# Tachyon Common Lisp におけるプリティプリント

5F−10

小島 透* 五味 弘* 高橋 順一* 長坂 篤**

*(株)沖テクノシステムズ ラボラトリ
**沖電気工業(株)

## 1 はじめに

Common Lisp 第2版[1]では、プリティプリント制御関数を Common Lisp 標準関数として定義することで、第1版で不明確であったプリティプリント機能の仕様を明確にした。またユーザが自由に表示関数を定義できるように拡張が行われた。

本稿では、ワークステーション OKIStation7300 上に開発した第2版準拠の Tachyon Common Lisp[2]におけるプリティプリント機能の実現方法と、第2版の仕様の拡張の提案を行う。

## 2 第2版プリティプリントの概要

第2版プリティプリント機能は、第1版を機能拡張した XP[3]をベースにしており、以下の特徴を持っている。

1. 見やすさを損わない限り、改行を抑えて一行に表示する
2. 一行を超える表示になることを事前に防ぐ(MISER スタイル)
3. 同じ意味を持つ部分については、同じ字下げを行う
4. リスト以外のオブジェクトに対しても利用できる

処理の流れは大きく二つに分けられる。

まず、表示するオブジェクトが、ディスパッチ表に登録されているタイプのサブタイプであるか調べる。そして当てはまるタイプの中で、一番優先順位の高いものに対応する関数をこのオブジェクトのプリティプリント関数と決める。

次にプリティプリント関数では、論理ブロックを単位としてセクション間の条件付き改行一つ一つについて改行するしないを決定し、設定に従った字下げを行う。

便宜上、前者をディスパッチ部、後者を整形出力部と呼ぶ。

## 3 ディスパッチ部の高速化

ディスパッチによるオーバーヘッドを減らすには、オブジェクトごとのタイプチェックをいかに高速に行うかがポイントとなる。これは特に副構造を持つオブジェクトでは、それぞれの要素について再帰的にディスパッチする必要があるため重要である。XP では、(cons (member *symbol*)) のタイプと構造体タイプに対しては、それぞれハッシュテーブルに登録することを勧めている。

Tachyon Common Lisp では更にハッシュテーブルに登録されないタイプに対して効率を上げる工夫を行った。関数 GET-PREDICATE はタイプを引数に取りそのタイプに対応する述語があればそれを返す。

```
(GET-PREDICATE 'CONS)   → CONSP
(GET-PREDICATE 'VECTOR) → VECTORP
```

これを用いて関数登録時に述語があるタイプに対してはその述語をタイプの代りに登録することで実行速度を上げた。第2版にはベクタに対してのプリティプリント関数の例が載っている。このようなものに対してのプリティプリント関数が定義できるのは第2版の優れた点であり、これらのタイプには Common Lisp 関数で述語が存在する場合が多い。ハッシュテーブルに登録していないタイプが増えた時、特にこの方法は有効である。

表示部はタイプによる分岐を行う構造となっている。そのため分岐処理を一部省いたディスパッチ関数を用いた方が効率は良い。Tachyon Common Lisp では、リスト、構造体、その他のタイプの3つの専用のディスパッチ関数を用意し、システム内部ではそれを用いることにした。

## 4 整形出力部の効率向上

整形出力部を設計するにあたって注意したのは次の二点であった。

1. 整形情報のアクセスを高速にする
2. 実行時のメモリ使用量を極力少なくする

そのため3つの配列を用いて、整形情報を系統的に分類するのが良いと考えた。

Tachyon Common Lisp は、全ての文字を2バイトで扱うため、日本語との相性が非常に良い。しかしそのため文字列の使用バイト数をそのまま表示幅と見ることはできない。また PPRINT-TAB や TAB コードを含んだセクションの表示幅は表示カラムが判らないと決定できない。この正しい表示幅を決定する作業を減らすため、論理ブロックに含まれるセクションの情報と

---

Pretty Print in Tachyon Common Lisp
Tohru KOJIMA, Hiroshi GOMI, Junichi TAKAHASHI*
Atsushi NAGASAKA**
*Oki Technosystems Laboratory, Inc.
**Oki Electric Industry Co., Ltd.

