

CLにおける逐次実行向けルールコンパイラ

7K-7

松田 裕幸* 山之内 徹** 渡辺 正信***

* 日本電気技術情報システム開発(株) **日本電気(株)基本ソフトウェア開発本部

*** 日本電気(株)C&Cシステム研究所

1. はじめに

CL(Conceptual-network-based Language) [1] は知識ベースシステム構築ツールとしてオブジェクト指向、データ指向、論理指向、ルール指向を統合したパラダイムを提供している。エキスパートシステムではルール指向プログラミングが重要なパラダイムとなる。

本論文ではCLにおけるルール指向プログラミングの有効性を効率の面からサポートするルールコンパイラについて述べる。特に、オブジェクト指向によって階層的に表現されたルール、ワーキングメモリへのアクセス方式に重点をおいて論じる。

2. CLにおけるルール指向プログラミング

CLで提供しているルール指向プログラミングの代表的な機能としては次の4つを挙げることができる。これらはオブジェクト指向パラダイムと協調することによって有効なプログラミングを可能とする[2] :

- 1) ワーキングメモリ(知識ベース)とルールベースをオブジェクト(Object)という同一の表現形式で記述する。すなわち、ワーキングメモリ中の知識単位(チャンク)もルールベース中の各条件(condition)、動作(action)もすべてスロットのバリューとして与える。ルールの例を図1に示す:

```
Slave-R-1
IF : (Condition-1 Condition-2)
Condition-1 : (THERE-EXISTS :Anything
               (I* 'Slave-Disk)
               (&= (THE Connected-Parents
                    :Anything) nil))
:
:
THEN : (Action-1 Action-2 Action-3 ...)
Action-1 : (&CONNECT :Anything to :Parent)
:
:
```

図1 ルールの例(文献[2]より引用)

ここで、ルール変数:Anythingには領域関数I*によってSlave-Diskの複数のインスタンスが候補として与えられる。&=は変数:Anythingに対する制約を表わす。

2) ワーキングメモリをオブジェクトとして構造化することによって、ルール変数にマッチする対象(オブジェクト)を限定でき、探索がメモリ全体に広がることを防ぐ。

3) 関連する(例えば、特定の順序で連続して起動するような)ルールをルールセット(これもまたオブジェクトとして定義される)としてまとめることによって、ルール群に対する制御を可能にする。

4) ルール言語としては、条件関数、動作関数、パターンマッチング関数等をユーザが定義、拡張出来るような表現形式を提供している。これは、CLにおけるドメイン言語プリミティブ[1]の思想を反映している。

以下、これらの特徴を生かしつつ、高速な推論機構を実現するためのルールコンパイル方式について述べる。

3. 逐次実行向けルールコンパイラ

CLではルールの適用基準、順序に関して複数の戦略を提供しており、本節では逐次実行向けルールコンパイラについて考える。すなわち、ある基準によってルールを選択する際、一意に選択すべきルールが決定される場合のみを対象とする。なお、適応可能なルールから一つを選択する競合解消型のルールコンパイラについては文献[3]を参照のこと。

今回、前節で挙げた機能に対し、以下の点を考慮した効率の良いルールコンパイル方式を提案する:

- 1) 複数のスロットにまたがる条件、動作記述に対する効率の良いアクセス手段はないか?
- 2) オブジェクトによって階層化したワーキングメモリに対する高速なアクセスをどう実現するか、あるいは一度ルール変数とユニファイしたオブジェクトの再計算を出来るだけ省くにはどうしたらよいのか?
- 3) インタープリタがシミュレートしているルールの実行過程をルールそのもののLisp関数のinline展開に直せないか、さらにはその展開したものの自身のコンパイラもできないだろうか?

