ＣＬにおける競合解消向ルールコンパイラ

7K-8

山之内　徹＊　松田裕幸＊＊　渡辺正信＊
＊日本電気（株）　＊＊日本電気技術情報システム開発（株）

１．はじめに

ルール指向プログラミングでは、複数のルールの条件部が成功したとき、どのルールを実行すべきかという問題（競合解消問題と呼ぶ）がある。このような問題の解決を含んだルールの処理は、解決すべき問題が大きくなると、実行速度が遅いという問題が生じてくる。　これに対処するため、Reteアルゴリズムに基づいたルールコンパイラが開発されている[1]。

一方、ＣＬのルール指向プログラミング機能は、オブジェクト指向、データ指向等と有機的に統合化するため、ワーキングメモリが構造化されており[2]、文献[1]にあげた方法をそのまま適用してもあまり良い結果は期待できない。　そこで、本論文では、ＣＬにおける競合解消問題を含んだルール指向プログラム[2]を高速に実行するためのルールコンパイラについて述べる。　なお、逐次実行向のルールコンパイラについては別途報告する[3]。

２．ルール表現

図１はＣＬでのルール表現の一例である。　ＩＦ部（ＩＦスロット）には２つの条件節(Condition-1とCondition-2)が、THEN部（THENスロット）には３つの帰結節(Action-1,...,Action-3)がそれぞれ定義されている。　ＣＬのルールは、ＩＦ部の THERE-EXISTS で始まる限定条件節に特徴がある。図１の例では、Condition-1 と Condition-2がこれにあたる。　ここでは、２番目の要素（-Anything と -Parent）がオブジェクト変数、３番目が領域定義、４番目以降がルール述語である。　THERE-EXISTS文は、領域定義中のオブジェクトで、全てのルール述語を成立させるものがある場合に成功し、そのオブジェクトの名前がオブジェクト変数にセットされる。従って，Condition-1 は，“Connected-Parents スロットの値が nilであるSlave-Diskのインディビデュアル(-Anything)が存在する”と読める。

ＣＬのワーキングメモリはオブジェクト指向に基づく汎化階層をもっており、この特徴を有効に利用するため、以下の領域定義が可能である。

① クラス名による指定
　指定されたクラスオブジェクトの特殊オブジェクトの集合が領域となる。

② ＩおよびＩ＊関数による指定
　引数として指定されたクラスオブジェクトの直下だけ（Ｉ），または全て（Ｉ＊）のインスタンスの集合が領域となる。

③ ＴＨＥ関数による指定
　オブジェクトのスロットの値に指定されたオブジェクトの集合が領域となる。

④ ＭＩおよびＭＩ＊関数による指定
　引数として指定されたクラスオブジェクトの集合内の各オブジェクトの直下だけ（Ｉ），または全て（Ｉ＊）のインスタンスの集合が領域となる。

３．ルールコンパイル方式

3.1 ネットワーク構造

ＣＬの競合解消向ルールコンパイラは，ルールセット単位でルールを１つのネットワークに変換し、基本的にはReteアルゴリズムと同様な方法でこのネットワークを処理する。　１ルールの帰結節の実行によって生じるワーキングメモリのオブジェクト単位での変化はオブジェクト生成、オブジェクト削除のトークンとして記述される。　このトークンは、コンパイルされたルールネットワーク中を，上から下へ流され，最後のノードに到達したトークンによって競合集合の変化が求められる。

図２は図１に示したルール１つだけを含むルールセットをコンパイルしたネットワークの例である。まず，Forkノード（Ｆ）に達したトークンは，全ての枝に流される。次に，RegionTestノード（ＲＴ）

```
Slave-R-1
  IF : (Condition-1 Condition-2)
  Condition-1 : (THERE-EXISTS -Anything
                              (I# (quote Slave-Disk))
                              (&= (THE Connected-Parents -Anything) nil))

  Condition-2 : (THERE-EXISTS -Parent
                              (MI# (THE Connectable-Parents -Anything))
                              (&< (THE Used-Port-Number -Parent)
                                  (THE Port-Number -Parent))
                              (&< (THE Used-Slave-Disk -Parent)
                                  (THE Slave-Disk -Parent))
                              (&= (THE TYPE -Parent)
                                  (THE TYPE -Anything)))
  THEN : (Action-1 Action-2 Action-3)
  Action-1 : (&CONNECT -Anything to -Parent)
  Action-2 : (&INCREMENT (quote Used-Slave-Disk) of -Parent)
  Action-3 : (&INCREMENT (quote Used-Port-Number) of -Parent)
```

図１　ＣＬのルール表現例

Rule Compilation for Conflict Resolution Rules in CL
Toru YAMANOUCHI[1], Hiroyuki MATSUDA[2], Masanobu WATANABE[1]
1) NEC Corporation,
2) NEC Scientific Information System Development Co. Ltd.

