ICOT Technical Report: TR-645

TR-645

Prologにおけるプロダクション照合フィルタの高速化

新谷　虎松（富士通）

May, 1991

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Prologにおけるプロダクション照合フィルタの高速化

Speeding up the Matching Filter for Prolog-based Production Systems

新谷虎松
(株)富士通研究所 国際情報社会科学研究所
tora@iias.flab.fujitsu.co.jp

ABSTRACT

In this paper, we propose a new matching filter for Prolog-based production systems(PS). The matching filter is used to obtain instantiations without using particular mechanisms through a matching process in PS. We call the new filter "the LHS filter". The LHS filter can be used to improve a matching process in PS and speed up PS. In order to realize the LHS filter, we exploit advantages of Prolog such as an indexing mechanism and a mechanism for logical variable bindings. The indexing mechanism is used to speed up the searching rules and working memory elements in a matching process. The variable bindings mechanism is exploited to realize an efficient join computation in the matching process. By using the LHS filter, we implemented the production system KORE/IE which is fast enough to construct applications on Prolog. The performance of KORE/IE is comparable with that of the OPS5.

1．まえがき

Prologをベースにしたプロダクションシステムにおいて，システムの機能およびルールの表現力を制限することなく推論の高速性を実現することは重要な課題である．特に，推論の高速性は，アプリケーションプログラムを構築する上で強く要望される本質的な要求である．元来，記号処理言語としてのPrologは，同様に記号処理言語であるLispに比べアプリケーションプログラムの実行が非常に遅い言語とされ，Prologによるアプリケーションプログラムの高速性はその機能性に比べそれほど追求されていなかった．また，Prologは，Lispに比べ複雑なデータ構造を利用したりプログラムを制御するための機能がほとんどないために，Prologを用いたプロダクションシステムの研究においても実用的な高速性を実現した成果が得られていなかったのが現状である．

本研究の目的は，Prologに内在する特長（例えば，節のハッシュインデキシング等）を効果的に適用することによりProlog上に十分に高速でありかつ実用的機能性を実現するプロダクションシステムを構築することである．目標とする高速性と機能性はLisp上にインプリメントされている前向き推論型プロダクションシステムであるOPS5[6]に匹敵することである．OPS5は，現在，最も広範に用いられている高速なプロダクションシステムであり，多くの実用的なアプリケーションの記述システムとなっている．

Prologプログラミングにおいてプロダクションシステムは，Prologの基本機能（例えば，ユニフィケーション等）を用いて容易に構築できることは良く知られている．既存のPrologによるプロダクションシステムの代表的な構築手法には，(1)プロダクション・インタプリタを構築する方式[8),1)]，(2)プロダクション（もしくはルール）を素直にPrologの節形式へ変換する方式[17),3)]，および(3)部分計算技法による(1)と(2)の融合方式がある[23)]．(1)はプロダクションシステムを構築するための最も一般的な方式であり，適切なルールの表現能力や推論機能を実現できる利点がある．一方，この手法はPrologでは非常に効率の悪い技法として良く知られており，プロダクションシステムを効果的に高速化することは困難である．(2)は，Prologシステム自身をプロダクション・インタプリタとして利用するものであり，ルールをPrologの節表現に変換する方式である．ルールは条件部（LHS(Left Hand Side)と呼ぶ）がPrologホーン節の本体に，結論部（RHS(Right Hand Side)と呼ぶ）がホーン節のヘッ

ドに変換される．この方式は，Prologの基本的計算メカニズム（つまり反駁メカニズム）を直接に利用するので(1)の方式に比べ高速なシステムを実現できる．しかしながら，ルールを直接的にPrologの節に展開する必要性から，ルールの表現力や推論機能が制限される欠点がある．(3)の方式は(2)の欠点を補うためのものである．この方式は，プロダクション・インタプリタを与えられたルールを用いて推論の前に予め部分計算し，対象とするルールに対して特殊化するものである．推論はこの特殊化されたPrologプログラムの実行として実現され，(1)に比べ高速である．(3)の方式の欠点は，推論の高速性が前もって与えるプロダクション・インタプリタの性能に依存し，部分計算により得られたプログラムには多くの制御情報が含まれ，(2)に比べ推論の効率が悪くなるのが一般的である．

これらPrologにおける(1)～(3)の手法の共通点は，プロダクション・インタプリタの存在を前提としていることである．これら手法に関連したプロダクションシステムの高速化はプロダクション・インタプリタのプログラムそのものが対象になっており本質的な高速化を期待できなかった．一方，OPS5で代表される前向き推論型プロダクションシステムの高速化技法として，RETEアルゴリズム[7]やTREATアルゴリズム[16]が良く知られている．これらアルゴリズムは，プロダクションシステムの推論メカニズムにおける照合過程を効率良く実行するためのものであり，McDermott[15]が提案した照合過程のためのフィルタの機能を実現している．本論文ではMcDermottのフィルタを一般的な視点からプロダクション照合フィルタと呼ぶ．実際のインプリメンテーションでは，一般に，プロダクション照合フィルタはルールコンパイラーを用いて，ルールの条件部からある種のネットワークを構成することにより実現される．プロダクション照合フィルタはワーキングメモリの変化の分に着目し，無駄な照合の繰り返しを極力避けることにより照合過程を高速化する．論理型言語上でRETEアルゴリズムを導入した高機能なプロダクションシステムとしてはPOPS2[11]がある．POPS2では，ネットワークアルゴリズムを効率化できる高機能な論理型言語処理系を使用することによりその高速化を指向している．しかしながら，標準的なProlog（例えば，C-PrologやQuintus Prolog）上でRETEアルゴリズム等のようなネットワークに基づくプロダクション照合フィルタを効率的に実現することは困難である．あえて，ネットワーク構造を実現しようとするならば，Prologにおいて非常に効率の悪いassert述語およびretract述語を多用することになり，システムの高速性がむしろ低下する原因になる．

筆者らは，標準的なPrologの利点を生かした高速な前向き推論型プロダクションシステムKORE/IE20)を試作している．ここでは，ルールの条件部からLHS節と呼ぶ照合過程に特殊化したPrologプログラムを生成することによりその高速化を実現した．本論文では，Prologの利点を最大限利用する視点からLHS節を改良したプロダクション照合フィルタの構成法とその詳細について論じる．主な改良点は，(1)インデキシング機能をさらに効果的に用いる，(2)LHS節のメモリスペースを削減することを主眼とする．本研究で得られたプロダクション照合フィルタはLHSフィルタと呼ばれ，プロダクションシステムにおける照合過程を効果的に高速化する．

