5J-1

Ｐｒｏｌｏｇ高速インタプリタの処理特性

竹内　洋一*　中村　光利**　太田　寛*　松本　茂**　迫田　行介*

*(株)日立製作所システム開発研究所　**(株)日立東北ソフトウエア

１．はじめに

Ｐｒｏｌｏｇ処理系[1]の性能の向上をはかるための高速インタプリタの実行方式を提案する。本高速インタプリタ方式では、木構造の項データを一次元に並べ替えた中間コード（展開コード）をプログラムとして登録し、インタプリタは本コードを解釈実行する。ここでは、本方式の概要と、木構造の項データをプログラムとして利用する従来方式との処理特性の比較結果を示す。

２．従来方式の問題点

Ｐｒｏｌｏｇ言語のプログラムは静的な部分（実行時に変化しない部分）と、動的な部分（実行時に変化する部分）に分けられる。静的な部分が、実行前にマシン語にコンパイルすることにより、高速実行される[2]のに対して、動的な部分のプログラムをマシン語にすることは現実的ではない。動的な部分は、実行時に、項データから生成されるため、頻繁に発生するコンパイル処理が実行性能を低下させるからである。このため、従来、動的な部分は、木構造の項データをほぼ、そのまま、利用していた。

この従来方式の問題点は、実行時に木構造の項データを解釈実行する必要があり、インタプリタの負荷が大きいことである。

３．改良方式の概要

（１）内部データの流れ

本提案方式では、プログラムの登録処理と実行処理とのバランスを考えて、木構造の項データとマシン語の間のレベルの中間コードを設定して、動的な部分を処理する。

本提案方式では**図１**に示すように、木構造の項データを展開コードに変換して、プログラムとして登録し、インタプリタは展開コードを解釈実行する。また、プログラムを参照する際には、元の項データに逆変換する。

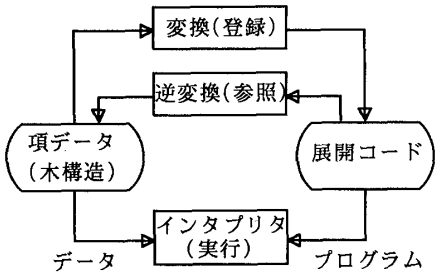

図１　内部データの流れ

（２）展開コード

図２に示すように展開コードは、基本的に、木構造の項データを、実行時のインタプリタの読み出し順序に並べたものである。

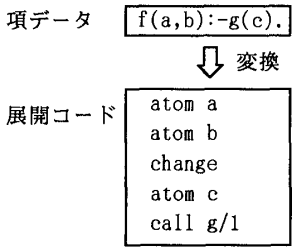

図２　展開コードの生成例

Characteristics of High Performance Prolog Interpreter
Yooichi Takeuchi*, Mitsutoshi Nakamura**, Hiroshi Oota*, Shigeru Matsumoto**, Kousuke Sakoda*
*Systems Development Laboratory, HITACHI, Ltd., **HITACHI Tohoku Software, Ltd.

この種の中間コードとしては、従来、ＷＡＭが提案されている[3]。ＷＡＭとの主な相違点は以下の通りである。

・　ヘッドとゴールの種別を持たない。
・　引数とその展開コードの並び順を一致させ、展開コード中には、引数の番号を記憶しない。

以上のように、展開コードはＷＡＭよりも、木構造の項データに近いコードであるといえる。

４．処理特性の比較

処理特性について、提案方式を従来方式と比較した結果（従来方式に対する比で表わす）について以下に記す。

（１）登録処理

登録処理（assert述語）についての比較結果を以下に記す。

表１　登録処理の比較

プログラム	処理速度比	容量比
CADﾌﾟﾛｸﾞﾗﾑ	1.2	0.47
言語ﾌﾟﾛｾｯｻ	1.3	0.54

提案方式では、処理速度が20～30％向上し、また、プログラムの容量が約50％に減少している。プログラムの容量の低下は、主に、提案方式における展開コードには、木構造を表すためのポインタデータが含まれないことによる。

一方、提案方式では木構造の項データから展開コードへの変換を伴うため、登録の処理速度が向上することは、当初、予期しないことであった。しかし、実際には、従来方式での木構造の項データをコピーする処理と、提案方式での項データから展開コードを生成する処理の内容の相違は小さく、むしろ、展開コードにはポインタデータが不要となることが大きく影響して、処理速度が向上する結果になったものと考える。

（２）実行処理

インタプリタの実行処理についての比較結果を以下に記す。

表２　実行処理の比較

プログラム	処理速度比
CADﾌﾟﾛｸﾞﾗﾑ	3.5
言語ﾌﾟﾛｾｯｻ	3.2

実行速度は、約３倍以上向上した。これは、インタプリタの処理が、展開コードの利用によって、簡易化されたためである。

（３）参照処理

参照処理（listing述語）では展開コードを一旦、木構造の項データに逆変換してから、文字列を生成する方法を採った。この結果、下表のように参照処理（ここでは、画面への出力時間は除いている）の速度は10％近く低下する結果となった。この対策としては、展開コードを直接、文字列に逆変換する方法が考えられる。

表３　参照処理の比較

プログラム	処理速度比
CADﾌﾟﾛｸﾞﾗﾑ	0.93

５．おわりに

Ｐｒｏｌｏｇの中間コード（展開コード）を設定し、そのインタプリタ処理系を提案した。その特性として、従来の木構造の項データを実行するインタプリタと比較して、実行処理の処理速度が３倍以上に向上した。さらに、プログラムの容量が約50％に低減するとともに、登録処理の処理速度も向上した。

６．参考文献

[1] 広瀬、他：論理型言語処理系「LONLI」の開発－実用化機能とその効果－，情報処理学会第31回全国大会 5M-4(1985)

[2] 竹内、他：論理型言語処理系「LONLI」における最適化コンパイル方式の提案と評価，情報処理学会第32回全国大会 3F-6(1986)

[3] 斎藤、他：Warrenの抽象命令セット，日経エレクトロニクス 1987.12.14(no.436),pp. 244-245