反駁メカニズムに基づく推論エンジンKORE/IEの高速化

4K－6

新谷虎松[1]・二神浩道[2]
([1]富士通(株)国際情報社会科学研究所・
[2](株)富士通ソーシャルサイエンスラボラトリ)

1. はじめに

KORE/IEは問題解決支援環境KORE（Knowledge Oriented Reasoning Environment）[1]における推論エンジン・サブシステム（部品）として機能する。一方、単独では、OPS5で代表される前向き推論型プロダクションシステムとして用いることができる。本講演では、特に、KORE/IEにおける推論の高速化のために用いた手法について議論する。

本手法は、認識-行動サイクルにおける構成要素を高速化するために、論理型言語の利点（例えば、高速な反駁（Refutation）メカニズムや部分計算技法など）を素直に導入することにより、その高速性を実現する。

2. 論理型言語による推論エンジンの概観

Prologを代表とする論理型言語は，その言語内に推論エンジン構築に必用なパターン・マッチング（ユニフィケーション）機構や解の探索のためのバックトラッキング機構を包含しており、また、それ自身、反駁メカニズム[2]に基づく強力な推論エンジンとして利用することができる。一般的に、推論システムを実現するには、ルール解釈実行機構の設計・構築が必用となるが、論理型言語を推論エンジンとして見なすことにより特別なルール解釈実行機構を構築することなしに容易に推論システムを構築することが可能である。この種のシステム(DCG, BUPなど）ではルールを論理型言語のプログラムに変換し、そのプログラムの実行という形で推論システムが構成される。

一方，竹内[3]はこの種の推論の実行効率とルール・インタープリタとしてのルール解釈実行機構実現による利点（ルールの汎用性，maintenability，readability）を統合するための技法として部分評価技法に基づく推論システム構成手法を提案している。この手法は、ルールをデータとして与えることによりルール解釈実行機構をそのルールに対して特殊化し、推論を論理型言語のプログラムとして直接実行することで論理型言語上に構築した各種インタープリター般に対しての高速化を指向する。この高速性は論理型言語の実行環境である反駁メカニズムの高速性に帰着する。しかしながら、この手法による推論システムは、推論エンジンの内部メカニズム（例えば、競合解消戦略、ワーキングメモリーの検索・更新、認識-行動サイクル等）の特徴を積極的に利用してないので、更に高速化することが可能である。

KORE/IEでは以上のような推論システム構築技法を背景にして、論理型言語上で十分に高速な推論エンジンを実現するために、推論エンジンのメカニズムの特徴を利用し、論理型言語の反駁メカニズムの高速性を直接利用することにより（ルール解釈実行機構実現による利点を損なうことなしに）その高速化を実現する。

3. KORE/IEにおける推論の高速化技法の概略

KORE/IEにおける推論動作は、OPS5等に代表されるプロダクション・システムと同様に、ルール解釈実行機構を意識した（つまり、①プロダクション記憶,②作業記憶,③推論部,の構成要素を考慮した）認識-行動サイクル(Recognize-Act cycle)により実現される。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。そこで、推論の高速化は、(1)～(3)の各動作を高速化することにより効果的に達成できる。この中で最も多くの処理時間が必用とされるのが(1)の照合であり、推論の高速性を左右する本質的な動作である。照合は全てのルールの条件部（ＬＨＳ(Left Hand Side)と呼ぶ）と作業記憶の内容を照らし合わせて、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が選択される。

サイクル毎に全てのルールのＬＨＳ要素と作業記憶要素を照合すると、ルール数や作業記憶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。OPS5では、この様なサイクル毎に照合を行うことを避けるために、Rete Matchアルゴリズム[4]を用いている。Rete Matchアルゴリズムは予め全てのルールの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）しておき、サイクルに関係なく、作業記憶の変化の分だけこのネットワークとの照合が行われ（ネットワーク内に照合に関する情報（作業記憶要素、

Speeding up of Inference Engine KORE/IE based on Refutation Mechanism
Toramatsu SHINTANI[1], Hiromichi Futagami[2]
[1]IIAS-SIS,FUJITSU LIMITED, [2]FUJITSU SSL,Ltd.