2．プロダクション照合フィルタ

プロダクションシステムにおける推論は，認識-行動サイクルを実行することにより達成される．認識-行動サイクルは，(1)照合(Matching),(2)競合解消(Conflict Resolution),(3)動作(Action),より構成され，これら(1)～(3)を繰り返す．(1)の照合は全てのルールのLHSと大域的なデータベースであるワーキングメモリ(Working Memory)（WMと略す）の内容（WM要素と呼ぶ）を照らし合わせて，LHSを満足するルール（インスタンシエーション(instantiation)と呼ぶ）を選び出す．インスタンシエーションはルールとそのルールのLHSと照合が成功したWM要素で構成される．インスタンシエーションの集合は競合集合と呼ばれ，(2)の競合解消時に起動されるべきルール（インスタンシエーション）が競合集合から一つ選択される．選択されたルールは(3)の動作過程で起動される．推論の高速化は，(1)～(3)のサイクルを高速化することにより効果的に達成できる．単純な認識-行動サイクルの繰り返しにおいて，照合過程は認識-行動サイクルの実行時間の大半(約９割)を占めている[10]．プロダクションシステムの高速化のためには照合過程を効率化することが本質的である．サイクルごとに全てのルールの条件要素とWM要素を照合すると，ルール数やWM要素数が多くなれば，照合に多くの時間が費やされることになり，実用的でない．また，照合時に必要とされる多くの変数情報を管理することも多くの時間が必要とされる原因である．

プロダクション照合フィルタは，照合過程を効率化するためのものであり，インスタンシエー

ションを特殊な解釈実行系を介することなく生成するために用いられる．プロダクション照合フィルタの概念は，McDermott，NewellおよびMoore[15]により提案され，(a)Condition Membership，(b)Memory Support，および(c)Condition Relationshipと呼ぶ3種の高速化に寄与する知識を組み合わせて用いることにより構成される．(a)は，どの条件要素がどのルールに含まれるかについての知識である．(b)は，どのWM要素がどの条件要素を満たしているかについての知識である．(c)は，ひとつのルールのなかでの複数の条件要素間でどのような関係があるかについて知識である．フィルタの構成例として，McDermottらは(a)と(b)を組み合わせたCondition-Membership Memory-Supportフィルタを提案している．

RETEアルゴリズム[7]は，先の(b)と(c)の知識を組み合わせたMemory-Support Condition-Relationshipフィルタに相当する[16]．RETEアルゴリズムはあらかじめ全てのルールのLHSをマッチングパターンとしてネットワーク化（データフローネットワークの一種を形成する）する．照合過程は，WMの変化の分をこのネットワーク（RETEネットワークと呼ばれる）にトークンとして流すことにより実現される．トークンの流れは，(c)のCondition Relationshipに関連した機能を提供する．照合時における情報（変数の束縛等）は中間結果としてネットワーク内に記憶され，次以降のステップでの照合過程で利用される．これにより無駄な照合の繰り返しを回避する．記憶場所はαメモリおよびβメモリと呼ばれ，(b)のMemory Supportに関連した機能を実現する．αメモリは，1入力ノードでの定数テストと呼ばれる照合テストに成功したトークンを記憶する．βメモリは，2入力ノードおける条件要素間の変数束縛テスト（joinテスト）に成功したトークンの組を記憶する．複数のβメモリを経由することによりネットワークの終端に達したトークンの組はインスタンシエーションに相当し，競合集合に付加される．RETEネットワークの特徴はネットワークのノードを共有することによりメモリの削減と実行性能の向上を図っている．

TREATアルゴリズムは，先の(a)と(b)の知識および新たに(d)としてConflict-set Supportと呼ぶ高速化に寄与する知識を組み合わせたプロダクション照合フィルタ（Condition-membership Memory-support Conflict-set-supportフィルタ）に相当する[16]．(d)は，変化するWM要素（seedと呼ばれる）に着目して，効率的に競合集合を決定するためのものである．例えば，WM要素の削除は，それを含むインスタンシエーションを競合集合から直接的に取り除く．さらに，TREATアルゴリズムでは，joinテストの際にこの変化分（具体的には，αメモリの変化）を最初に考慮するseed-orderingと呼ばれるヒューリスティックが用いられ，join演算の順序を最適化している．TREATアルゴリズムもRETEアルゴリズムと同様に，αメモリやWMの変化を利用することにより，認識-行動サイクルにおいて無駄な照合の繰り返しを回避する．TREATアルゴリズムの特徴は，βメモリを持たずにサイクルごとにjoinをseed-orderingで動的に計算しなおすことである．これは，αメモリだけを保存することにより，前照合過程の中間結果を保存するためのオーバヘッドを軽減しようとするものである．

TREATアルゴリズムやRETEアルゴリズムにおいて，join演算はデータ量が増大した場合非常に時間が要するものとなり，効率化が必要である．join演算の効率化には，様々なヒューリスティクスが必要とされる．最近では，join演算の最適化に関連して多くの研究が行われている[12),18)]．

RETEアルゴリズムやTREATアルゴリズムを効果的にインプリメントするにはネットワーク構造を効果的に実現するための手段（例えば，ポインタの利用等）が必要である．Prologでは，複雑なデータ構造を扱う機能がないので，ネットワークをベースにしたプロダクション照合フィルタを効率的に構成することは困難である．一方，join演算は，Prologの論理変数の機能をうまく利用することにより，特別な機構を構築することなしに実現できる．第3章ではPrologの利点を利用した新たなプロダクション照合フィルタの構成法を論じる．

3．Prologに基づく照合フィルタ

3.1 LHSフィルタの概略

LHSフィルタは，RETEネットワークと同様にルールのLHSからルール・コンパイラーを用いて生成される．LHSフィルタの概略は図1のように示すことができる．

図1は，図1上で与えられたプロダクションシステムKORE/IEルールのLHSをコンパイルすることにより，図1下で示すLHSフィルタが生成されることを示している．LHSフィルタは，LHS節と呼ばれるPrologのホーン節を用いて実現される．大文字で始まるアトムはProlog論理変数を表す．図1のルール記述において，r1はルール名を表す．LHSにおける条件要素は，スロット記述を引数とするPrologの項で表現される．スロット記述は属性を表すスロットおよびその値の対である"スロット＝値"で構

成される．例えば，ルールの第１条件要素を表すパターンにおいて，"name=move"や"status=active"はスロット記述である．ここで，スロット"name"，"status"の値は，それぞれ"move"，"active"である．

図２は，図１におけるLHSフィルタの第１番目のLHS節の構成を説明している．LHS節のヘッドの引数は，スロット値の並び，タイムタグ，およびインスタンシエーションから構成される．これら，引数の並び順は，効果的にPrologのインデキシングを利用する上で重要である(3.2節参照)（旧LHS節[20]では，厳密にこの並び順が最適化されていなかった）．LHSにおける変数は，LHS節ではPrologの論理変数として表現される．スロット値の並びの順は，スロットと対応させるために予め定義される（プロダクションシステムKORE/IEではスロット値の順を"literalize"コマンドを用いて定義する）．例えば，図２のLHS節のヘッドにおける

