っている[1]。筆者らはPROLOGをデータフロー方式で実行する並列推論マシンの検討ならびにシミュレーションによる評価を進めているが、本稿ではPROLOGで記述されたプログラムを、ストリームを用いたＯＲ並列／ＡＮＤパイプラインを特徴とする本マシンのデータフローグラフに変換するコンパイラについて述べる。

２．処理方式

本コンパイラは、PROLOGのプログラミング言語としての妥当性を検討することをも含め、PROLOGによるプロダクション・システム（PS）の実現により開発する。プロダクション・システムによる開発の利点には

（１）Modularity　：作業効率の良さ、拡張性
（２）Readability：意味理解の容易性

があげられる[2]。本コンパイラの全体の構成を図１．に示す。図からも分かる通りPSは階層的に使われている。上位レベルのルールの集合がプログラムの大前の制御に使われ下位レベルのルールの集合は個々の部分問題の解決のために使われる。

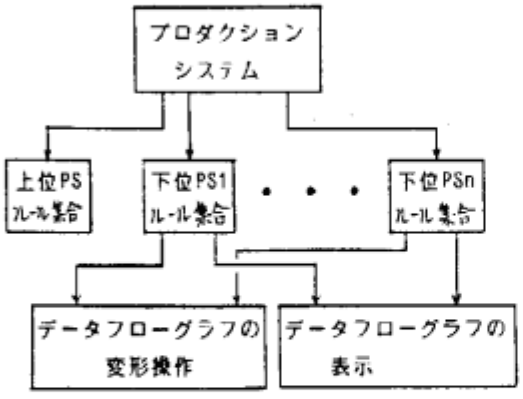

図１．コンパイラの全体構成

３．言語

PROLOGの言語仕様としては、現在検討が進められている核言語第一版（KL1)への拡張を考慮し当面はPure PROLOGに各種の制御オペレータが付加された言語仕様を考える。

中間言語は、本マシンのデータフローグラフの記述に向いた言語仕様を考える。中間言語プログラムの各命令はデータフローグラフの各ノードに対応し、各命令に付加されている当該命令の実行結果の送り先は同じくデータフローグラフの各アークに対応する。中間言語は専用のアセンブラにより本マシンの機械語に変換される。中間言語によるappendの記述例を図２．に、そのデータフローグラフ表現を図３．に示す[3]。

```
r<=append(x,y,z)
begin
        xx=copy(x)
        yy=copy(y)
        zz=copy(z)
        (res,stream=create_stream(xx))
        ret(res,r)
        a1=unify_with_nil(xx)
        a2<=unify_and_share(yy,zz)
        b1=cons(a2,nil)
        b2=cons(a2,b1)
        b3=cons(a1,b2)
        append_stream(b3,stream)
        (h1,t=decompose(xx))
        (h2,u=decompose(zz))
        head<=unify_and_share(h1,h2)
        s1=true(stream,head)
        h3=true(head,head)
        t1=true(t,head)
        y1=true(yy,head)
        u1=true(u,head)
        r1<=append(t1,y1,u1)
        rr=rest(r1)
        <=gen_append(s1,h3,rr):rr
end
```

図２．中間言語の記述例

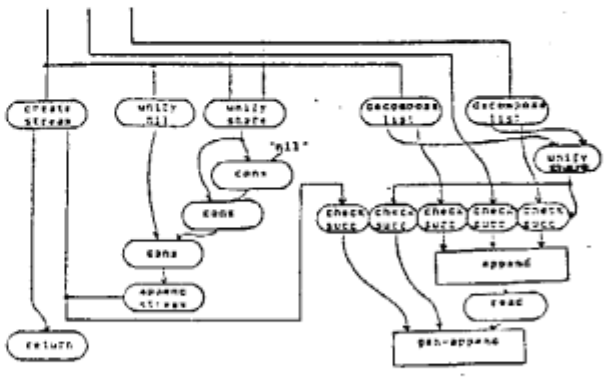

図３．appendのデータフローグラフ例

更に、本マシンの中間言語の記述には、PROLOGの非決定的制御機構がcreate_stream 及びappend_stream の２種類のオペレータの導入により実現されている特徴を見ることができる[4]。

