研　究　速　報

UDC 007.3：681.3.02

知識依存型システム作成のためのプロダクション・システム
——SPROS——

Production System for Building Knowledge-Based Systems

坪井邦明*・沼尾雅之**・石塚　満*
Kuniharu TSUBOI, Masayuki NUMAO and Mitsuru ISHIZUKA

1. まえがき

人工知能研究を基礎とする知識工学[1]は，新時代の情報システム設計の方法論として大きな期待が寄せられている．知識工学の一手法として広く使用されているのが，プロダクション・システムである．プロダクション・システムは，IF A THEN B というルールの形式で表される知識の集合（これをルール・ベース，あるいは知識ベースと呼ぶ），関連するデータを蓄える構造体（データ・ベース），およびデータ・ベースに基づいてルール・ベース中のルールを選択・適用して推論を進めていく推論機構からなる．認知心理学の面からは，ルール・ベースは人間の長期記憶に，データ・ベースは短期記憶あるいは感覚器に，推論機構は思考過程に相当する．このようなシステムは，次のような特徴を持つ．

（1） ルール（知識）の追加・変更・削除が容易
（2） ルールが一片の完結した知識であるので，意味の理解が容易
（3） ルールから，システムの動作の理解が容易

プロダクション・システムは，第五世代コンピュータの核言語として注目されている1階述語論理に基づく言語 PROLOG や，フレーム理論に基づくシステムに比べ，上記のような特徴のゆえに，知識依存型システム設計の，一つの有効な手法である．

我々はすでに Frantz-LISP 上に汎用プロダクション・システムを作成し，画像解析用[2]，その他の目的[3]に利用して成果を得ている．今回は，これを原型とし，高速性，制御の容易さ，及び開発・デバッグ機構の拡充を中心に改良して，UtiLisp[4]上に移植した．このプロダクション・システム（SPROS と名付ける）の概要を示す．(UtiLisp は生研の計算機室でも稼働しており，したがって SPROS の使用も可能である．)

2. SPROS の概要

2.1 推論機構

図1は SPROS の推論機構のトップ・レベルを示している．推論は次のように進む．

（1）データ・ベースの内容に適合するプロダクション・ルールを選び，実行する．変数に対する複数のデータの組み合わせが可能な場合は，そのうちの一つの組み合わせを選ぶ．データの組み合わせを選ぶ際，かつて使用されたことのある組み合わせは避ける．

（2）適合するルールがない場合は，1ステップ前の状態に戻り（バックトラック：注1），別のデータの組み合わせがあればそのうち一つを選んで再実行し，そうでなければ，別の適合するルールを探す．

（3）一つのルールが実行されると，禁止状態の検査をし，禁止状態が発見される（適合するスーパー・ルールがある：注2）と，バックトラックを引き起こす．

2.2 データ・ベースの構造

データ・ベースの中の一つの値は
(〈context〉〈attribute〉〈value〉)
という三つ組の形式で表され，データ・ベースは次のような構造をしている．

〈data-base〉 ::＝(〈data-set〉の集合)
〈data-set〉 ::＝(〈context〉(〈data-pair〉の集合))
〈context〉 ::＝シンボル
〈data-pair〉 ::＝(〈attribute〉〈value〉)
〈attribute〉 ::＝シンボル｜値
〈value〉 ::＝シンボル｜値

たとえば，旅行計画に関するデータ・ベースの一部は，次のようになる（ただし日本文字は使えない）．

((沿線駅　(山田線　盛岡)　(山田線　宮古)
　　(東北新幹線　盛岡)　(東北新幹線　大宮)…)

注1：バックトラックのために，推論の各ステップの状態を記憶する HISTORY-LIST を備えている．適合したデータの組み合わせ，使用されたデータの組み合わせは，ルール自身が記憶する．

注2：スーパー・ルールは，条件部のみで帰結部を持たない（帰結はシステムのバックトラックのみの）ルールである．

* 東京大学生産技術研究所　第3部
** 日本アイ・ビー・エム株式会社

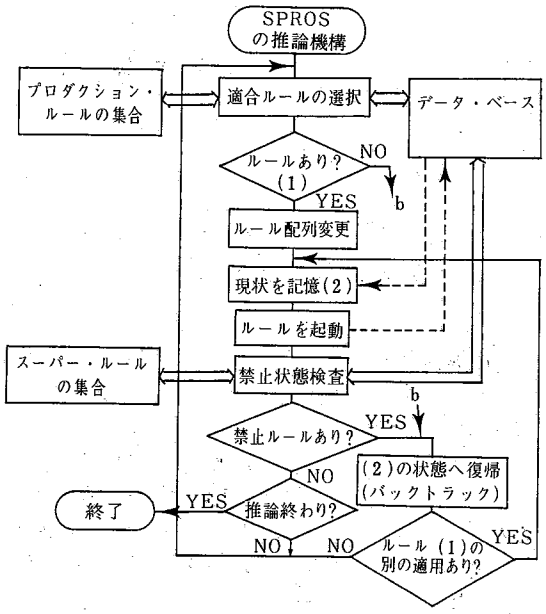

図1　SPROSの推論機構

(乗換線　(盛岡　東北新幹線)　(盛岡　山田線)
　　(大宮　東北新幹線)　(大宮　京浜東北線)…)
(旅行　(目的地　宮古)
　　(現在地　六本木)))