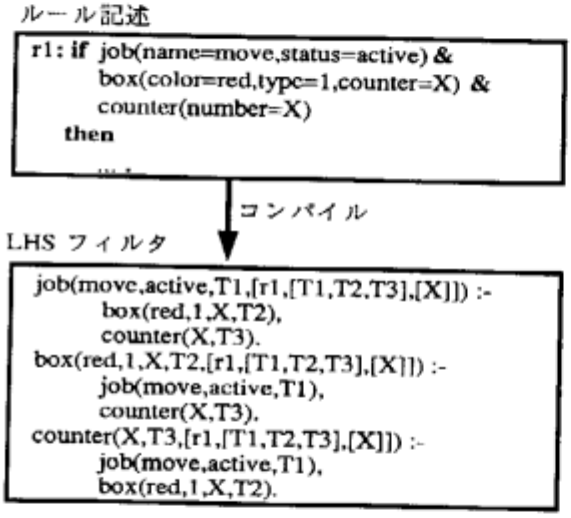

図１ LHSフィルタ生成の概略
Fig.1. Generating an LHS Filter

スロット値"move"，"active"は，それぞれスロット"name"，"status"に対する値である．変数Tiはタイムタグを表す．LHS節のヘッドの最後の引数はインスタンシエーションを表す．インスタンシエーションは要素数が３のリストで表現される．リストの第１要素，第２要素，第３要素は，それぞれ，ルール名，タイムタグのリスト，変数リストを表す．第２要素は，第１要素で示すルールのLHSとマッチしたWM要素のタイムタグのリストである．第３要素はルールのLHSに現れる変数のリストを表す．

LHS節は，対応するルールのLHSの具体的な意味解釈を与えている．例えば，図１上のルールは，条件要素"job"，"box"，および"counter"にマッチするWM要素が全て生じると起動可能になる．図１下で示される第１番目のLHS節は，条件要素"job"にマッチするWM要素が生じたなら，ルールr1が起動可能になるためにはその他に条件要素"box"および"counter"とマッチするWM要素が必要であることを表現している．同様に，第２番目，第３番目のLHS節は，それぞれ条件要素"box"や条件要素"counter"にマッチするWM要素が生じた場合に，ルールr1が起動可能になるために必要とされるその他の条件要素の必要条件を表現している．つまり，LHS節のヘッドはWM要素の変化を利用するための受け口として利用される．LHS節の本体は，ルールを満足させるために他に満たすべき条件要素の必要条件を表す．LHS節におけるこのようなルールチェック機能は，第２章で述べたプロダクション照合フィルタに関連した(a)Condition Membership，および(c)Condition Relationshipの知識を組み合わせたフィルタ機能を実現していることに相当する．

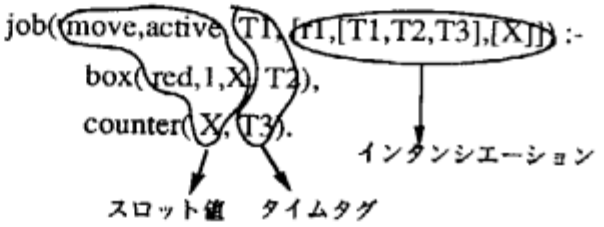

図2. LHS節の構成
Fig.2. Structure of an LHS clause

3.2 LHSフィルタを用いた照合過程

図１で示すように，LHSフィルタでのLHS節の数は条件要素の数だけ生成される．これはWM要素の変化に伴うLHS節の呼び出しを高速化するために，Prologのインデキシング機能を利用するためのものである．Prologのインデキシング機能は，Prologでは標準的に備わっている内部メカニズムであり，Prologにおいて節の参照を高速化するためのものである．例えば，Quintus Prologでは，節は節のヘッドのファンクタ名と第１引数を用いてハッシュインデキシングされる．この特長を生かすことにより，Prologデータベースへ選択的かつ高速にアクセスするPrologプログラミングが可能になる．そこで，本LHSフィルタでは第１引数になるべく定数がくるように，スロット値が用いられる．

図３は，図１で得られたLHSフィルタを用いた

照合過程の概略を示している．図３では，WMの変化として，KORE/IEのmakeコマンドを用いたWM要素の生成の例を図示している．makeコマンドは新たなWM要素をWMに付加する一方，WMの変化の分としてLHS節へのProlog queryに変換される．queryは，literalizeコマンドで定義されたスロット情報を参照することにより，LHS節のヘッドを呼び出す形式に変換され，実行される．図３においてqueryの第１，第２引数はスロットの値を表す．第２引数は，makeコマンドでは与えられていないので未定義変数として扱われる．第３引数はタイムタグであり，makeコマンドの実行時にシステムより自動的に付加される．第４引数は，求めるべきインスタンシエーションであり論理変数として与える．

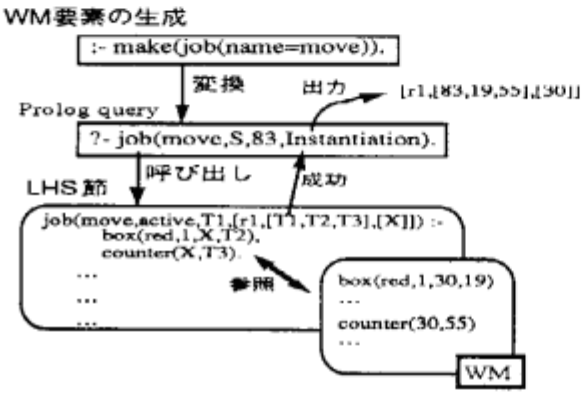

図3. LHS節へのquery例
Fig.3. A Prolog query tol.HS clauses

LHSフィルタを用いた推論過程において，このようなProlog queryへの変換は実行時に随時に行う必要はない．Prolog queryは，ルールをコンパイルする際に，RHSにおけるWMを変化させるコマンド（make, modify, remove等）を部分計算[23]することにより（推論を実行する前に）予め得られる．これにより，変換のための処理時間を前もって取り除くことができ，推論の高速化に貢献する．さらに，LHSの情報を用いることにより，部分計算をRHSのアクションに対して適用できるので実行時の推論の高速化が図れる．

