記　号　処　理 73－5
（1994．3．18）

PHLの新インタプリタ

寺島　元章

電気通信大学大学院　情報システム学研究科

概要

本稿ではCommon LISPとStandard LISPの機能融合を目指したPHLの仕様とそれに基づく新インタプリタの機能について述べる．PHLの特徴は，可搬性の実現とコンパイラ指向と豊富なデータ型の存在にある．それらのデータオブジェクトは256MBを超えるような大容量記憶空間に配置可能である．こうした機能は多様で大容量データを扱う最近のLisp応用プログラムの処理に効果的である．

Implementation of a new PHL Interpreter

Motoaki Terashima

Graduate School of Information Systems
University of Electro-Communications
1-5-1 Chofugaoka, Chofu-Shi, Tokyo 182 Japan

Abstract

The design of revised PHL and its implementation by a new interpreter are described. The PHL is designed to be a subset of Common LISP and also to be upward compatible to Standard LISP. It is portable and compiler oriented system and is characterized by the existence of many data types. A great amount of storage more than 256 MB can be allocated to their data objects. These PHL features have a good effect on the process of Lisp applications which use various and many data objects.

1 序

数年前から研究室ではPHL(Portable Hashed Lisp)と名付けたリスト処理系を作成してきた. PHLはLispの一方言で, 当初はCommon Lisp [1] (以下, CLと呼ぶ)を意識してその仕様に整合するように設計された. また, 使用計算機が自由に選択できるという可搬性の実現のためにPHL処理系はC言語(gcc)で記述された. 現在使用中のCコンパイラは実に良い目的コードを生成するので, かって筆者も経験したような一度作成したプログラムを対象計算機の機械語で書き直すという事態にはならないようである. そこで, PHLコンパイラも対応するCのプログラムを生成する変換系の役目だけにし, 完全に計算機独立な処理系にする構想で作業が進行中である.

PHLの特徴はコンパイラ指向と豊富なデータ型の存在にある. それらのデータオブジェクトは256MBを超えるような大容量記憶空間に配置可能である. こうした機能は多様で大容量データを扱う最近のLisp応用プログラムの処理を強力に支援する. しかし, PHLはCLとの互換性を重視したことから, HLisp[2]の機能継承やStandard LISP[3](以下, SLISPと呼ぶ)との整合に問題が生じた. とりわけ, 数式処理システムであるREDUCE3[4]を高速に実行することはこれまでのPHLでは不可能である. そこで今回, CLとSLISPの諸機能の融合を目指してPHLの仕様の見直しを行ない, 新仕様のインタプリタをワークステーション(SONY RISC-NEWS)上に作成したので, その設計方針と実装, 評価について以下に述べる.

2 PHLの新仕様

CLの仕様に基づいて, そのサブセットと言うべき新たな言語を作ろうとする試みはKernel Lisp[5]などにも見られるが, PHLの新仕様策定は一言で言うとCLとSLISPの折衷案を作ることである. 具体的には,

1. SLISPで動作するプログラムはPHL処理系で実行可能で, かつ同じ動作をする
2. SLISP非依存で記述されたPHLプログラムはそのままCL処理系でも実行可能で, かつ同じ動作をする

が設計目標である. 1984年に出版されたCLの本の表紙にはStandard LISPの名前があり, CLはSLISPのスーパーセットのように見られるが, 実際にSLISPで記述されたプログラムがすべてCL準拠の処理系で正しく動作するわけではない[1]. たとえば, car と cdr の関数の nil に対する操作の相違や, map関数族の引数の個数とその順序の不一致があり, CLには見られない関数もSLISPにある. そこで, まず, 実行効率と作成の容易さを主眼にCLの豊富な機能を制限する方向で見直しに着手した. 当然のこととして,

1. ほとんど使用されないような機能
2. そのために全体の効率が低下するするような機能
3. 他の機能で代用でき, かつそれで極端に非効率とならない機能
4. C言語で効率良く書けないような機能

などが削除の対象となった. 紙面の制約上そのすべてを詳細に記述することができないので, 主要な点を以下に述べる.

