高速Common Lisp - HiLISPの実現

2E-1

湯浦 克彦　高田 綾子　青島 利久　安村 通晃
黒須 正明　武市 宣之
((株)日立製作所　中央研究所)

1．はじめに

自然言語解析、設計自動化、知識処理システム等の基本言語としてLispが注目されている。我々は、Lispの共通言語となりつつある Common Lisp[1]に準拠した HiLISP (High Performance List Processor) のインタプリタ／コンパイラの基本方式を設計し、HITAC Mシリーズ計算機上に処理系を作成した。

HiLISPの設計方針としては、compiled-functionの実行速度を向上させることと、インタプリタでの性能も軽視せずに高速性を維持することを取り上げた。Lisp言語では、まず、関数制御の最適化が compiled-function/インタプリタの高速化の要となる。さらに、Common lispでは、多値の導入、インタプリタでの変数管理規則(スコープとエクステント)の厳密化のほかキーワード・パラメータの追加、引数の型の汎用化など機能が拡張されているので、これらの拡張に伴う性能の劣化を防ぐことが大きな課題となった。本稿では、多値の扱いを含む関数制御およびインタプリタでの変数管理について、HiLISPで適用した特徴的な技法と処理系の性能評価結果を報告する。

2．関数制御とくに多値返答の高速化

2．1　関数制御と多値の扱い

Lisp 言語は関数型言語であり、関数制御の速度が compiled-function の限界性能を決める第１の要因となる。

従来Lisp言語では、関数は唯一の値（単値）を返答するのみであったが、Common Lisp では多値を返答する機能が拡張された。関数制御においては関数呼出し側に単値または多値の返答を知らせる処理や、呼出し側が多値を受取った際に多値のうち第１値のみ抽出したりもしくは多値をそのまま受け渡しする制御が新たに必要となり、これらを高速に実現することが大きな課題となった。

2．2　多値実現の諸方式とその問題点

（1）スタックに値を格納する方式(図1(a))
　単値と多値は同型式で格納され受取った側の制御は容易だが、スタック操作の負荷が大きい。

（2）返答レジスタに第１値を格納する方式(図1(b))
　第１値の抽出は自動的になされるが、別レジスタ等に保持する多値を無効にする処理が必要である。

（3）返答レジスタに値を格納する方式(図1(c))
　受取る側で単値／多値を区別しなければならない。

2．3　HiLISPにおける多値経路振り分け方式(図2)

Common Lisp の多値仕様では、末尾呼出し以外では多値を受取ると第１値を抽出し、末尾呼出しでは多値を受取るとそのまま返答する。HiLISPでは、この性質に着目して単値／多値を図１(c)のように同一のレジ

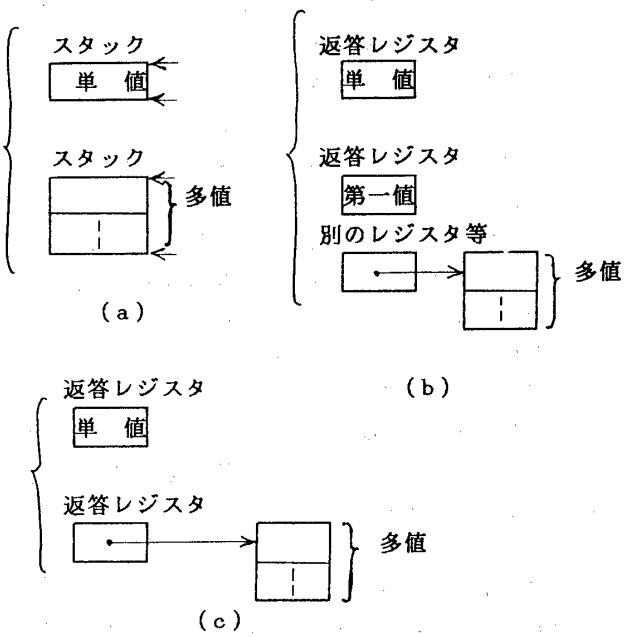

図１　多値保持諸型式

Implementation of HiLISP - high performance common lisp
K. YUURA, A. TAKADA, T. AOSHIMA, M. YASUMURA, M. KUROSU, N. TAKEICHI
HITACHI,Ltd.