図３において，Prolog queryはLHSフィルタを構成する第１番目のLHS節を選択的かつ高速に呼び出すことになる．これは，Prologの節がファンクタ名（この例では，"job"）と第１引数（この例では，"move"）を用いてハッシュインデキシングされるからである．本例において，queryは第１番目のLHS節のヘッドとのユニファイが成功するので，LHS節の本体のゴールが実行される．本体の第１ゴールおよび第２ゴールは，ルールを満たすべき他に必要とされるWM要素のパターンを表している．ゴールはWMを参照する．WMがPrologの内部データベースを用いて実現されているなら，この参照もPrologの節のインデキシング効果で選択的かつ高速に実行される．

LHS節本体のゴールが成功すると，LHS節のヘッドの最後の引数（ここでは，第４引数のインスタンシエーションを表すリスト）へ論理変数束縛の情報が節の本体からヘッドへ後向きに伝播される．その結果，呼び出したqueryの最後の引数である"Instantiation"に具現化された（つまり，変数を含まない）インスタンシエーション情報が出力される（ユニファイされる）．プロダクションシステムでは，普通，照合過程で複数のインスタンシエーションが生成される．本アプローチでは，Prologのバックトラッキング機能を素直に用いてLHS節に対するqueryの複数解を求めることにより，WM要素の変化に即した複数のインスタンシエーションを得ることができる．

3.3 負の条件要素とLHSフィルタ

ルール記述では，図１で示した正の条件要素以外に，負の条件要素がある．負の条件要素は，適合するWM要素が存在しないときに真となる．さらに，負の条件要素は，あるルールの負の条件要素のパターンとマッチするWM要素がWMから消去された場合も，そのルールのトリガーとなるように機能する．負の条件要素の機能を実現するためには，単純にPrologのnot（つまり，negation as failure）の機能を用いた実現では不十分である．そこで，このような負の条件要素の機能を実現するために，LHS節のヘッドの引数には，さらに条件要素の正負を表す引数IO（つまり，正の条件要素に対しては"＋"，負の条件要素に対しては"－"の値が用いられる）および，LHS節の呼び出し形式を表す引数Form（値として，"＊"もしくは"＝"がある）がある．引数IOの値は，LHSをコンパイルする際に前もって決定され，競合集合への操作（インスタンシエーションの付加および削除）のための情報として用いられる．引数Formの値は，照合過程においてLHS節が呼び出される際に決定される．具体的には，照合過程において，WM要素を生成する際に作られるProlog query（例えば，図３の場合）からは引数Formへ値"*"が渡される（つまり，unifyされる），また，WM要素を削除する際に作られるProlog queryからは引数Formへ値"="が渡される．引数Formの値は負の条件要素に関連したWM

要素が削除された場合の照合を実行するための情報として用いられる．

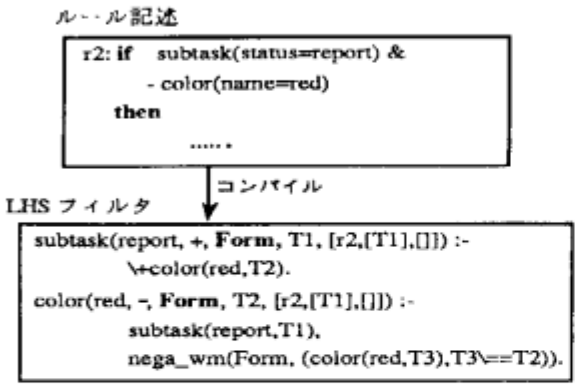

図４．負の条件要素を含むLHS フィルタ例
Fig.4. An LHS filter including a negative condition

例えば，図４上の負の条件要素をもつルールをコンパイルすることにより図４下で示すLHSフィルタが生成される．ここで，ルールのLHSにおいて第１番目のパターン（つまり，"subtask(status=report)"）が正の条件要素を，第2番目の記号"-"が付加されたパターン（つまり，"-color(name=red)"）が負の条件要素を表している．LHSフィルタにおいて，第1番目のLHS節は，LHSの第1番目の正の条件要素に関連したものであり，ヘッドの引数IO（ここでは，第2引数）には"＋"の値が割り付けられる．同様に，第2番目のLHS節のヘッドの引数IO（ここでは，第2引数）には"-"の値が割り付けられる．第2番目のLHS節の本体におけるnega_wmは，負の条件要素とマッチするWM要素が削除されたときにWMにまだ他にその負の条件要素とマッチするWM要素があるかどうかをチェックするためのものであり，概略は次のように定義される．

```
nega_wm(*, _) :- !.
nega_wm(=, X) :- ¥+ X.
```

ここで，図４のLHSフィルタを例にとり，負の条件要素に関連したWM要素の付加および削除にともなう照合過程を示す．図５上は，WM要素を付加するmakeコマンドの照合過程に関連した内部処理の概略を示している．ここで，述語callは，Prologの組み込み述語であり，引数で与えられた項をqueryとして実行する．ここでは，負の条件要素に関連したLHS節color（図４のLHSフィルタにおける第２番目のLHS節）に対するqueryが実行される．このqueryにおいて，整数"123"はタイムタグである．本例では，LHS節colorのヘッドの呼び出しは成功し，本体を実行する．本体の第２ゴールであるnega_wm述語の呼び出しは，第１引数が"*"として与えられるので無条件で成功する．もし，本体の第１ゴール呼び出しが成功するとLHS節colorに対するqueryが成功し，その結果，変数Instantiationに照合過程で求めるインスタンシエーションが出力される．queryが成功すると，次に述語assert_csが実行される．述語assert_csは，引数IOの値が"+"なら第２引数で得られるインスタンシエーションを競合集合へ付加し，引数IOの値が"-"なら得られたインスタンシエーションと同じタイムタグをもつインスタンシエーションを競合集合から削除する．本例では，LHS節colorに対するqueryの結果，引数IOの値は"-"に束縛され，得られたインスタンシエーションと同じタイムタグもつインスタンシエーションを競合集合から削除することになる．

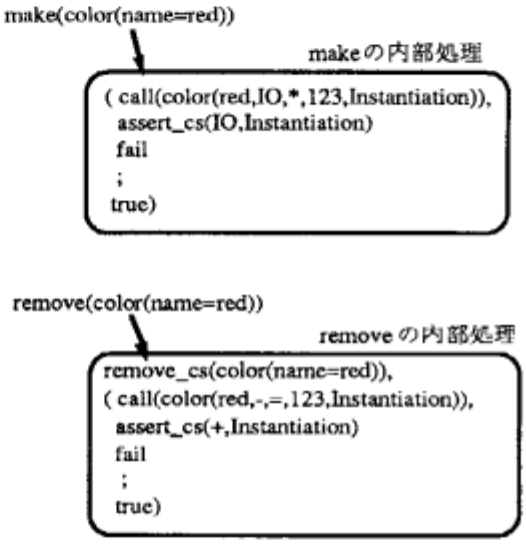

図５．負の条件要素に関する照合過程例
Fig.5. Matching process for a negative condition

図５下は，WM要素を削除するremoveコマンドの照合過程に関連した内部処理の概略を示している．ここで，remove_csは，削除されたWM要素（colorに関する）のタイムタグと同じタイムタグをもつインスタンシエーションを競合集合から削除する．次に，本例ではLHS節colorのヘッドの呼び出しは成功し，本体を実行する．本体の第２ゴールであるnega_wm述語の呼び出しは，第１引数が"＝"として束縛されるので第２引数で与えられるゴールのnotの判定を試みる．もし，本体の全てのゴールが成功すると，LHS節colorに対するqueryが成功し，その結果，変数Instantiationに照合過程で求めるインスタンシエーションが出力される．queryが成功すると，次に述語assert_csが実行される．本例では，引数IOの値は"＋"であるので，得られたインスタンシエーションが競合集合へ加えられる．

