5B-2 Common Lisp による ファジィ集合処理システムの実現

久米 健司 馬野 元秀
(㈱新学社 電子出版事業本部) (大阪大学 大型計算機センター)

1. はじめに

Lisp に ファジィ集合を取り扱うための機能を追加したファジィ集合処理システムを、すでに Franz Lisp (UNIX 4 BSD 上)、 UTI-Lisp (ACOS-6/MVX 上)、muLisp-86 (MS-DOS 上) で作成している。このシステムを作成するにあたって主に考慮している点は、

① ファジィ集合を自然な形で入出力できるようにする
② ファジィ集合演算用の関数を追加する
③ Lisp の関数をファジィ集合に適用できるようにする
④ 元になる Lisp とできるだけ整合性を保つ

である。

このファジィ集合処理システムを Common Lisp に移植したので報告する。 この移植は muLisp-86 版[1][2]をベースにして行なった。以後、Common Lisp 版を新システムと、muLisp-86 版を旧システムと呼ぶことにする。

2. 変更の概要

新システムを作成するにあたって旧システムと表記上の違いをできるだけ少なくなるようにしたが、 Common Lisp との整合性を保つため多少の違いがでてしまった。変更点は以下のようである。

① 入出力の方法

ファジィ集合処理システムでは、ファジィ集合やファジィ関係を、例えば、

```
{0.3/a, 1/b, 0.7/c}                    (1)
{0.5/(a,1), 1/(b,2), 0.6/(c,3)}        (2)
```

のように表現する。これはS式の範囲を越えている。しかし、この形式で入出力が行なえるように、旧システムでは関数(fread, fprint など)を作成し、ファジィ集合を入出力するときには、Lisp の入出力関数(read, print など)を用いないようにしていた。一方、新システムでは、次節で述べる 入力マクロと構造(structure)を定義することにより通常の Lisp の関数でファジィ集合を入出力できるようにした。また旧システムではトップレベル・ループを fread-eval-fprint ループにし、エラー処理関数も独自のものを作成していたが、これらの変更も不要になった。

②ファジィ集合型の扱い

ファジィ集合処理システムでは、ファジィ集合をファジィ集合型という型のアトムとして取り扱っている。

旧システムでは、 ファジィ集合を内部で *FS で始まるリストとして保持していたため、通常の Lisp の型チェック関数を用いることができなかったので、独自の型チェック関数(fatom, flistp, fconsp など)を用いていた。一方、新システムでは、ファジィ集合を構造として定義したため、通常の Lisp の型チェック関数が使用できるようになった。

③ 拡張 Lisp 関数の表記の違い

ファジィ集合処理システムには、 Lisp の関数を拡張して、ファジィ集合の演算にも適用できるようにするための表記法があり、

```
(&+ about-3 about-4)                   (3)
```

と記述することにより、通常の関数 + によって、 ファジィ集合のたし算が行なえるが、この表記法を変更し、

```
(#&+ about-3 about-4)                  (4)
```

とした。これは、①で示した入出力の方法の変更にともなった改訂である。

④ パッケージの使用

旧システムでは、システム内部で使用しているシンボルの名前との衝突をさけるために、ユーザは @ と * で始まるシンボル名を用いてはいけない、という制限を設けていた。新システムではパッケージを用いることにより、この制限を取り除いた。ファジィ集合処理システムはパッケージ fuzzy に入れた。

⑤ 関数の形式の Common Lisp への準拠

Common Lisp にあわせてキーワード引数を用いることにより、ファジィ集合処理用の関数の表記を統一したり、旧システムでは２つに分かれていた関数を１つにまとめた。

3. ファジィ集合処理システムの内部処理について

① ファジィ集合の定義

新システムでは、ファジィ集合を次のような構造を用いて定義した。

```
(defstruct (fset (:print-function print-fset))
           value)                      (5)
```

ここで、print-fset は、 構造の出力形式を定義した関数である。value には、ファジィ集合の値を入れる。

したがって、(1)のファジィ集合は

```
#S(fset value ((0.3 . a) (1 . b) (0.7 . c))) (6)
```

と表現されることになる。

Implementation of Fuzzy-Set Manipulation System in Common Lisp

Kenji Kume[1] and Motohide Umano[2]
1 Shingakusha Co.,Ltd. 2 Osaka University

また、組〈…〉もファジィ集合と同様に構造で表現され、

```
(defstruct (tuple (:print-function print-tuple))
          value)                                    (7)
```

と定義されているので、(2)のファジィ関係は、

```
#S(fset value ((0.5 . #S(tuple value (a 1)))
               (1   . #S(tuple value (b 2)))
               (0.6 . #S(tuple value (c 3))))) (8)
```