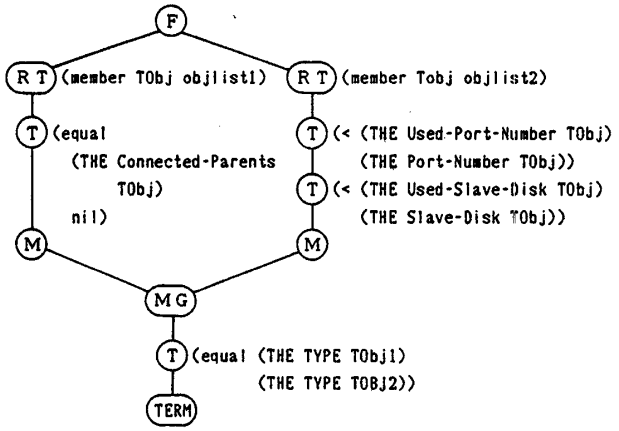

図２　ＣＬのルールネットワークの例

では，変化のあったオブジェクトが領域定義を満たすかどうかのテストが行なわれ、Testノード（Ｔ）ではルール述語のテストが行なわれる。　RegionTestノード、Testノードとも、テストが成功した場合にはトークンを下に流すが、失敗した場合にはその時点で終了する。　Memoryノード（Ｍ）では、流れてきたトークンを保存し，Merge ノード（ＭＧ）ではMemoryノードに保存されていたトークンと新たに流れてきたトークンを合併し，下に流す。　TERMノードは対応するルールの競合集合への登録／削除を行なう。

3.2 RegionTestの処理

前節で述べた４種類の領域定義は、コンパイルされたネットワーク上では、RegionTestノードによって処理される。　これらのうち、③ THE関数、④ＭＩ，ＭＩ＊関数による指定では、領域定義の中で、他の条件節でセットされた変数を用いることが可能である。　図１のルールでは、Condition-2 がこれにあたり、Condition-1 のオブジェクト変数である -Anythingが Condition-2の領域定義（MI＊(THE Connectable-Parents -Anything)）の中で使用されている。　このような領域の指定を間接的指示法と呼ぶ。

間接的指示法による領域のテストは、領域定義の中で使われている変数の値が決まってからでないと行なうことはできない。　このことは、これらの条件節に対応する RegionTest は、 Mergeノードの後でしか行なえないことを意味するが、　一方、Rete アルゴリズムでは、Merge ノードの後のテストが多くなるに従って実行速度が遅くなることが知られている[1]。

そこで本ルールコンパイラでは、コンパイル時に、全ての領域定義をルール実行前の初期ワーキングメモリで評価してしまい、オブジェクトのリストとして保存しておく。　これにより、実行時に RegionTest ノードでは、トークンとして流されてきたオブジェクトがこのリストのメンバかどうかのテストだけを行なえばよく、間接的指示法による RegionTest ノードも他のノードと同様に高速化がはかれる。

この場合、ワーキングメモリが変化し、その結果、間接的指示法により定義された領域が変化すると正しい推論を行なえなくなるが、ワーキングメモリの変化をトークンとしてネットワークに流す際、領域（オブジェクトのリスト）の動的な更新を行ない、対処することができる。

3.2 他のパラダイムとの統合化の方法

ＣＬのルール指向プログラミング機能は、オブジェクト指向、データ指向等のプログラミングパラダイムと有機的に結合しているため、以下の特徴がある。

① ワーキングメモリとなるオブジェクトの汎化階層内に各種のデモンを定義することができ、デモンによるワーキングメモリの参照／更新が可能となる。

② ルールの条件節、帰結節からメッセージを送ることができ、起動されたメソッドによるワーキングメモリの参照／更新が可能となる。

これらの特徴は、ルールをコンパイルした際にも保持される。　これは、デモン、メソッド等がＣＬのルール言語、およびドメイン言語プリミティブによって拡張されたルール言語を用いて定義されるためで、ルール言語処理系内にトークンの作成等ルールコンパイラのための処理を組み入れることによって達成されている。

４．終わりに

本論文では、ＣＬのルールコンパイラのうち、競合解消向のものについて報告した。　このコンパイラは、Reteアルゴリズムを拡張したもので、

① 限定条件節における多様な領域定義を高速に処理することができる、

② デモン、メッセージ送信等、他のプログラミングパラダイムとの有機的な結合を保存することができる、

等の特徴を持つ。

現在、ルールネットワークの実行時処理系を試作し、ハンドコンパイルされたルールを用いて評価を行なっている。　今後は、コンパイラ自身の実現と評価を行なう予定である。

参考文献

[1] Forgy,C.L., “Rete : A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem,” Artificial Intelligence 19,1982, pp17-32.

[2] 渡辺，岩本，山之内，出口，松田，“ＣＬにおけるルール指向プログラミング,” 情処　知識工学と人工知能研究会，46-3，1986.5.

[3] 松田，山之内，渡辺，“ＣＬにおける逐次実行向ルールコンパイラ,”情処第33回全大，1986.10,pp1161-1162.