以上のような競合集合の生成過程は，第２章で述べた(d)のConflict-set Supportのフィルタ機能を実現したことに相当する．

3.4 LHSフィルタの特長

LHSフィルタの実現において特筆すべきことは，照合過程におけるjoin演算がPrologの論理変数への代入過程に帰着され，特別な機構を構築することなく効率的に実行されることである．つまり，WMの変化に伴うqueryは，LHS節へ適用されることにより，Prologの基本的計算メカニズム（つまり，反駁メカニズム）に基づいてjoin演算が高速に計算される．ここでのjoinの順は，WMを変化分（例えば，make されたWM要素）に対応する条件要素が先に来るように動的に順序が決定される．さらに，LHS節の本体に着目したjoin演算の最適化が可能である．具体的には，Prologにおける問い合わせの最適化14)に基づいて，LHS節の本体のゴール（つまり，条件要素）の並べ替えを行うことによりjoin演算の最適化を図ることができる．LHSフィルタでは，現在，LHS節の本体のゴールの並べ替えによるjoin演算のための最適化は行なっていない．これは今後の課題である．

LHSフィルタでは，第２章で述べたプロダクション照合フィルタに関する(b)のMemory-supportの機能は採用していない（つまり，照合過程において中間結果を保存しない）．採用しない理由は，Prolog処理系が，一般に，このような中間結果を保存・更新するためには多くのオーバーヘッドを必要とするからである．むしろ，LHSフィルタでは，Prologの節の高速な参照機能を効果的に利用することにより，照合時の再照合の効率化を図っている．再照合は，LHS節の本体において，ハッシュインデキシング効果によりWM要素を選択的に参照することにより高速化される．つまり，LHSフィルタは，第２章で述べたプロダクション照合フィルタに関する(a)のCondition Membership，(c)のCondition Relationship，および(d)のConflict-set Support知識を組み合わせた照合フィルタ（Condition-Membership Condition-Relationship Conflict-set-Supportフィルタ）の機能を実現する．

3.5 認識-行動サイクル

プロダクションシステムKORE/IEにおいて，LHSフィルタを用いた認識-行動サイクルは図６のように示せる．図６において，Prolog節はKORE/IEの推論ステッパーであるrunコマンドの定義の概略を表している．認識-行動サイクルはrunコマンドを用いて実行される．引数Ｎは認識-行動サイクルの繰り返し数を指定するために用いられる．節の本体は，三つのゴールにより構成される．第１番目のゴールは，競合集合から予め指定された競合解消戦略を用いてひとつのインスタンシエーション"Instantiation"を選ぶ．第２番目のゴールは選ばれたインスタンシエーションの情報を用いて，選ばれたルールのRHSを実行する．普通，RHSの実行はWMの変更を伴う．WMの変更は，3.2節で述べたLHSフィルタを用いた照合過程に帰着され，競合集合を更新する．RHSの実行では，複数のWM要素のWM への付加および削除が行なわれる（更新は，削除と付加の組み合わせである）．競合集合の計算は，WMの変更に伴い随時行なわれ，全てのRHSの実行（つまり，複数のWMの変更）が終了した後に次の競合解消のステップに移る．

認識-行動サイクルは，繰り返し数Ｎから１を減じて，第３番目のゴールを呼び出すことにより再帰的に繰り返される．サイクルは，引数Ｎが０になるか，もしくは，第１ゴールでインスタンシエーションが得られない時（つまり，競合集合が空の時）に終了する．

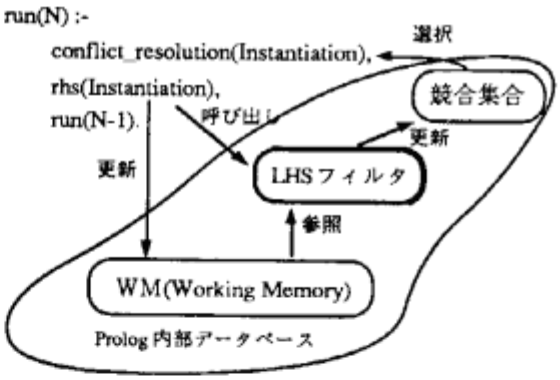

図6. LHSフィルタを用いた認識-行動サイクル
Fig.6. A recognize-act cycle using LHS Filter

図６で示すように，KORE/IEでは，節のハッシュインデキシング効果を積極的に利用するために，WMや競合集合をPrologの内部データベースを用いて実現している．そのために，WMや競合集合の更新には，Prologのassertおよびretract機能を最小限利用している．LHSフィルタの枠組みにおいて，WMや競合集合を引数として持ち歩くことも可能であり，Prologのassertおよびretract機能の使用は回避できる．これにより，ある程度の高速性は犠牲になる一方，プログラムの保全性（maintenabilty）と拡張性は向上する．その際，LHS節の本体におけるWM要素のチェック機能はmember述語等を

用いて実現される．

4．メモリスペースの削減

図1で示したように，LHSフィルタでのLHS節の数は条件要素の数だけ生成され，メモリスペースを必要とする．メモリスペースの削減のためには，条件要素ごとにLHS節を作らない必要がある．これは，LHS節ヘッドに対するPrologインタプリタ（例えば，C-Prolog）のインデキシング効果を犠牲にすることになる．しかしながら，Prologコンパイラの利用やプロダクション照合フィルタとしてLHSフィルタと同様な機能を実現することにより実用的な高速性が期待できる．図7にメモリスペースを考慮したLHSフィルタの概略を示す．例として図1におけるルールを用いる．本フィルタを便宜上，m-LHSフィルタと呼び，m-LHSフィルタを構成するProlog節をm-LHS節と呼ぶ．

Prolog処理系の多くは，第1引数の最も左のリテラルだけを調べてインデキシングを行うので，そのようなProlog処理系では，インデキシングによるm-LHSフィルタの高速化は期待できない．一方，K-Prolog[26]のように，インデキシングのために，述語のすべての引数について，第2レベルの引数まで調べるようになれば，m-LHS節ヘッドの第一引数に対するインデキシング効果を得ることができ，m-LHSフィルタはLHSフィルタと等価な高速性を実現する．