と表現される。

この方式により、ユーザは、ファジィ集合を単に1つの型として考えればよく、ファジィ集合に特別の考慮を払う必要がなくなり、システムの作成者は、ファジィ集合とCommon Lispとの整合性をはかるためのこまごまとした関数を作成しなくてもすむようになった。

② 入力マクロによるファジィ集合の入力処理

ファジィ集合の入力は、入力マクロを用いて行なう。これは、

```
(set-macro-character #\{ #'brace-reader) (9)
(set-macro-character
    #\} (get-macro-character #\}) nil) (10)
```

のように定義する。ここで、brace-readerは、入力文字列として、{ が現れたら、それ以降、} が現れるまでファジィ集合の要素列が続くものと見なして入力を代行する関数である。入力時に(1)が読み込まれたとすると、関数 brace-reader により、

```
(fuzzy::|%fset%| (list (cons '0.1 'a)
                       (cons '0.2 'b) (cons '0.3 'c)) (11)
```

というS式に変換される。

次に、パッケージ fuzzy の中の関数 |%fset%| が評価される。この関数は構造 fset を生成する。リーダが直接構造 fset を生成せずに、いったん(11)のS式を生成するようにしたのは、ファジィ集合の要素中に、評価すべき関数が含まれていることがあり(前に ! をつける。バック・クォート・マクロの ,(コンマ)に相当)、それを評価後、構造を生成する必要があったからである。また、組についても同様な処理が行なわれる。

③ 拡張 Lisp 関数の処理

拡張 Lisp 関数の処理も入力マクロで処理される。これは、

```
(set-dispach-macro-character #\# #\&
                             #'expand-lisp) (12)
```

と定義されている。ここで、関数 expand-lisp は、シンボル @apply を生成するだけの関数である。例えば、

```
(#&+ {0.1/1} {0.3/2})                              (13)
```

という入力は、リーダにより #& の部分が、@apply というシンボルに変換されたのち、マクロ @apply が評価されて、最終的に次のような関数に展開される。

```
(fapply #'+ (list {0.1/1} {0.3/2}))                (14)
```

ここで、関数 fapply は、関数 apply をファジィ集合演算用に拡張した関数である(計算は拡張原理[3]に基づく)。

このように、展開を二重にしたのは、関数 fapply に渡す関数のスコープを維持したかった(この例で言えば、関数 + を #'+ の形で引数にしたかった)からである。

4. Common Lisp 版での制限

現在、新システムには次のような制限がある。

① クォートの扱い

ファジィ集合の定数((1)など)の要素は、評価されないようになっている。したがって、ファジィ集合の前にクォートはつける必要がない。

旧システムでは、ファジィ集合の前にクォートを付けても付けなくても、その意味は変わらなかった。

新システムでも、' の処理を入力マクロで処理するなどして旧システムとの互換性を保とうとしたが、' の意味と、スペシャルフォーム quote が、ファジィ集合に対する処理に関してのみ互換性がないものになるなど、一部に制限が付いてしまった。ただし、{…} にバック・クォートの機能があるので実際にはほとんど問題にならない。

② 分数の処理

これは、旧システムも同じ問題を抱えているが、ファジィ集合中では分数が使用できない(/ はグレードと要素の区切りである方を優先する)。

ファジィ集合の要素間で演算を行なった結果、ファジィ集合の要素が分数になる場合もあるが、この場合には問題はないが、結果の表示が見づらいものになってしまう。

③ 〈 と 〉の処理

これも、旧システムからの問題点であるが、組を表わすために 〈 と 〉 を使用しているので、これらの文字を含むシンボルが記述できない。このため、エスケープ文字を付けることにより、この制限に対処している(例えば char〈 は、char\〈 と表記する)。

5. おわりに

現在 このシステムはワークステーション NEWS (ソニー社製、OS は UNIX 4.2BSD)上の KCL (Kyoto Common Lisp)で稼動しているが、KCL や NEWS に依存した機能は使用していないので、他のマシン上の KCL 以外の Common Lisp でも稼動すると思われる。

今後、さらにシステムの機能拡張を行なっていきたい。

[参考文献]

1. 馬野、久米(1987):「Lispによるファジィ集合処理システムの概要」、情報処理学会第35回(昭和62年後期)全国大会、pp. 780-790、No. 7Q-1.
2. 久米、馬野(1987):「Lispによるファジィ集合処理システムの実現」、情報処理学会第35回(昭和62年後期)全国大会、pp. 791-792、No. 7Q-2.
3. L. A. Zadeh(1975): "The Concept of a Linguistic Variable and Its Application to Approximate Reasoning," Information Sciences, Vol. 8, pp. 199-248; Vol. 8, pp. 301-357; Vol. 9, pp. 43-80.