8E−4 Tachyon Common Lisp − 標準化と拡張性 −

堀川 恵美* 新谷 義弘* 長坂 篤* 高橋 順一** 五味 弘**

*沖電気工業(株) 総合システム研究所 **(株)沖テクノシステムズ ラボラトリ

1 はじめに

Lisp 言語は, その誕生以来, 言語の持つ柔軟性/拡張性により多くの方言が存在したが, 仕様の共通化や可搬性などを目的とした Common Lisp [1](以下, CLtL1 という)の登場により統一が進み, Common Lisp は業界標準として定着した. その後, 第1版の持つ言語仕様の不完全な部分を修正し, ANSI 規格とするための作業が ANSI X3J13 において進められた. この作業の成果をを反映した形で, オブジェクト指向機能(MOP を除く CLOS)等を付け加えた Common Lisp 第2版 [2] (以下, CLtL2 という) が出されたが, これはまた, ANSI Common Lisp[3](以下, ANSI CL という)へのつなぎ役もかねていた. 現在, ANSI CL は, public review が終了し, 早ければ 今春にも制定される予定である.

CLtL2 と ANSI CL では, 残念ながら仕様が異なる点がある. Tachyon Common Lisp[6]は, CLtL2 に基づいた処理系であり, ANSI CL に基づく仕様に変更する必要があるが, ANSI CL の制定時期に不明確な所があり, また, CLtL2 で書かれたプログラムの継承のためにも当面2つの仕様を両立させておくべきだと考えている.

本稿では, この新しい ANSI CL 仕様と CLtL2 仕様との言語仕様の違いに対処するために採用した方法および このために新たに開発したコンパイラの拡張方式について述べる.

2 CLtL2 と ANSI CL

ANSI CL 規格作成の段階で議論された CLtL1 における仕様の詳細化および修正の多くは既に CLtL2 に反映されているが, CLtL2 と ANSI CL との言語仕様の差異は大きく以下のクラスに分けられる.

1. CLtL2 と ANSI CL とが同じ仕様
   ほとんどの関数がこのクラスに属する
2. CLtL2 と ANSI CL とが異なる仕様
   このクラスには次のものがある.
   - インタプリタ実行時の変数の型検査
     ANSI CL では, 行なわなくても良くなった.
   - 文字の型指定子の名前の変更
   - 引数の追加
     defsetf, subtypep, typep, constantp, upgraded-complex-part-type, compile-file 等
   - 戻り値の変更
     rename-package, compile-file, read-line 等
   - ドキュメンテーションまたは宣言の追加
     defpackage, with-hash-table-iterator, pprint-logical-block 等
   - CLOS のかなりの部分
     generic-function 等の追加, 削除. 仕様の明確化等
3. CLtL2 では未定義, ANSI CL では定義
   - 新規関数
     copy-structure, floating-point-invalid-operation, floating-point-inexact, array-displacement 等
   - 宣言子 ignorable
4. CLtL2 では定義, ANSI CL では削除
   special-form-p(special-operator-p), define-setf-method(define-setf-expander), get-setf-method(get-setf-expansion), get-setf-method(get-setf-expansion), 等名称が変更になったもの
5. CLtL2 では仕様が曖昧, ANSI CL で詳細化
   - エラー時のエラーの型
   - 列を含むリスト関連関数の proper リストの検査が厳しくなった

以上から, CLtL2 処理系での ANSI CL 仕様への対処では, CLtL2 で仕様が曖昧だったものは, ANSI CL の仕様で定義し, 新規に登場した関数や名称の変更になった関数は, CLtL2 を破壊することはないので, 問題となるのは, CLtL2 と ANSI CL で仕様が異なるものについての対処だけである.

3 ANSI CL への対応方法

2つの異なる仕様に対応するもっとも簡単な方法は, それぞれの仕様に対応するファイルにすることである. しかしながら, 同じ関数を含んだファイルを2つ作成してしまうと, 保守性や開発能率の上で問題となる. 本処理系では, CLtL2 と ANSI CL では, 仕様が異なる点があると言っても全体から見ればほんの一部であり, また, 致命的な仕様の違いがないことに注目し, ファイルは一元管理し, *features* で切り分けることにした.

前節であげた主な点について対応方法を述べると, 以下のようになる.

1. インタプリタ実行時の変数の型検査
   該当部分がアセンブリコードなので, CLtL2/ANSL CL のいずれかを示すシステムモードビットによる切替えを行なう. これによオーバーヘッドは, 型宣言がある場合に2〜4ステップである.
2. 文字の型指定子
   base-character が, base-char になったものである.
   場合分けを*features*によっておこなう.
3. オプショナル引数など引数の追加
   場合分けを*features*によっておこなう.
4. 戻り値が異なる関数がある
   場合分けを*features*によっておこなう.

Tachyon Common Lisp − Standarization and Extension
Emi Horikawa*, Yoshihiro Shintani*, Atsushi Nagasaka*
Junichi Takahashi** and Hiroshi Gomi**
*Oki Electric Industry Co, Ltd., Systems Laboratories
**Oki Technosystems Laboratory, Inc.