2.1 データ型

CLの機能制約はPHLのデータ型にも反映される. PHLのデータ型は, スカラ型と非スカラ型(構造をもつ型)とに大別される. スカラ型データには, シンボル, 数値, 文字がある. 数値には整数と浮動小数点数がある. シンボルと整数の長さには制限はなく, 表記がすべて意味をもつ. 整数は短整数(絶対値が1億未満の埋め込み即値表現整数)と長整数(bignum), 浮動小数点数は短実数(埋め込み即値表現実数)と任意精度浮動小数点数(bigfloatnum)とからなるが, これは埋め込み即値表現数値を(データ格納領域

[1]このため, REDUCE3の核言語であるSLISPをCL準拠に書き直すプロジェクトがかなり以前から米英で進行中である.

が存在しないアドレスとして) 効率良く具現するための公知の技法である. ただし, 任意精度浮動小数点数の具現は開発途上にあり, 未公開である. 非スカラ型データには, コンス, ベクトル, 配列がある. ベクトルと配列の成分や成分の個数に特別な制約はない. 文字列は (その成分型が文字であるという) ベクトルの一種である. 配列やベクトルはブロックと呼ばれる処理系が提供する構造体として具現される.

結果として, CL が提供する分数や複素数は PHL のデータ型から除かれた. ただし, ブロックの属性としてこの両者を識別するタグは残してあるので, 将来の需要動向によりこれらを見直す (PHL のデータ型に戻す) ことは可能である. これに関連したことでは, reader でのマクロを定義する関数群はないし, 数値入力では基数は固定化される. これにより, 組み込みのマクロ以外は使用できなくなるが, プログラムの静的解析が容易になるとともに, 処理系の入力部の簡素化が図られている.

これらの PHL データ型は SLISP が提供するデータ型を包括する. ただし, SLISP からの要請で特殊記号を含むシンボルの生成に必要な$$記号 (LISP 1.5 [7]) の機能を含めたため, $記号は CL と異なる特殊な扱いを受けることとなった.

2.2 変数

CL の貢献の 1 つに名付きオブジェクトに対するインタプリタとコンパイラの整合を厳格にしたことが挙げられる. たとえば, コンパイラが局所変数を自然な形で静的変数と扱うのに対して, 浅い束縛 (shallow binding) 機構を採用したインタプリタではそれが動的変数のように振舞うのはどう見ても好ましくない. PHL インタプリタは局所変数は静的変数として扱う. しかし次表のように, Lisp は伝統的に種々の性格をもつ変数を有する.

表 1. Lisp 変数の類別

	有効期間 (extent)	
有効範囲 (scope)	動的	無期限
静的	単純局所変数	閉包内変数
無制限	動的変数	大域変数

たとえば,

```
(lambda (x) ((lambda (y) (- x (* y z))) (/ x z)))
```

のラムダ式において, x と y は単純局所変数, z は動的変数か大域変数である. 単純局所変数はその有効期間が関数の呼び出しとそれからの戻りに対応し, インプリメンテーションの見地からはスタック変数と呼べるものである. ただし, 内側のラムダ式の形式に現れる x はそこに限ると単純局所変数ではないが, 構文的にその外側のラムダ式の単純局所変数であることから, これも単純局所変数と言うことにする. なお, こうした「透過的」な単純局所変数の処理については 3.2 節で述べる. これに対して,

```
(defun compose (f g)
    #'(lambda (x) (funcall f (funcall g x))))                    [1]
((lambda (a) (defun f (x) (setq a x))) 0)
```

に現れる f, g, a の各変数は静的な有効範囲をもつが, その有効期間は永続的で, 明らかにスタックの原理に従わないものである. これらを閉包内変数と呼ぶ. こうした変数は閉包 (closure) の生成に関連して作られるからである. 閉包は関数の定義とそれが生成された場所での静的な環境の対である. この環境はデータ格納領域に作られる閉包内に格納される必要があるが, これを自然な形で実現するには環境を連想リスト (a-list)[7] のような仕掛けで最初からデータ格納領域に置くのがよい. 構文解析の伴わないインタプリタでは, 閉包とその閉包内変数の予知とその効果的な対策が事実上不可能であるからである. しかし, こうした配慮は表 2 が示すように処理系の効率を大幅に低下させる. なお, 表中にある a-list 法ではデータ格納領域容量によってはガーベッジコレクション (GC) が起動されるが, 評価時間は GC の実行時間を含まない純計算時間のことである. したがって, 実行時間はさらに長くなることも想定される.