本アプローチの特長は，ひとつのルールをなるべくひとつのm-LHS節へ対応させたことにある．LHSにおいて同じクラス名（すなわち，条件要素を表すProlog項のファンクタ名）を持った条件要素があった場合は，その数の分だけのm-LHS節が生成される．これは，WMの変化に伴った照合過程を実現するためである．メモリスペースは，図7のようなm-LHS節を構成することにより効果的に削減できる．削減量はルールの条件要素の数に依存するが，例えば，本例のような場合(条件要素が3つのルール)，約60%のメモリスペースを削減する．メモリスペースの計測には，Quintus Prologにおける述語statisticsを用いた（ここでは，LHS節およびm-LHS節をPrologへロードした際に消費したプログラム空間の量（バイト数）を用いた）．

図7で示すように第一引数のリスト（このリストを，便宜上，リストLと呼ぶ．）におけるリスト要素の位置はクラス名ごとに決定される．このようなリストLを構成するために，クラス名がliterarize宣言された順を用いる．例えば，本例では，クラス名job，box，およびcounterがこの順でliteralize宣言されている．本m-LHS節のヘッドの第2，3引数は，それぞれ，3.3節で述べた引数IO，引数Formに相当する．第4引数は，図2で示したインスタンシエーションと同じである．第1引数のリストLの構成に着目することにより，WMの変化に即した照合過程を実現できる．

リストLの長さは不定長（つまり，リストの後半部が未定義変数になっている）で表現される．これは，ルールのインクリメンタルなコンパイルを実現するためである．リストLの前半部（つまり，"|"より前の部分）で与えられている要素の数はルールベースで用いられる異なったクラス名の総数に相当する．ルールがインクリメンタルにコンパイルされ，新たなクラス名が用いられた場合，リストLの前半部の最後に新たなクラスのための場所が確保されたm-LHS節が生成される．この時，既にあるm-LHS節のリストLの後部を変更する必要はない．なぜなら，図8で示すように，リストLを呼び出す際には，リスト要素の前からの位置が利用されるからである．リストLの要素は，図8で示すようにスロット値およびタイムタグのリストで構成される．スロット値の並びの順は，3.1節で述べたようにliteralize宣言により予め決定される．

```
lhs_clause([[move,active,T1],[red,1,X,T2],[X,T3]|_],
                        +, Form, [r1,[T1,T2,T3],[X]]) :-
        job(move,active,T1),
        box(red,1,X,T2),
        counter(X,T3).
```

図7. メモリスペースを考慮したLHS節
Fig.7. A new LHS clause for reducing memory space

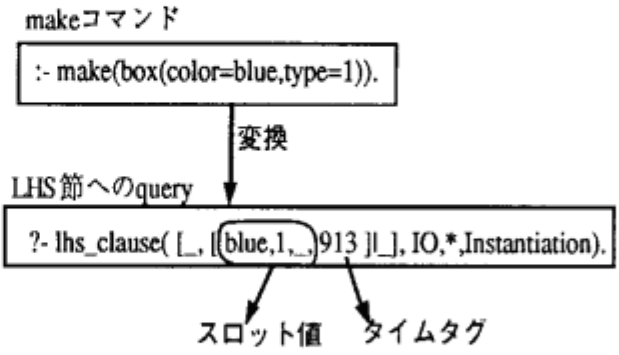

図8. 新LHS節へのquery
Fig.8. A query to New LHS clauses

図7におけるm-LHS節を呼び出すためのPrologqueryは，図8で示される．図8で示すように，例えば，queryはWMを変化させるコマンドmakeにより生成され，m-LHS節を呼び出す．この時，パターンboxに関連した（WM要素の変化に関連した）インスタンシエーション"Instantiation"が計算される．これは，パターンboxは2番目にliteralize宣言されているので，リストLの2番目の位置だけが具体化されたqueryを生成し用いたからである．これにより，第2章で述べたConflict-set-support照合フィルタの機能を実現する．さらに，m-LHS節の本体に必要な条件要素を列挙したことにより，図1のLHSフィルタがもつその他のフィルタ機能も実現する．

5．プロダクションシステムKORE/IE

KORE/IEは，LHSフィルタを用いた高速な前向き型推論プロダクションシステムである．KORE/IEの機能はOPS5の機能を包含しており，OPS5的なルールプログラミングが可能である．OPS5によるルールプログラミング技法は，現在において数多く蓄積されており[4]，これら技法を利用できることはシステムの実用性を向上させるうえで重要なことである．本章では，プロダクションシステムKORE/IEの概略について論じる．

5.1 ルール記述

KORE/IEにおけるルール記述は，Prolog項の記述のシンタックスを十分に取り入れることにより，その可読性とルール表現の柔軟性を実現している．KORE/IEにおけるルールは，(1)ルール名,(2)シンボル"：",(3)シンボル"ｉｆ",(4)LHS(left hand side),(5)シンボル"ｔｈｅｎ",(6)RHS(right hand side),(7)シンボル"."により構成され，次のように記述される．

RULE_NAME:
　if　Condition$_1$ & ... & Condition$_n$
　then　Action$_1$ & ... & Action$_m$.

ここで，RULE_NAMEはルール名である．Condition$_i$, Action$_j$は，それぞれLHSにおけるルールの条件要素，RHSにおけるルールのアクション（RHSアクションと呼ぶ）を示す．複数の条件要素やRHSアクションはシンボル"&"を用いて区切られる．スロット記述には，等しいことを示す"="以外に，"＼＝＝", "<", ">"等のPrologで用いられる比較式を同様な意味で用いることができる．

WM要素の内部表現は次のようなProlog項で表現される．

ｸﾗｽ名（値1, 値2, ..., 値n, ﾀｲﾑﾀｸﾞ）

タイムタグは，WM要素の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．値1, 値2, ..., 値nはパターンのスロットの値を表す．値とスロットを対応させるために，内部表現の引数の順はliteralizeコマンドにより予め定義される．RHSアクションとしては，WMを直接に操作するコマンドとして，make（WM要素の作成）,remove（WM要素の削除）,modify（WM要素の更新）が用意されている．それ以外のアクションはPrologの組み込み述語やユーザ定義の述語を用いる．

5.2 KORE/IEの特長

KORE/IEではルールベースごとに競合解消戦略の定義を可能とすることにより，より柔軟なルールベース間の協調問題解決機能を実現する．KORE/IEにおける協調問題解決は，黒板モデル[13]における知識源間の協調問題解決方式に相当する．KORE/IEにおける協調機構実現手法の特長は，特別なメタな制御機構を構築することなしに，協調のための制御機構としてPrologの計算過程を直接的に利用することにある．具体的には，ルールベースごとにLHSフィルタを用いた照合過程を実行し，各ルールベースのインスタンシエーションを認識-行動サイクルごとに実行することにより実現される．ここで，ルールベースの優先順位は予め定義される[19]．

例えば，KORE/IEにおいて，OPS5におけるMEA戦略[6]は次のように定義される．

define_strategy mea :=
　select latest instantiation
　　by comparing first time tag.

