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††

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．本研究の目的は，Prologに内在する特長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前向き推論型プロダクションシステムである OPS 5 に匹敵することである．本研究では，Prolog 処理系に適したプロダクション照合フィルタを実現し，照合過程を高速化する．ここで得られたプロダクション照合フィルタは LHS フィルタと呼ばれる．LHS フィルタの実現において，Prolog のインデキシング機能および論理変数束縛機能を利用する．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る．論理変数束縛機能は照合過程における join 演算を効果的に実現するために用いられる．LHS フィルタは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した．

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 OPS 5[6] に匹敵することである．OPS 5 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[1),8)]，(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[3),17)]，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あらかじめ部分計算し，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

† Speeding up the Matching Filter for Prolog-based Production Systems by TORAMATSU SHINTANI (International Institute for Advanced Study of Social Information Science, FUJITSU LIMITED).

†† 富士通(株)国際情報社会科学研究所

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/IE[20]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)の照合はすべてのルールの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．サイクルごとにすべ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の知識である．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すべてのルールの 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入力ノードに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

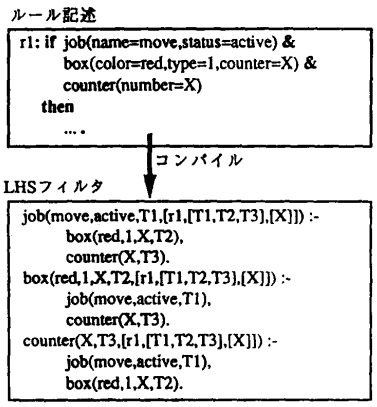

図1 LHS フィルタ生成の概略
Fig. 1 Generating an LHS filter.

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あらかじめ定義される（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，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．

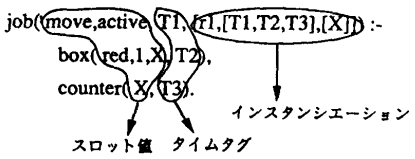

図2 LHS 節の構成概略
Fig. 2 An outline of an LHS clause.

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 要素がすべ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．

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

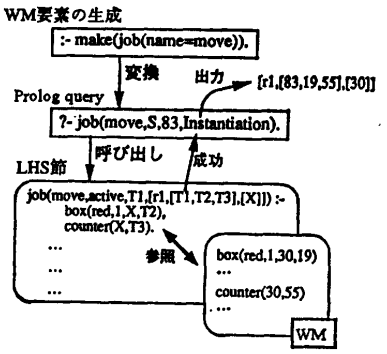

図 3 LHS 節への query 例
Fig. 3 A Prolog query to LHS clauses.

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．

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）あらかじめ得られる．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

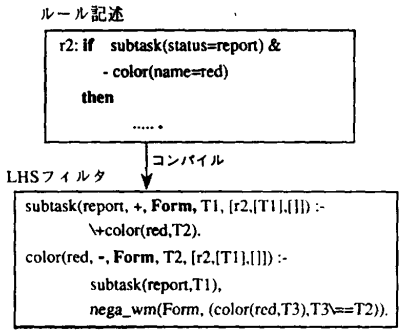

図 4 負の条件要素を含む LHS フィルタ例
Fig. 4 An LHS filter including a negative condition.

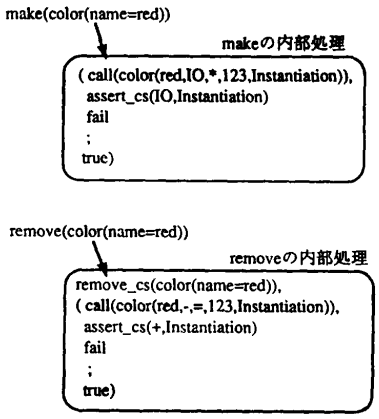

図 5 負の条件要素に関する照合過程例
Fig. 5 Matching process for a negative condition.

要素に関連した WM 要素が削除される場合の照合を実行するための情報として用いられる.

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ことになる.

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, 本体のすべ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，競合集合からあらかじ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われ，すべての 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．

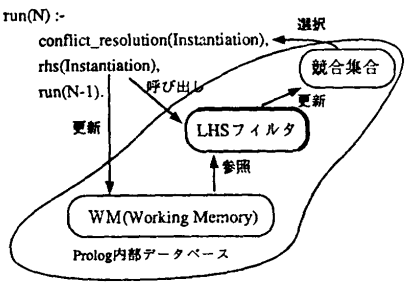

図 6 LHS フィルタを用いた認識-行動サイクル
Fig. 6 A recognize-act cycle using an LHS filter.