PHL ではその実行効率を向上させるために局所変数はすべてスタックに置かれる. このため, 閉包内変数の出現はそれに対応する値との対を, 閉包タグを持つブロックに移すという付加的な作業を要す

る. 概念的には, スタック中のフレームが保持する環境の一部がデータ格納領域に移動したと考えればよい. これにより, 閉包の扱いに対しての負荷は増大するが, 処理系自体の効率が低下するわけではない. PHL の変数の扱いは SLISP の機能を十分に包括する. なお, SLISP の変数の類別は表 3 で与えられる.

表 2. 評価時間 (秒)

プログラム	PHL	(stack)		HLisp (shallow binding)	
	(a-list)	順 scan	逆 scan	Interpreter	Compiler
(fib 20)	2.09	1.38	1.34	3.12	0.59
(tarai 10 5 0)	37.50	25.45	24.95	40.12	8.44
(tak 18 12 6)	9.00	5.18	4.92	—	—
(tak1 18 12 6)	7.74	4.50	4.39	—	—

表 3. SLISP 変数の類別

	有効期間 (extent)	
有効範囲 (scope)	動的	無期限
静的	local 変数	—
無制限	fluid 変数	global 変数

2.3 関数

データ型, 変数に続いて制御構文について述べる. Lisp では制御構文は, (組み込み) 関数, (組み込み) マクロ, 特殊形式などを用いて記述される. ここではこれらを総称して関数と呼ぶ. 関数の仕様に関しても原則は CL の機能制限である. CL の関数自体を削除することは問題ないが, PHL にある CL と同一名の関数はその機能が CL と同一か, 機能制限としてのサブセットにするという方針を出来る限り貫いた. PHL での動作を CL と同じにするためであり, 結果として, 関数の使用に対しての無用の混乱を回避することができる.

機能制限の例はラムダ式のパラメタに見ることができる. PHL のパラメタ構文には, 位置, 付加 (`&optional`), 余剰 (`&rest`) の 3 種だけが存在する. これは, 各パラメタの予想される使用頻度とその処理効率を考えて, 全体としての効率向上を意図したことによる. 結果として, キーワード (`&key`) パラメタが構文からはなくなったが, その処理は多少の負荷を覚悟すれば, 余剰パラメタを介して行なうことができる. こうした PHL の仕様は, CL が提供する多様なパラメタ[2]を簡素化し, 使用頻度が高いと思われる位置と付加の 2 つのパラメタを含む関数の入口処理を高速化することになる.

一方, キーワードパラメタ処理の高負担はそれが典型的に現れる汎用的な関数の特化を一層推し進めることになる. これは SLISP の `memq` に見られるような専用関数群の復活につながる. コンパイラではこの種の作業は専用関数群の機能に相当する内部処理ルーチンを作り, それらを呼出せばことが済むが, インタプリタではこうした関数を自前で用意する必要がある. そこで, 当面は SLISP を包括する程度の専用関数が入っている. `memq`, `eqn`, `clength` がその例である.

また, SLISP との上位互換性から SLISP 固有の関数が PHL の仕様に含まれている. シンボル操作関数の `explode` と `compress` (MacLISP の `implode` に相当), 入出力関数の `rds` と `wds`, エラー処理関数の `errorset` と `error`, prog 形式 (prog[3]) などである.

map 関数族は仕様として見ると CL と SLISP の融合である. `map` と `mapl` 以外の同一名の map 関数族は両者の仕様を満たす. 処理系では, それらの引数の型検査を行ない, 両者のいずれかの適切な処理が行なわれる. 実行時に型検査の必要があるインタプリタではこれは負担増とはならないが, 当然のこととして, エラーの範囲が狭まるという問題点もある.

CL の継承という点では, 多値に関する仕様は有益な機能として PHL に受け継がれている. 多値機能の効率的な具現法は開発途上にあるが, 3.2 節で述べる単純な方法でもそのために多値を使用しないプログラムの実行効率が大きく変ることはない.

[2]筆者はかって, CL の本 [1] の 81 から 82 ページにあるようなこの種のパラメタ問題を精選して演習で行なったことがあるが, 一種の謎解きのようだと学生には非常に不評であった.

[3]マクロとしてではない.