4．プロダクション・システムの実現

本プロダクション・システムの基本部分を図４．に示す。本プロダクション・システムにおける推論は、recognize-act サイクルの単純な繰り返しにより実行される。recognize部では複雑なconflict resolution の機能はなく、最初に見出された適用可能なルールを起動する方法をとっている[5]。(first-match)

```
produce(X,Y,Z,U):- produce1(X,Y,Z,U,[]).

produce1(Clist,STM,Rslt,[Rname|Plan],Hist)
   :-recognize(Clist,STM,Rname,Action),
     act(Action,STM,New_STM),
     produce1(Clist,New_STM,Rslt,Plan,[Rname|Hist]).

produce1(Clist,Rslt,Rslt,[],Hist).

/* If there are no rules which can apply,
   then the STM at that time is the Result. */

recognize(Clist,STM,Rname,Action)
   :-prod_rule(Class,Rname : Cond => Action),
     member(Class,Clist),
     hold(Cond,STM).
```

図４．プロダクション・システムの基本部分

```
append([],Y,Y).

    a(ha(1),[])
    a(ha(2),va(1))
    a(ha(3),va(1))
    max(ha,4)
    va(1,[ha(2),ha(3)])

append([U|X],Y,[U|Z]):-append(X,Y,Z).

    a(ha(1),lst(1))
    a(ha(2),va(3))
    a(ha(3),lst(2))
    a(ba(1,1),va(2))
    a(ba(1,2),va(3))
    a(ba(1,3),va(4))
    max(ha,4)
    max(ba(1),4)
    va(1,[ca(1),ca(2)])
    va(2,[cd(1),[ba(1,1)]])
    va(3,[ha(2),[ba(1,2)]])
    va(4,[cd(2),[ba(1,3)]])
    lst(1,va(1),va(2))
    lst(2,va(1),va(4))
```

図５．appendの引数表現の例

5．引数の表現とルールの記述

引数の表現はシステムを効率良く実行する上で重要な意味を持つ。本システムにおいては、head predicate,body predicate の引数をそれぞれha(N),ba(M,N) で表わし、clause毎の引数処理により相互に結合された引数表現が中間結果としてジェネレートされる。構造データ引数についてはlst(N,-,-)で表わし、共有変数の場合には同一の変数番号を割当てる。appendの引数表現の例を図５．に示す。

次に、プロダクション・ルールの例を示す。ルールは一般に、

```
prod_rule(Class,Name:Cond->Action)
```

の表現形式を取っており、引数の表現によりデータフローグラフの変形操作が行われる。head predicateの引数がnil の場合のデータフローグラフの変形を行うプロダクション・ルールの例を図６．に示す。

```
prod_rule(u_head,arg003:
          [a(ha(N),[]),N1 is N+1,
           absent(max(ha,N1)),
           dfg(DFG)]
        =>
          [append([c(N)=cons(un(ha,N),c(N1)),
           un(ha,N)=unify_with_nil(ha(N))],DFG,N_dfg),
           delete(a(ha(N),[])),
           replace(dfg(DFG),dfg(N_dfg))]).
```

図６．プロダクション・ルールの例

6．おわりに

本稿では現在検討を進めているデータフロー方式によるPROLOGマシンのコンパイラについて述べた。今後は核言語第１版(KL1) への拡張の検討、ならびに中間言語の記述能力の検証を進め、その過程で不十分な点を改良して行くと共にコンパイラの評価をも行う予定である。

最後に、日頃ご指導をいただく第一研究室村上室長はじめ並列推論マシングループ諸氏に感謝の意を表する。

＜参考文献＞

[1] 伊藤，尾内，益田，清水，“データフロー機構に基づくPROLOGマシン”，情報処理第２６回全国大会．

[2] 佐藤泰介，“プロダクション・システムの試作とその使用経験”，情処人工知能と対話技法研資 3-1(1978)

[3] N.Ito,K.Masuda,H.Shimizu, “Parallel Prolog Machine Based on Data Flow Model”,ICOT Technical Report TR-035(1983.8).

[4] 伊藤，益田，清水，“データフロー方式PROLOGマシンにおける非決定的制御機構”，情報処理第２７回全国大会．

[5] 古川，“PROLOGによる問題解決”，情報処理第２３回全国大会．