変数の束縛等）は全て格納される）、インスタンシエーションの更新を行う。このようなRete Match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データ構造（ポインタ等）が必要であり、論理型言語で効率的に構成することは困難である。

そこで、KORE/IEでは、認識-行動サイクルにおける照合の処理時間を高速化するための方略として、先に述べた部分評価技法（広義の意味で解釈し、与えられたプログラムの中で、計算できる部分を計算し、計算できないものは元のままに残しておくこと）を用いる。具体的には、作業記憶要素の変化をデータとしてＬＨＳを動的に特殊化することであり、この特殊化されたものはカレントな作業記憶要素を表現しており、Rete Matchアルゴリズムにおけるネットワークを用いた作業記憶要素の表現に対応する。また、この時のインスタンシエーションの追加・更新は、論理型言語におけるプログラムの実行（高速な反駁メカニズム）による副作用として得られる。

4. ルールのコンパイル

KORE/IEではルールを論理型言語のプログラムにコンパイル（もしくは、変換）することにより、推論の高速化を実現する。このコンパイルにおいては、認識-行動サイクルにおける各段階を高速化するために、ルールのLHSとRHSのそれぞれに対応した論理プログラムが生成される。ここで、前者をLHSコンパイル、後者をRHSコンパイルと呼ぶ。LHSコンパイルは、先に述べた認識-行動サイクルにおける照合の高速化のために、LHSの動的特殊化ための論理プログラムを生成する。RHSコンパイルは競合解消段階で選ばれたインスタンシエーションをもとに高速に動作（例えば、LHSからの変数の引渡し等を含む）を実行するための論理プログラムを生成する。例えば、次のKORE/IEルールは、

```
気温: if 気象情報(場所＝Place,気温＝Temp) &
        仕事(名前＝"気温の測定",場所＝Place)
     then
        modify(1,[気温＝compute(Temp＋10)]).
```

LHSコンパイルにより、LHS要素ごとに、つぎの論理プログラム（シンタックスはC-Prologに準ずる）を得る。

```
lhs_call(気温,1,[気象情報,場所＝Place,気温＝Temp],
                                Time_tag,[Place,Temp]) :-
    asserta(lhs_match(気温,1,[気象情報,場所＝Place,
                    気温＝Temp],Time_tag,[Place,Temp]).
lhs_call(気温,2,[仕事,名前＝"気温の測定",場所＝Place],
                                Time_tag,[Place,Temp]) :-
    asserta(lhs_match(気温,2,[仕事,名前＝"気温の測定",
                    場所＝Place],Time_tag,[Place,Temp]).
lhs_data(気温,2).
```

これらLHSコンパイルされたプログラムは、作業記憶を変化させる動作（make, modify, remove）から呼ばれ、その副作用として照合に関する情報がアサーションとして保存される。例えば、次のような作業記憶要素を生成する動作は、

make(気象情報(場所＝東京,気温＝13))

次の論理プログラムにおける質問として解釈される。

```
?- lhs_call(Rule,Position,[気象情報,場所＝東京,
                                気温＝13],2.6,_).
```

（ここで、2.6はシステムが決定するタイムタグである。）

また、RHSコンパイルにより次の論理プログラムを得る。

```
rhs(気温,((modify(T,[気温＝compute(Temp＋10)])),
                        [Place,Temp],[(1,T)]).
```

このプログラムの第２引数は、幾つかのRHS要素で構成され、第３引数はLHSに現れる変数のリストであり、第４引数はLHS要素を指定するDesignator(modify等で用いる）の情報を保存する。例えば、次のようなインスタンシエーションが競合解消の結果が得られたなら、このRHSは

instantiation(気温,[2.6,1.5],[東京,13])

（ここで、気温はルール名、[2.6,1.5]はタイムタグの並び出あり、[東京,13]はLHSに現れる変数のリストに対応した変数の束縛情報である。）

?- modify(2.6,[気温＝compute(13＋10)]).

として、実行される。

5. おわりに

表１にVAX11/780上でインプリメントしたKORE/IE(C-Prolog (version 1.4)を用いた)とOPS5(Franz Lisp(Opus 38.79)を用いた)の比較を示す。例題はMonkey and Bananas [5](20ルール(文献中のテスト・ルールは除いた))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)。表１において、OPS5(Compiled)はOPS5インタプリタをLiszt (Franz Lisp Compiler)でコンパイルしたものであり、OPS5 (interpreted)はOPS5インタプリタをそのままFranz Lispにロードしたものである。

表1. 性能比較

	OPS5 (Compiled)	OPS5 (interpreted)	KORE/IE
ルールコンパイル時間(秒)	3.15	23.8	5.75
ルール実行時間(秒)	0.93	53.1	12.88

表１で示すようにProlog上のKORE/IEはLisp上のOPS5(interpreted)に比べ十分に早く、更に高速なProlog(例えば、Quintus Prolog等)を用いればOPS5(Compiled)に匹敵する高速性が期待される。なお、本研究は、第五世代コンピュータ・プロジェクトの一環として行われたものである。

[参考文献]

[1]新谷,他:関係テーブルに基づく推論エンジンKORE/IE, 情報33全国大会,pp.1411-1412(1986).

[2]R.Kowalski: Logic for Problem Solving, North Holland (1979).

[3]竹内,他: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.43-50.

[4]C.L.Forgy: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, AI 19,pp.17-37(1982).

[5]L.Brownston, et al.: Programming expert systems in OPS5, Addison Wesley, pp.388-408(1985).