3 処理系

PHL インタプリタの設計目標の第一はその処理の高速化である．これを端的に言うと，多少の親切さを犠牲にしても実行効率を上げることである．これは，インタプリタの実行効率は二次的なものでプログラム作成の効率向上が図られるべきであるという既成概念と対比する．trace の機能や引数の型検査はあるが，引数の個数の検査やエラー発生後のバックトレースの情報は通常[4]表示しないなどの徹底した効率化が図られている．この背景には，エラーとなる場合も含めて解をより早く得たいことがある．昔そうであったように計算機が解を求めるのに長い時間を要するならば，そのエラー情報は貴重で出来る限り集めるのが得策であった．しかし，高機能ワークステーションが占有使用できるという計算機環境では再試行は大した負担にはならない．現に，このインタプリタも比較的短期間にいくつかのプロトタイプ (プログラムの大幅な書き直し) を経て作成されたものである．その評価に用いたのは，Tarai 関数，TPU, bit などのプログラム [8] である．また，高速なインタプリタはコンパイラが順調に稼働するまで実用的な処理系として十分に機能するという利点もある．

3.1 データの表現

PHL のデータは 1 語 (32 ビット) を用いて表現される．1 語はタグ部とアドレス (データ) 部とに分けられる．PHL で採用したタグ方式はいわゆるポインタタグと呼ばれるもので，そのデータを指す側 (ポインタ) がその型を識別する情報をもつ．各語の最上位ビット (MSB) は *value bit* とも呼ばれる．CONS データ，シンボル，ブロック，長整数は MSB=0 で表す．この場合，タグは 3 ビット長で，000(CONS データ)，001(シンボル)，010(ブロック)，011(長整数と任意精度浮動小数点数) と割当てられる．残りの 29 ビットがアドレス部で，データ格納領域上に置かれた成分 (データ) の先頭 (語) を指すポインタが入る．

現在主流となったバイトアドレシングの採用を前提にすると，アドレス部の下位 2 ビットは 00 で他には使用されない．これは GC での印付け (marking) で活用される．現処理系ではアドレス部の先頭ビットはすべて 1 である．これはヒープ領域がこれをベースに始まることによるもので，このためデータ格納領域として使用できる容量は 28 ビット分の 256KB になる[5]．当然のこととして，CONS データの参照はマスキング演算なしに (高速に) 行なうことができる．ベクトルや配列の属性 (型や成分の個数など) はブロックの先頭の語 (block-header) が保持する．これはいわばデータタグ的な扱いとなるが，これらの型は他の型ほど頻繁に参照されることはないので，全体としての実行速度にさほどの影響を与えないとの考慮である．文字列の終端は C 言語仕様に合せて零の値が置かれる．

シンボルは，値，関数定義，属性リストと文字列のための領域から構成される．また必要に応じてシステム用の 1 語が確保される．

即値表現数値と文字値のデータは MSB=1 で表す．この場合，タグは 4 ビットで，MSB と次の印付けビット (*mark bit*) とを除いた残りの 2 ビットで，短実数 (00)，短整数 (01)，文字 (10) を識別する．1 語の残りの 28 ビットはデータ部と呼ばれる．データ部には，即値表現と言われるように値が直に置かれる．

表 4. タグ表現 (上位 4 ビット)

0001	CONS データ
0011	シンボル
0101	ブロック
0111	長数値
1000	短実数
1001	短整数
1010	文字符号
1011	間接ポインタ (未使用)

ブロックヘッダ

1000	識別タグ	ブロック長

シンボル

大域値
関数定義
属性リスト
システム (必要時確保)
1010 7 bit 文字符号
...
1010 ... 0000000

[4]こうした検査や back-trace の表示を行なうための処理系の変更は C(gcc) のマクロプリプロセッサに対する指示という形になっている．

[5]正しくは，gcc の malloc 関数の返す値が 0x10000000 のゲタをはいていることによる．また，後述するスタックもヒープ領域に確保されることから，データ格納領域容量は厳密には 256MB 未満である．

このような表現を採用することで, 以下に述べるような利点が得られる.

1. 整数はその絶対値が1億未満については即値表現 (短整数) となり, 記憶を消費しない. また, 同値も eq で検査できる. 長整数は1億が基数 (radix) となり, これは日本式桁表記に好都合である.