4) ルールの表現はCLのルール言語が提供する基本関数の他に、Lisp関数、パターンマッチング関数があり、それらのコンパイルは可能か？

ルールコンパイルの処理は大きくスロットマージとスロットコンパイルに分けることが出来る(図2)。以下、各処理について述べる。

[1] スロットマージ

[1-1] 複数の条件節スロットと動作節スロットをそれぞれ一つのスロットにマージする。これによって、1回のスロットアクセスで条件チェック、動作実行が可能となる。

[1-2] この際、バックトラックを含めた各条件節単位の制御をLisp関数のin-line形式に展開する(いま、ルールセット単位の制御指定は考慮していない)。各条件節の制御はワーキングメモリのその時点での状態に依存するために、予め特定の制御コードを埋め込んでおくことは出来ない。

[1-3] また、ルール変数のスコープ関係に対する考慮、変数に対してユニファイするオブジェクトの集合(値)の再計算の除去を考慮する。これは、一度ルールが発火して、次に発火するまでは、一度計算したオブジェクトの集合の再計算が不要であるからである。

[1-4] さらに、ルールインタープリタが行なっているパターンマッチングの計算を展開コードのなかに効率良く埋め込む。ただし、ワーキングメモリ中のパターンに対応する部分の形式がルールの実行に応じて大きく変化するような場合には注意しなくてはならない。

[2] スロットコンパイル

[1] によってスロットマージされた条件節スロットと動作節スロットはそれぞれLispコンパイラによってコンパイルされCompiled-ConditionsスロットとCompiled-Actionsスロットに格納される。

以上の処理によって、従来ルールインタープリタが各ルールのスロット単位ごとに逐一解釈実行していた過程を、Compiled-ConditionsスロットとCompiled-Actionsスロットのバリュー(コンパイル済の関数)に対する関数コールに置き換えることが出来る。この方式にもとづいて、文献[2]で紹介したルールをハンドコンパイルし、現在CLが提供しているルールインタープリタと比較して、数倍の性能をあげることができている。

4. まとめ

オブジェクト指向パラダイムと結びついたルール指向プログラミングは、自然な知識表現、ルール表現を提供できる。また、オブジェクトのスロットバリューの宣言的解釈(そのインタープリタをユーザが書くことができる)によって、柔軟な知識の表現と、また多彩なルール制御方式を記述できる。これはCLが、フレーム形式に基づくオブジェクト指向概念を知識の表現、ルールの表現双方に採用しているからである。

今回の報告においては、CLのこうした高い記述力がエキスパートシステム構築に際して、有効に発揮できるための一ツールとして逐次実行型のルールコンパイラを提案した。なお、現在ルールコンパイラを作成中である。また、ルールセット、アジェンダを含めた制御に対するコンパイルは今後の課題である。

謝辞

本研究の機会と貴重な助言を頂いたC&Cシステム研究所山本昌弘部長、小池誠彦課長、日本電気技術情報システム開発(株)吉本弘課長に深謝します。

文献

- [1] 渡辺、岩本、出口、"CL:ドメイン言語構築機能を強化したフレーム型知識表現システム、" 情報処理学会 知識工学と人工知能研究会、1985.5.
- [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、" 情報処理学会 知識工学と人工知能研究会、1986.5
- [3] 山之内、松田、渡辺、"CLにおける競合解消向けルールコンパイラ、" 情報処理学会第33回全国大会論文予稿集、1163-1164、1986.10.

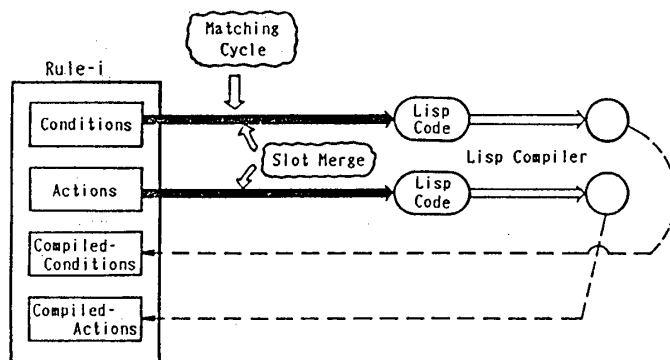


図2. 逐次実行向ルールコンパイル方式