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とき（つまり，競合集合が空のとき）に終了する．

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（maintainability）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[24] 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1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（条件要素が三つのルール），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）．

```
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7で示すように第1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 literalize 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．このとき，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 宣言によりあらかじめ決

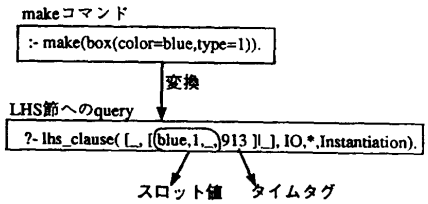

図 8 新 LHS 節への query
Fig. 8 A query to New LHS clauses.

定される.

図7における m-LHS 節を呼び出すための Prolog query 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．このとき，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 OPS 5 の機能を包含しており，OPS 5 的なルールプログラミングが可能である．OPS 5 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，タイムタグ）

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 コマンドによりあらかじめ定義される．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あらかじめ定義される[19]．

例えば，KORE/IE において，OPS 5 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，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 と同じ戦略になる 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，SUN 3/260 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 42.16) を用いた) の比較を示している．例題は monkey & banana) (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いる．

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

表1 ルールの実行
Table 1 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：SUN 3/260, KORE/IE：Quintus Prolog (Release 1.6), OPS5 (VPS2 Version)：Franz Lisp (Opus 42.16).

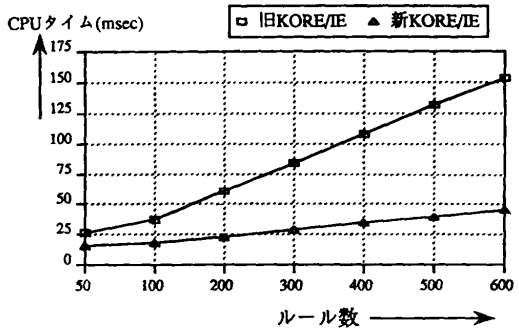

図9 LHS フィルタの評価
Fig. 9 Timing LHS filters.

```
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)).
     :
     :
m: 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 rules.

に，WM の要素を次々に付加していくものである．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（VPS 2 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すべての条件要素とすべての 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的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い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 OPS 5*, 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*, Hayes, J. E., Michie, D. and Pao, Y.-H. eds., pp. 455-470, Ellis Horwood (1982).
4) Cooper, T. and Wogrin, N.: *Rule-based Programming with OPS 5*, Morgan Kaufman Publishers (1988).
5) Doyle, J.: Truth Maintenance System, *Artif. Intell.*, Vol. 12, pp. 231-272 (1979).
6) Forgy, C. L.: OPS 5 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, *Artif. Intell.*, Vol. 19, pp. 17-37 (1982).
8) 古川：プロダクションシステム，Prolog 入門，pp. 126-133，オーム社 (1986).
9) Furukawa, K., Fujita, H. and Shintani, T.: Deriving an Efficient Production System by Partial Evaluation, *Proc. the North American Conference on Logic Programming '89*, pp. 661-674 (Oct. 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 POPS 2，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. 29, No. 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 (1984).
15) McDermott, J., Newell, A. and Moore, J.: The Efficiency of Certain Production Implementations, in *Pattern Directed Inference Systems*, Waterman, D. A. and Hayes-Roth, F. eds., pp. 155-176, Academic Press (1978).
16) Miranker, D. P.: TREAT: 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 and Symposium on Logic Programming*, 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, 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，*The Logic Programming Conference '85*，予稿集，pp. 155-165 (1985).
24) 紀：Prolog コンパイラの開発，*archive*, No. 10, pp. 92-131，CQ 出版社 (1989).

(平成元年 8 月 31 日受付)
(平成 2 年 10 月 9 日採録)

新谷 虎松（正会員）

1955 年生．1980 年東京理科大学理工学部経営工学科卒業．1982 年同大学院修士課程修了．同年より富士通(株)国際情報社会科学研究所に勤務．論理プログラミング，知識利用技術，意思決定支援，CSCW などの研究に従事．電子情報通信学会，日本ソフトウェア科学会，日本認知科学会各会員．