2. 実数は精度が20ビット以内の単精度浮動小数点数の範囲にあれば, 即値表現 (短実数) となる. 記憶を消費しないで済むし, 同値も eq で検査できる. これは CL の埋め込み即値表現実数よりも精度は高い. 短実数は数式処理に広く含まれる実数計算を強力に支援する. これは仮数部22ビットの (H-lisp の) 短実数で動作させた REDUCE[4] からの教訓である.

3. 文字は ASCII 7 bits 符号であれば, 4文字まで (1語に) 詰めることができる. また漢字も使用できる.

要約すれば, PHL のデータ表現は256MB という大きな記憶空間にコンスデータや配列, ベクトル, シンボルなどの配置を可能にし, 即値表現数値や文字についても各語への効果的な詰込みを可能にしていると言える.

3.2 記憶配置

PHL のデータオブジェクトはデータ格納領域とスタック領域に置かれる. データ格納領域に置かれたオブジェクトは GC の対象になる. PHL 処理系の GC は圧縮方式 [9] を採用しており, ブロックを含む可変容量オブジェクトの効率的な管理を実現している. スタック領域はその両端からユーザスタックと制御スタックが伸びる. 前者は関数の引数の引渡しや変数束縛, (多) 値の返却, 関数実行時の作業用に使用される. GC 時の根 (root) とすべきデータオブジェクトはすべてユーザスタックに積む必要がある[6]. こうした領域は一般にフレームと呼ばれるが, PHL のフレームは次のような特徴をもつ.

1. インタプリタが解釈実行する関数[7]では引数値とそれに対応するシンボルが隣接する (図1参照). 機能的には a-list 構造がスタック上に展開されたと考えればよい.

2. 動的変数のシンボルとその束縛値の対も同じフレーム内に置かれる. この種の環境は制御スタックと呼ばれる別のスタックにとるのが一般的な技法であるが, 単純局所変数と動的変数の値を同じフレーム内に格納することによって, この両者の値の参照は同じ手続きで行なうことができる. また, 制御スタックが簡素化され, 関数の戻りでの操作が通常において不要となる. 欠点は, 動的変数の参照に要する線形コストである. また, パラメタの位置関係によってはフレーム内でシンボルと値の対を作るときに入れ替えが起きることがある. たとえば,

   ```
   (defvar a)
   (defun good (a x y) ... )
   ```

   ならば動的変数が先に来ているのでこのままの位置でフレームに格納されるが,

   ```
   (defvar a)
   (defun bad (x y a) ... )
   ```

   であると, フレームに a, x, y の順に格納されるように入れ替えが生じる.

3. 各フレーム (またはその一部) は必要に応じて連鎖状になる. これは, 動的変数と透過的な単純局所変数の参照に複数のフレームをまたぐことが必要であることによる.

4. フレーム内走査はパラメタ表記とは逆の順序で (後から) 行なわれる. フレームが連鎖あるいは結合される場合, その走査はパラメタ記述とは逆の順で行なう方が効率的である. このことは表2の実行時間からも言える. このため, 同一名の位置パラメタはその最後に登場するものと対応する値がそのパラメタ (変数) の値となる[8]. たとえば,

---

[6] GC がデータオブジェクトの再配置を行なうような処理系ではこれをさぼるといわゆる懸垂問題 (dangling problem) という厄介なバグを引き起こす.

[7] SLISP の de や df, dm で定義された関数のこと

[8] この仕様は LISP 1.5 とは異なる.

((lambda (x y x) (list x y x)) 1 2 3) = (3 2 3)
≠ (1 2 1)

である. しかし, こうしたトラブルを回避する目的でパラメタの名前の検査をしてもその負荷はインタプリタの実行速度にほとんど影響を与えない. また,

(defun foo (x y) (prog (z) ...))

のようなREDUCE に頻繁に登場する関数では, x, y と z は同一のフレームに置くことができる. これは, x と y の探索時間を短縮化につながる.