2.3 プロダクション・ルール

プロダクション・ルールは次のような形式とする．ただし，スーパー・ルールは→以下を持たせない．

```
(rule-name
  (system-if-function)
        ・
  (user-if-function)
        ・
        ・        )
  →
  (system-then-fusction)
        ・
  (user-then-function)
        ・
        ・        )
```

条件部と帰結部は→によって区切られ，どちらも上から順に適用される．条件部では〈attribute〉と〈value〉には変数（＊〈変数名〉で示す）が使用でき，データ・ベースの内容によって値が代入され，あるいはすでに代入された値が棄却されたりする．変数のスコープは，一つのルールである．

system-if-functionには，次のものが用意されている．

(1)(IS 〈context〉〈attribute〉〈value〉)

データ・ベース中に(〈context〉〈attribute〉〈value〉)であるデータがあること．

(2)(NO 〈context〉〈attribute〉〈value〉)

データ・ベース中に(〈context〉〈attribute〉〈value〉)であるデータがないこと．

(3)(SET (user-function) 〈variable〉)

条件として直接関与するものではないが，(user-function)の評価結果で，変数〈variable〉を束縛する．(user-function)は任意のLISP関数．

system-then-functionには，次のものが用意されている．

(1)($DELETE$ '(〈context〉〈attribute〉〈value〉))

データ・ベースの中から，このデータを削除する．

(2)(ADD '(〈context〉〈attribute〉〈value〉))

データ・ベースにこのデータを加える．ただし，すでにこのデータが存在する場合は，データ・ベースは変化しない．

(3)($CHANGE$ '(〈context〉〈attribute〉〈value〉))

データ・ベース内の(〈context〉〈attribute〉)が持つ値を〈value〉で置き換える．

(4)($PUSH$ '(〈context〉〈attribute〉〈value〉))

データ・ベース内の(〈context〉〈attribute〉)が持つ値を退避した上で，このデータを付け加える．

(5)(POP '(〈context〉〈attribute〉))

データ・ベース内の(〈context〉〈attribute〉)が持つ値を削除し，退避されていたデータを復帰させる．

(6)(END)

推論の正常な終了を示す．

user-if-functionはLISPの任意の関数であり，評価値がNILでなければ条件は満たされる．user-then-functionは，LISPの任意の関数でありそのまま評価される．

たとえば，旅行計画を立てるプロダクション・ルールの一部は，次のように書ける（データ・ベースの例を参照）．

```
(P10  ($IS$  旅行  目的地  ＊X)
      ($IS$  旅行  現在地  ＊Y)
      ($IS$  沿線駅  ＊L  ＊X)
      ($NO$  沿線駅  ＊L  ＊Y)
      ($IS$  乗換線  ＊S  ＊L)
      →
      ($PUSH$ '(旅行  目的地  ＊S)))
(P20  ($IS$  旅行  目的地  ＊X)
      ($IS$  旅行  ＊Y)
```

(IS　乗換線　*Y　*L)
(IS　沿線駅　*L　*X)
→
(ADD '(交通　*L (*Y　*X)))
($CHANGE$ '(旅行　現在地　*X))
(POP '(旅行　目的地)))

3. SPROSの特徴

SPROSの推論機構は，前バージョンや他のシステムと比較して，次のような特徴がある．これらはおもに，推論効率，制御の容易さのためのもので，システム構築・デバッグ機能については，次節で述べる．

3.1 ルールの競合

適合するルールが同時に複数存在する（ルールが競合する）場合，それらのうち最も有効なルールを選ぶことが望ましいが，一般的な選択戦略の組み込みは困難である．また，そのために毎回全ルールの適合を調べるのは時間的にも問題である．SPORSでは，適合するルールが一つ見つかった段階で照合作業を打ち切り，バックトラックが発生した時に残りのルールの中から適合するルールを探す．ただし，3.4で述べるように，適合を調べる順序は適応的に変化する．

3.2 データの競合

あるルールに対して，変数の束縛の組が複数存在する(データが競合する)場合がある．前バージョンではこれに対して並列動作を採用しており，それはある場合には有効であるが，人間がルールの動作を理解しきれない；制御がしにくい(むしろ不都合な場合もある)，等の問題もあった．SPROSでは競合するデータのうち，一つの組み合わせのみを採用して実行する．

3.3 推論の無意味なループの防止

各ルールは，採用したデータの組み合わせを記憶していて，まったく同じ状態で再び起動されることはない．これによって，推論の無限ループの発生等が回避される．

3.4 ルールの並びの適応的変更

各ルールの条件部・帰結部・上述の既使用データ等は，ルール名アトムの属性リストに格納されており，このルール名を連ねたリストがルールの並びである．照合作業はこのリストの先頭のものから順に行われるが，この単純な構造のリストの配列を変更するのは簡単である．