ここで，下線を施した部分はユーザが記述する所であり，他はキーワードである．下線部は，それぞれ，戦略名，インスタンシエーションの選択基準の指定（他にoldestを指定できる），比較すべきタイムタグの指定（他に，second, third, fourth, fifthまたは整数を指定できる）を表している．KORE/IEにおいて，競合解消戦略の基本的内部処理はLEX[6]を行っている．これは，インスタンシエーションの生成時にタイムタグを大きい順にソートし，競合解消のときにソートされたタイムタグリストの要素を順に比較することにより実現している．上記のMEAの定義は，そのタイムタグリストをソートする際に，第一条件要素のタイムタグがソートリストの先頭になるように制約するものである．その結果，上記の記述だけで，第一条件要

素のタイムタグが一致する場合はLEXと同じ戦略なるMEA戦略の定義が実現される．

以上のような競合解消戦略定義は，次のように，ルールベースに対してその戦略名を宣言することによりルールベース固有の競合解消戦略として用いられる．

?- strategy(<Rule_base>,<Strategy>).

ここで，第1引数に具体的なルールベース名，第2引数に具体的な競合解消戦略名を指定する．

また，KORE/IEでは，不確実な知識に基づく推論を効果的に実現するためにTMS[5]を利用した非単調な推論メカニズムを実現している[21]．非単調な推論メカニズムの実現は、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の構築を容易にする．

6．評価

元来，Prologをベースにしたシステムは Lispをベースにしたシステムに比べプログラムの実行速度が非常に遅いとされている．そこで，本性能評価では，Prologシステム上にインプリメントしたプロダクションシステムKORE/IEと，Lisp上では最も早いとされるプロダクションシステムであるOPS5との比較をすることにより，KORE/IEの高速性（間接的には，LHSフィルタの高速性）を示すことにする．表1は，SUN3/260上でインプリメントしたKORE/IE(Quintus Prolog (Release 1.6)を用いた)とOPS5(Franz Lisp (Opus 42.19)を用いた)の比較を示している．例題はMonkey and Bananas1)(27ルール)である．本例題は，文献2)において様々なエキスパートシステム（例えば，CLIPS,OPS5,ART,KEE等）のベンチマークテストに用いられた標準的な例題であり，手段-解析に基づく問題解決能力をテストするものである．KORE/IEのルール記述はOPS5のルール記述と一対一に対応づけた（KORE/IEでは，OPS5的なルールプログラミングが可能であり，LHSおよびRHS記述を表現的に同様にした）．

表1．ルールの実行
Table1. Rule Execution Time

	旧 KORE/IE	新 KORE/IE	OPS5
time(秒)	1.24	0.81	0.79

例題: monkey & banana (27ルール)
計算機: SUN3/260
KORE/IE: Quintus Prolog(Release 1.6)
OPS5(VPS2 Version): Franz Lisp(Opus 42.16)

表1において，OPS5はLiszt(Franz Lisp Compiler)でコンパイルされたものである．"旧KORE/IE"は文献20)で試作したKORE/IEであり，"新KORE/IE"は，本研究により新たに改良されたLHSフィルタを用いた新たなKORE/IEのバージョンである（本評価では，m-LHSフィルタ（4章参照）は特に対象としていない）．それぞれのKORE/IEバージョンは，Prologコンパイラーによりコンパイルされ，さらにルール記述から得られたLHSフィルタもPrologコンパイラーを用いてコンパイル実行した．表1で示すようにProlog上の"新KORE/IE"はPrologのインデキシング機能を効果的に利用したことにより，Lisp上のOPS5に十分に匹敵する早さを実現している．これは，LHSフィルタがPrologの上で効率的に機能していることを示してる．

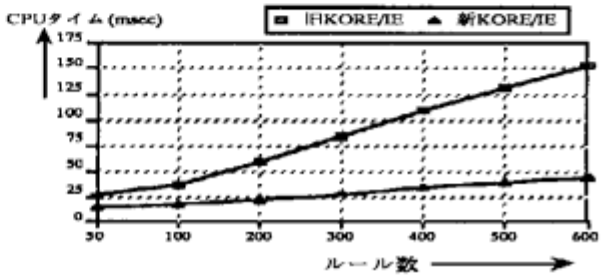

図9. LHSフィルタの評価
Fig.9. Timing LHS filters

```
r1 : if f(s1=1, s2=X) & g(s1=1, s2=X)
     then
       make(f(s1=2, s2=2)) & make(g(s1=2, s2=2)).

r2 : if f(s1=2, s2=X) & g(s1=2, s2=X)
     then
       make(f(s1=3, s2=3)) & make(g(s1=3, s2=3)).
     :
     :
rn: if f(s1=n, s2=X) & g(s1=n, s2=X)
    then
      make(f(s1=n+1, s2=X+1)) & make(g(s1=n+1, s2=X+1)).
```

図10. テストルール
Fig.10. The test rulea

図9は，"旧KORE/IE"と"新KORE/IE"をルール数を増やしていくことによる性能を示している．本性能評価で用いたテストルールは，図10で示すように，WMの要素を次々に付加して行くものである．本テストルールは，その実行においてjoin演算も含まれ，通常のルールプログラミングで良く用いられる形態のサブセットとなっている．本テストは，ルール数およびWM要素数を増やすことにより，インデキシングの効果を評価するためのものである．図9では，表1で用いたOPS5とは特に比較していない．なぜなら，ここでのOPS5は，

Forgyによりインプリメントされたパブリックドメインバージョン（VPS2バージョン）であるが，前評価の結果，本OPS5がαメモリに関してハッシングをきちんと行なっていないと思われたからである．最新のOPS5バージョンは，αメモリに関してハッシングしているように思われ，本システムとの比較・評価は今後の課題である．

図9のグラフにおいて，縦軸はルールひとつ当たりの平均実行時間(CPU時間(ミリ秒))であり，総実行時間をテストルール数で割り算したものである． 横軸はルール数を示している．図9で示すように"新KORE/IE"は，LHSフィルタがPrologにおける節のヘッド探索の高速性（つまり，インデキシング効果）を十分に取り入れたことにより，ルール数にほとんど依存しない理想的な実行速度を達成している．