して論理ブロックに入った時に、表示幅、タブの有無、非条件付き改行の有無を一つの配列に蓄えた。これによって論理ブロック内での整形出力の計算を効率良く行なえた。

論理ブロックの開始終了と条件付き改行の情報を、それぞれ前置後置文字列、字下げモード、字下げ数とともに一つの配列に蓄える。これによって論理ブロックを超えた処理をしやすくした。

PPRINT-TABによって指定されたタブも配列に蓄えた。特にこの配列から必要なタブの情報を取り出す時は、順次取り出す事が多いが、同じ所であったり、飛んだり戻ったりすることがある。このために一度取り出した時の位置を覚えておき、次からはその位置から検索することで高速にタブの情報を取り出せた。

またこれら3つの配列だけを利用することで、プリティプリントとしての実行時のメモリ使用量は、情報の量がそれぞれの配列を超えた時を除いてほぼ零になった。

## 5 第2版の仕様の拡張

我々は、第2版の仕様に対して以下の二つを提案する。

MISERスタイルの条件付き改行は、残りの幅が一定以下になったとき改行を行う。しかし、そのブロックに含まれるセクションは全て左づめになってしまうため、余り見映えが良くない。また条件に合うまでは改行を抑えた出力を行っているため、既に手後れで結局一行を超える表示になってしまうことがある。

一つ目の提案は、残りの幅が一定以下になったとき改行を行うが、全て左づめにせず、ただそこだけで改行を行うMISERの閾値を設けることである。この閾値を普通の閾値より大きめに取ることで、表示幅と表示行数の節約になる。残りの表示幅によって表示結果が変化する例を以下に示す。

```
0----+----1----+----2----+----3----+----
          |          <---- MISERの閾値 ----
          |<---- 提案するもう一つの閾値 ----
          |          |
(MULTIPLE-VALUE-BIND (X Y Z)
                     (DECODE-FLOAT F)
   データ1           |
   データ2 ..)       |                - 1 -
          |          |
          |          |
          |  (MULTIPLE-VALUE-BIND
          |       (X Y Z) (DECODE-FLOAT F)
          |     データ1
          |     データ2 ..)          - 2 -
          |          |
          |          |
          |          |  (MULTIPLE-VALUE-BIND
          |          |   (X Y Z)
          |          |   (DECODE-FLOAT F)
          |          |   データ1
          |          |   データ2 ..)   - 3 -
          |          |
          |          .
```

新しい閾値により、二番目の表示を行うことが可能となる。なおその時には他の条件付き改行と同じように字下げの指定に従う。

次に、同じ条件の要素が数多くある時、タブを利用すると非常に見やすくなる。しかしタブサイズは固定的に指定するため、現状ではフレキシブルな対応ができず、もし要素の表示幅がタブサイズより大きなものがあると、次のように見づらいものになってしまう。

```
April   May      June     July     August
September        October  November
December
```

二つ目の提案は、要素の最大幅に従ってフレキシブルにタブサイズを変える機能を付けることである。プリティプリントでは表示幅を調べることは必要であるため、この機能を付けることは可能である。タブサイズを要素の最大幅＋1とすれば以下のような出力になり、見やすい上に一目で要素の数が分る。多少場所を取る可能性があるが、要素幅を考えた上でタブサイズを自分で指定する必要のない、この機能はあると便利と考えられる。

```
April    May       June      July
August   September October   November
December
```

## 6 おわりに

実行時のメモリ使用量を減らすことに重点を置いたため、メモリの静的領域が多い、今後は静的領域を減らすことが課題である。

## 参考文献


[1] Guy L. Steele Jr., "COMMON LISP THE LANGUAGE Second Edition", Digital Press 1990.

[2] 大江 哲男他, "OKI Common Lisp の開発", 情報処理学会第43回全国大会, 5L−2.

[3] Waters, Richard C., "XP:A Common Lisp Pretty Printing System", AI Memo 1102. MIT Artificial Intelligence Laboratory, March 1989.


```
(DEFUN PPRINT-LET (S LIST)
  (PPRINT-LOGICAL-BLOCK (S LIST :PREFIX "(" :SUFFIX ")")
    (WRITE (PPRINT-POP) :STREAM S)
    (PPRINT-EXIT-IF-LIST-EXHAUSTED)
    (WRITE-CHAR #\SPACE S)
    (PPRINT-LOGICAL-BLOCK (S (PPRINT-POP) :PREFIX "(" :SUFFIX ")")
      (PPRINT-EXIT-IF-LIST-EXHAUSTED)
      (LOOP (PPRINT-LOGICAL-BLOCK (S (PPRINT-POP) :PREFIX "(" :SUFFIX ")")
              (PPRINT-EXIT-IF-LIST-EXHAUSTED)
              (LOOP (WRITE (PPRINT-POP) :STREAM S)
                    (PPRINT-EXIT-IF-LIST-EXHAUSTED)
                    (WRITE-CHAR #\SPACE S)
                    (PPRINT-NEWLINE :LINEAR S)))
            (PPRINT-EXIT-IF-LIST-EXHAUSTED)
            (WRITE-CHAR #\SPACE S)
            (PPRINT-NEWLINE :FILL S)))
    (PPRINT-INDENT :BLOCK 1 S)
    (LOOP (PPRINT-EXIT-IF-LIST-EXHAUSTED)
          (WRITE-CHAR #\SPACE S)
          (PPRINT-NEWLINE :LINEAR S)
          (WRITE (PPRINT-POP) :STREAM S))))
```

プリティプリント表示例