推論効率を高めるため，現在，次のようなルールの再配列を行っている．

問題を分割してゆくような場合は特に，同一ルールが繰り返し使用されることが多いので，各ステップで適合したルールを並びの先頭に配置する．これは3.2の競合データをすべて適用しなくてはならない場合にも有効である．一方，バックトラックの原因になったルールは，しばらく使わないように，並びの最後に配置する．

なお，その他の経験的な方法でルールの並びを変更することも可能である．

4. 対話・デバッグ機能

知識依存型システムは，人間の思考法に近い推論を重ねて，問題を解決するものであるため，人間にとって馴染み易く，また，高度な処理を実現できる．しかし一方，人間の思考と同様，決定性のアルゴリズムを持たないので，間違った知識（の断片）があると，それを見い出すことはなかなか容易ではない．

SPROSは，知識システムの開発を重要な目的としているので，人間とのインターフェースを重視し，充実したデバッグ機能を備えてこれに対処している．これらの機能はすべてコマンド形式で対話的に行われ，システムの稼働中にも知識の修正や新たな知識の供給等が可能であり，また，システムの説明機能としても有用である．

以下，SPROSおよびデバッガのコマンドの概要を紹介する．なお，コマンドの説明は？を入力すればいつでもえられる．また，デバッグ・モードは，ATTENTION-KEYの押下により起動される．

4.1 SPROSのコマンド（%はプロンプト）

%PRINT：データ・ベース，プロダクション・ルール，スーパー・ルール等の内容を表示する．

%EDIT：データ・ベース，プロダクション・ルール，スーパー・ルール等の内容を構造エディタにより修正・編集する．

%TRON：推論中のデータ・ベース，変数の束縛等の様子を逐次表示する．

%TROFF：%TRONを解除する．

%BTRON：推論中の変数の束縛の様子を逐次表示する．

%BTROFF：%BTRONを解除する．

%GO：推論の開始．

%QUIT：LISPのTOPLEVELへ抜ける．

4.2 推論中のデバッグコマンド（%%はプロンプト）

%%P：注目時点の状態を表示する．はじめはデバッグ・モードに入ったときの状態に注目している．表示する内容は，

1. データ・ベースの世代
2. データ・ベースの内容
3. 適合したルール（これから起動される）
4. 変数の束縛状態
5. まだ調べていないルールのリスト．

%%W(〈context〉〈attribute〉〈value〉)：推論過程の

研　究　速　報

最初にデータ・ベース中にこのデータが表れた時点に注目する.

%%B：注目時点を一つ前に移す.

%%A：注目時点を一つ後に移す.

%%S：注目時点のデータ・ベースの内容を新たにデータ・ベースとして設定する.

%%RULES：データ・ベースに適合するすべてのルールをさがし，ルール名を表示する.

%%MATCH〈ルール名〉：このルールがデータ・ベースに適合するか否かを調べ，適合すれば変数の束縛を表示する.

%%BIND〈番号〉：最後の%%MATCHにおいてえた束縛のうち〈番号〉番目の束縛を採用する.

%%ACT：最後の%%MATCH %%BINDによるルルを実行する. HISTORY-LISTもつくられる.

%%BACK：強制バックトラック.

%%PRINT：前述の%PRINTと同様.

%%EDIT：前述の%EDITと同様.

%%R：通常の推論に復帰.

%%Q：LISPのTOPLEVELに抜ける.

5. むすび

操作性能を向上させて実装したプロダクション・システムSPROSについて報告した. 本システムを用いた知識依存型システムは既に成果を得ている[5].

SPROSは，知識依存型システムの開発に重点を置き，デバッグ機能等に注意を払って作成したものである. 推論速度の向上も図られているが，逐次型のルール照合を行っているので，ルール数が増えると，ルール数に比例して速度が低下する問題がある. これに対しては，弁別ネットを用いるプロダクション・ルールの内部表現を採用して対処するように検討中である. これによって，推論時間をルール数にほぼ関係なく短くできる. また，前述の三つ組のデータ・ベースのみでなく，別に開発しているフレーム型システムを結合して，これを直接扱えるように検討中である. これによって，知的CADシステムのような問題への適用を計画している.

謝　辞　安田教授，多次元画像情報処理センタの尾上教授，高木教授，坂内助教授をはじめ，日頃ご討議いただく方々に感謝します.　(1983年9月9日受理)

参考文献

1) A. Barr & E. A. Feigenbaum eds.: "The Handbook of Artificial Intelligence", William Kaufmann Inc., Vo L. 1, 1981 (邦訳：共立出版), VOL. 2•3, 1982
2) M. Ishizuka. M. Numao. Y. Yasuda: "A Rule-Based Intekpketation of Contour Patterns with Curves", Inter Graphics'83, Tokyo, April, 1983
3) 坪井，沼尾，石塚："ルールに基づく日本民謡の旋律構造解析"，電子通信学会，パターン認識と学習研資 PRL 82-61，1982. 12
4) Chikayama, T.: "UtiLisp Manual", METR 81-6，東大工学部計数工学科，1981. 9
5) 坪井，石塚："知識工学手法による音楽解析の試み…日本民謡の旋律構造解析"，日本音響学会，音楽音響研資，MA 83-8，1983. 9