5. CLOS のかなりの部分
   CLtL2 が出た時点では, 正確に仕様が規定されたわけではなかったので, 当初は PCL ベースとし, 本格的な CLOS の開発は ANSI CL 版において規定された仕様で作成の予定であった. したがって CLOS については, ANSI CL 仕様のみとなる.

以上のように, 本処理系では CLtL2 と ANSI CL の仕様の違いに対して, 次のような方法で対処した.

1. S式については, *features* 機能を用いて場合分けを行なう.
2. アセンブラ部分で, CltL2 としてもその機能が欲しいものは, システムビット等を利用して実行時に分岐する.
3. アセンブラ部分については, make 時にその切り分けを行なう. ただし, 今回はこの対処に該当するものはなかった.

4 コンパイラの拡張性

4.1 コンパイラの制御構造

Tachyon Common Lisp の コンパイラ [5]は, 変数や関数の引数及び戻り値の型情報をもとに最適化を行なっている. 例えば, 入力引数の型により最適関数へと展開する. つまり, 入力引数がリストだとリスト専用関数へと展開する. このため, コンパイラは, CLtL2 に定義されている関数の引数及び戻り値の型情報をデータベース化している.

CLtL2 と ANSI CL で引数や戻り値が異なる場合は, このデータベースを変更する必要がある. これは, データベース生成時に *features* で切り分けても良いし, 生成後に明示的に変更しても良い.

さらに, このデータベースは, コンパイラの各フェーズにおける呼びだし関数も格納している. ここでコンパイラの各フェーズとは, コンパイラの 7 つのフェーズのうちの次のフェーズである.

1. 前処理
   マクロ展開等
2. 変数解析及び文法エラーチェックフェーズ
3. 上位中間言語変換フェーズ
4. 中間言語の生成
5. アセンブラレベルのオープンコード化

Tachyon Common Lisp のコンパイラは,

1. 省メモリ化
2. 高速性
3. fixnum(有限整数) 優先

を考えて設計されているが, ユーザの要求には「メモリを使っても良いから高速に実行させたい」, 「速度は遅くてもメモリを使うな」等, 処理系の方針とは 矛盾する場合も出てくる. ある程度は optimize などの宣言で対応できるが最終的には処理系作成者のポリシーになる. これは, ユーザとは異なる意見になることもありうる. また, ユーザが追加したいコードやコンパイラの生成するコードより効率の良いコードがあるかも知れない.

これらの問題や仕様の拡張・変更に対応するため, データベースは, コンパイラの各フェーズにおける呼びだし関数も格納することにした.

4.2 コンパイラの拡張

ここでは, もっとも利用されるであろうアセンブリ言語レベルのオープンコード化について説明する. 例えば, (numberp x) というS式がコンパイルされると,

```
(move nil <X> (in 0))          ;; X を入力レジスタ 0 へ
(call-to-system  nil numberp)
```

という中間言語に変換される. これをアセンブリコードに変換するのであるが, データベースに numberp をオープンコードにする関数を入れておくと(call-to-system nil numberp) を変換しようとする時に自動的にそのオープンコード化関数が呼び出される. データベースになければ, 通常, eval 経由の関数呼び出しになる.

本処理系は, 最適化のレベルによっては, numberp をオープンコード化する. その場合,

```
and  #b111110 (in 0) (in 0)
br-zero return-true                ;; fixnum
....
xor #b011000 (in 0) *0-value-reg*
br-zero return-true                ;; double-float
mov  *nil-value-reg* (in 0)        ;; return nil
```

というように fixnum 優先のコードに展開される. ところが, ユーザにとっては, もっとも頻繁に使用される数値は, 最後の double-float であった場合には, double-float 優先のオープンコードであった方が高速になる. この場合, double-float 優先のオープンコードを生成するような numberp のオープンコード化関数をデータベースに登録すれば, その関数でオープンコード化されるようになる.

5 おわりに

Tachyon Common Lisp における ANSI CL 規格に対応する方法および言語仕様変更やユーザの要求に対応するためのコンパイラの拡張方式を示した. 本稿の執筆時点においては, ANSI CL が正式に何時制定されるかはわからないので, 過渡期においての対処としては, これが最適と考える.

コンパイラの拡張性は, ANSI CL 等の新しい仕様に対するだけでなく, ユーザにとって効率の良いコードを生成するためにも非常に有効であろう.

参考文献

[1] Guy L. Steele Jr.: "Common Lisp: the Language", Digital Press, 1984

[2] Guy L. Steele Jr.: "Common Lisp the Language Second Edition", Digital Press, 1990

[3] draft proposed American National Standard for Information Systems-Programing Language-Common LISP. X3J13/92/102

[4] 五味他: "Tachyon Common Lisp の実現方式", 記号処理研究会 92-SYM-64-3, 情報処理学会. 1992

[5] 新谷他: "Tachyon Common Lisp コンパイラの高速化方式", 記号処理研究会 92-SYM-64-4. 情報処理学会, 1992

[6] 長坂他: "Tachyon Common Lisp: An Efficient and Portable Implementation of CLtL2", *Proceedings* of the 1992 ACM Conference on Lisp and Functional Programming, ACM, 1992.