スタに保持して、戻りアドレスを分けた。末尾呼出し以外では多値で戻ると第1値を抽出して単値の処理に合流し、末尾呼出しでは、単値で戻ると単値戻りアドレスへ、多値で戻ると多値戻りアドレスへと経路を振り分け、単値／多値の判定なしで制御する。この方式によって通常の単値のみのプログラムの動作にほとんど負荷なく多値の仕様を実現することができた。

3．インタプリタでの変数管理

3．1　静的スコープとスタック結合方式

Common Lispでは、インタプリタにおいてcompiled-functionとの仕様統一のため局所変数に静的スコープを採用している。Lispプログラムは、いくつかのスコープに出たり入ったりしながら動作するため、変数名と値との結合表をスコープ毎に準備するdeep binding方式が、唯一の変数名アドレスにその時点での値を設定する shallow binding方式より有利と考えられた。HiLISPでは、deep binding方式を高速化するため、変数名と値との結合表をスタック上のアレイとして実現するスタック結合方式を採用した。

3．2　無限エクステントと間接ポインタの併用

Common Lisp では、関数引数での例外的な問題を解決するため、静的スコープに加えて局所変数の無限エクステントを明確に定めている。スタック結合方式では、スタック上に変数の結合表を持つため、プログラムの動作に伴ってスタックが上下すると参照不可能になる（動的エクステントになる）問題がある。HiLISPでは、動的エクステント外でも有効としなければならない変数が特殊式functionほかいくつかの式内に限定されるという Common Lisp仕様上の特徴に着目して、特殊式functionほかを処理する時にのみその文脈で参照可能な変数の値をヒープ上へ移動し、クロージャとして保存している。スタックの結合表の元の値の位置には移動先への間接ポインタを設定することにより、スタックを経由する高速の変数参照と、しかる後のクロージャを経由する変数参照を矛盾なくすすめることが可能となった（図3）。

4．性能評価

表1にLispコンテストによるいくつかの例題の速度実験結果を挙げる。今後さらに大規模な評価実験と性能分析を進める予定である。

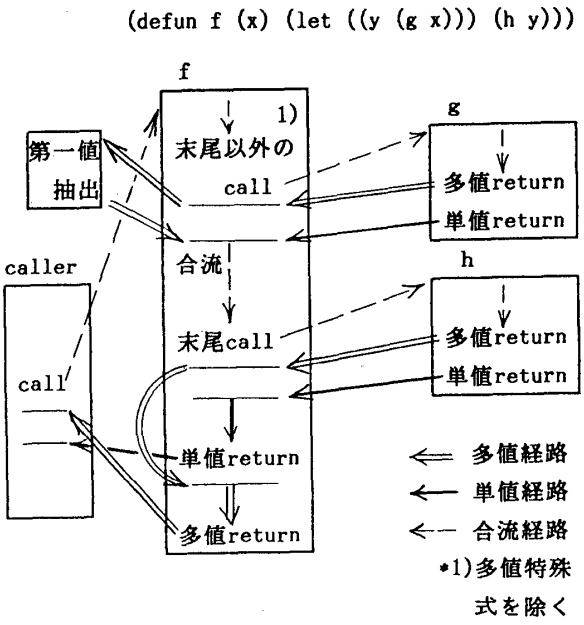

図2　多値経路振り分け方式

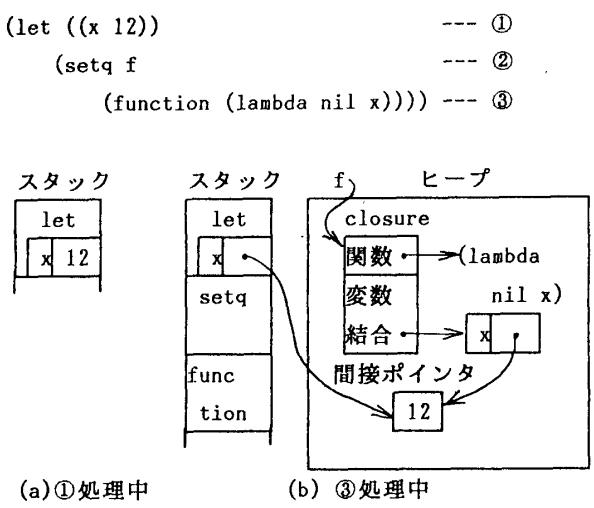

図3　スタック結合と間接ポインタの利用

表1　HiLISP 実行速度 on M280H

	compiled-func	インタプリタ
tarai-5	230 msec	12300 msec
list-tarai-4	46	540
tpu-6	195	2700
prolog-sort	30	580
Boyer	1250	27000

参考文献

1) Guy L. Steele Jr.: "Common Lisp the Language"