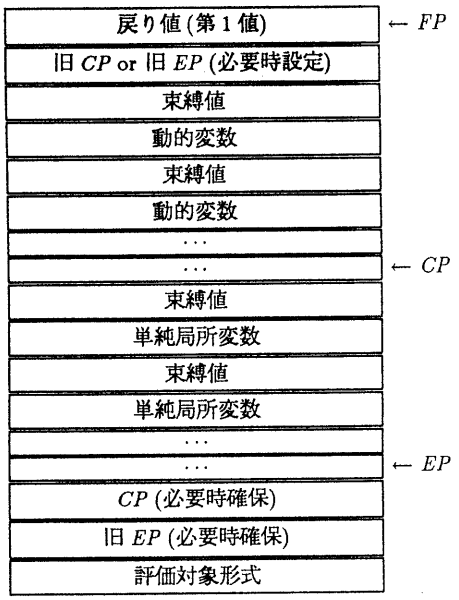
PHL を記述する C 言語も関数の呼出しでスタックを使用する. これはシステムスタックと呼ばれるものであるが, PHL の可搬性を実現するためにこのスタックには手を付けない. そこに積まれるデータは引数である FP や EP などのフレームポインタ, 関数からの戻り番地やレジスタ値である. システムスタックはその機能から隠れた (implicit な) 制御スタックと見なすことができる. しかし, こうしたデータが必要になるエラー処理や大域的出口の処理のためにその一部が PHL の制御スタックに積まれることになる.

4 まとめ

本稿では CL と SLISP の機能融合を目指した PHL の仕様とそれに基づく新インタプリタの機能とその効率化について述べた. 画期的な処理速度の向上は PHL コンパイラの完成を待たなければならないが, 新インタプリタでも実用として十分に機能するという感触を得ている. しかし, 設計にあたり, 特殊と思われる機能は他の機能に比べて冷遇してあるので, こうした機能を使う人はそれなりの覚悟で使って欲しいというのも実感である.

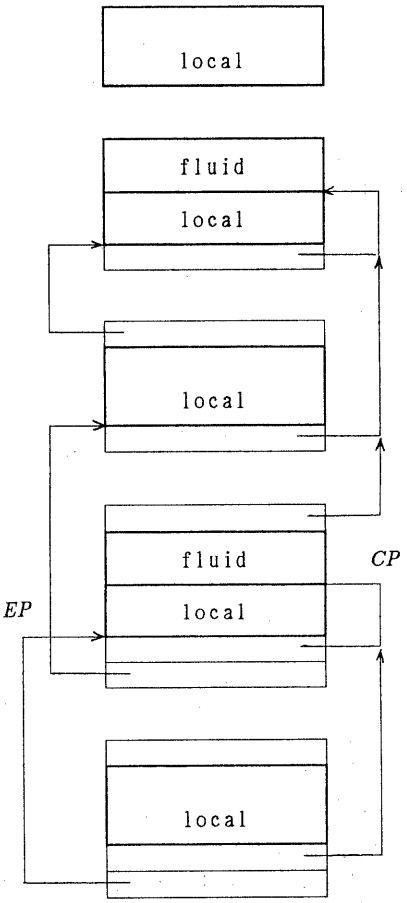
5 参考文献

1. Steele, G. L. Jr. : *Common LISP, (2nd ed)*, Digital Press, MA. (1990).
2. Terashima, M. and Kanada, Y. : HLisp – Its concept, implementation and applications, Journal of Information Processing 13 (3) pp. 154–160 (1990).
3. Marti, J. et. al. : STANDARD LISP REPORT, SIGPLAN Notices, 14 (10) pp.48–68 (1979).
4. Hearn, A. C. : REDUCE User's Manual (Ver. 3.3), The Rand Corporation, CA. (1990).
5. 湯浅太一他 : LISP 国際標準化のためのベース言語 KL について, 記号処理研究会資料, SYM 65 pp. 27–34 (1992).
6. Gries, M. L. et al. : PSL: A Portable LISP System, Proc. of 1982 LISP and Functional Programming (ACM), pp.88–97 (1982).
7. McCarthy, J. et al. : *LISP 1.5 Programmer's Manual*, MIT Press, MA. (1962).
8. 竹内郁雄 : LISP 処理系コンテストの結果, 記号処理研究会資料, SYM 5–3 (1978).
9. 寺島元章 : 古典的ガーベッジコレクションからの話題, 記号処理研究会資料, SYM 68–5, pp.31–38 (1993).



記号
FP : フレームポインタ
CP : フレーム内 common 領域
EP : フレーム内 local 領域

図 1　フレームの構成



fluid ：動的変数

local ：単純局所変数