7．むすび

RETEアルゴリズムやTREATアルゴリズムにおいて，照合過程の高速化に寄与した本質的な点は，認識-行動サイクルごとに全ての条件要素と全てのWM要素を照合するのではなく，WMの変化分のみに着目することにより，不必要な照合の回数を減らしたことである．さらに，照合の中間結果を保存することにより，以前と同じ照合の無駄な繰り返しを回避している．LHSフィルタは，新たな視点でPrologの利点を生かすことにより，照合過程の高速化に寄与する本質な点を効果的に実現したプロダクション照合フィルタである．LHSフィルタの実現において，Prologの利点として，Prolog節のインデキシング機能，論理変数束縛機構（ユニフィケーション）を利用した．LHSフィルタ実現の枠組において，インデキシング機能は，ルールおよびWM要素を選択的かつ高速に参照するために利用された．論理変数束縛機構はLHSとWM要素の照合およびjoin演算を効果的に実現するために用いられた．

LHSフィルタの実現において特筆すべきことは，join演算がPrologの論理変数への代入過程に帰着され，特別な機構を構築することなくPrologの基本的計算メカニズムにより効率的に実現されたことである．LHSフィルタでは，照合過程において中間結果を保存しない．これは，LHSフィルタの主な特徴である．保存しない理由は，Prolog処理系が，一般に，このような中間結果を保存・更新するためには多くのオーバヘッドを必要とするからである．むしろ，LHSフィルタでは，Prologの節のハッシングによる高速な参照機能を効果的に利用することにより，照合時の再照合の効率化を図った．LHSフィルタのその他の特長として，ルールのインクリメンタルなコンパイル機能がある．これは，RETEネットワークでのような共有メモリ方式を採用しなかったことによる．LHSフィルタにおいて，ルールのインクリメンタルなコンパイルはLHS節をインクリメンタルに宣言して行くことにより容易に実現する．LHSフィルタは，第2章で述べたプロダクション照合フィルタに関する(a)のCondition Membership，(c)のCondition Relationship，および(d)のConflict-set Support知識を組み合わせた照合フィルタ（Condition-Membership Condition-Relationship Conflict-set-Supportフィルタ）として特徴づけられる．

LHS節フィルタの枠組みは，競合集合やWMをLHS節の引数として実現することにより，副作用をもつPrologのassertおよびretract機能を利用することなしに効率的に実現できる[9]．この性質は，GHC[22]等の並列論理プログラミング言語を用いた並列化に適している．LHSフィルタに基づくプロダクションシステムの並列化は，今後の課題である．

本研究の目的はProlog上に十分に高速でありかつ実用的機能性を実現するプロダクションシステムを構築することである．LHSフィルタを利用したプロダクションシステムKORE/IEは，ルールの記述力を制限することなしに推論の高速性を実現した．第6章では，具体的にベンチマークテストを用いることによりKORE/IEの性能評価を行なった．性能評価の結果，KORE/IEの高速性は，Lisp上にインプリメントされている前向き推論型プロダクションシステムであるOPS5に匹敵することを示した．本研究の結果は，Prologの実用性を実証したものであり，Prologを用いて実用的な推論システムを構築する際の有効なプログラミング指針を提供する．

なお，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なわれたものである．

謝辞 日頃よりご指導頂く当研究所戸田光彦研究員ならびに國藤進研究員に感謝いたします．本研究をまとめるに当たり、貴重な御意見を頂いたICOT研究担当次長の古川康一氏に深謝いたします．KORE/IEを試作・改良するに当たり，SWCの二神浩道氏の協力を感謝いたします．

参考文献

1) Brownston, L. , Farell,R. and Kant,E. : Programming expert system in OPS5, Addison Wesley(1985)

2)Brekke,B. : Benchmarking Expert System Tool Performance, Ford Aerospace Tech Note(1986).

3)Clark,K.L. and McCabe,F.G. : PROLOG:A Language for Implementing Expert Systems, in Machine Intelligence 10, pp.455-470(1987).

4)Cooper,T. and Wogrin,N. : Rule-based Programming with OPS5, Morgan Kaufman Publishers(1988).

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

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

7)Forgy,C.L. : Rete :A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, Artificial Intelligence Vol.19, pp.17-37(1982).

8)古川: プロダクションシステム, Prolg入門, オーム社,pp.126-133(1986).

9)Furukawa,K., Fujita,H. and Shintnai,T.: Deriving an Efficient Production System by Partial Evaluation, Proc. The North American Conference on Logic Programming '89,pp.661-674,October(1989) .

10)Gupta,A., Forgy,C.L., Newell,A. and Wedig,R.: Parallel algorithms and architectures for rule-based systems, International Symposium on Computer Architecture, pp.28-37(1986).

11)広瀬：プロダクションシステム記述言語POPS2, 情報処理学会研究会報告, 86-PL-8, (1986).

12)石田: プロダクションシステムにおける条件記述の最適化,情報処理学会論文誌,Vol.29No.12,pp.1158-1169(1988).

13)Lesser, V.R. and Erman,L.D. : A retrospective view of the HEARSAY-II architecture, Proc. of IJCAI 5, pp.790-800(1977).

14)Li, D. : A Prolog Database System, Research Studies Press, England(1984).

15)McDermott,J., Newell, A. and Moore,J. : The Efficiency of Certain Production Implementations, in Pattern Directed Inference Systems, Academic Press, pp.155-176(1978).

16)Miranker,D.P.: TREATE:A Better Match Algorithm for AI Production Systems, AAAI-87,pp.42-47(1987).

17)Mizoguchi,F., Miwa,K. and Honma,Y.: An approach to PROLOG Based Expert System, Proc.Logic Programming '83,pp.22-24(1983)

18)Nayak, P., Gupta, A. and Rosenbloom,P.: Comparison of the Rete and Treat Production Matchers for Soar (A Summary), AAAI-88,pp.693-698(1988).

19)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-, Proc.Logic Programming '87,予稿集,pp.233-242(1987).

20)Shintani,T. : A Fast Prolog-based Production System KORE/IE, Proc. of the Fifth International Conference andSymposiumonLogicProgramming,MITPress,pp.26-41(1988).

21)新谷: プロダクションシステムKORE/IEにおける非単調推論 , Proc.Logic Programming '88,予稿集 ,pp.73-82(1988).

22)Ueda,K. : Guarded Horn Clauses, Proc. Logic Programming'85,LNCS-221,Springer-Verlag,pp.168-179(1986).

23)竹内,古川: Prologプログラムの部分計算とメタプログラムの特殊化への応用,TheLogic Programming Conference 85, 予行集,pp.155-165(1985).

24)紀：Prologコンパイラの開発, archive No.10, CQ出版社, pp92-131(1989).