記 号 処 理　32－5－2
（1985. 6. 18）

# Pros and Cons on the Common Lisp

安村通晃 （日立・中研）

## 1．はじめに

従来、ｌｉｓｐの標準化に関しては、いくつかの試み（たとえば、Standard Lispなど）があったが、いずれもうまくはいかなかった。これは、ｌｉｓｐのtranslator(移植用のツール)が比較的容易に作れること、様々な実験や拡張を自由に行ないたいため仕様を固定化したくないこと、などの理由があるが、最も大きな理由は、ｌｉｓｐがこれまで、比較的限られたユーザ内でしか使われなかったことではないかと思う。ｌｉｓｐがより広いユーザの間でも使われ初めた今日、Ｃｏｍｍｏｎ　Ｌｉｓｐの出現が、ｌｉｓｐの標準化に与える影響はきわめて大きい。まず、この点は評価してよい。

以下では、Ｃｏｍｍｏｎ　Ｌｉｓｐの仕様を、従来ｌｉｓｐとの比較、および、他言語との比較によって論ずることとする。

## 2．従来ｌｉｓｐとの相違点

Ｃｏｍｍｏｍ　Ｌｉｓｐは、Ｚｅｔａ　Ｌｉｓｐ（Ｌｉｓｐ　Ｍａｃｈｉｎｅ　Ｌｉｓｐ）などの影響を受けて、機能的に大きなｌｉｓｐになっただけでなく、次の点で従来ｌｉｓｐと大きく異なっている。

（ｉ）　ｌｅｘｉｃａｌ　ｓｃｏｐｅの採用
（ii）　ｇｅｎｅｒｉｃの徹底
（iii）　パラメタの種別の増加
（iv）　ｃｌｏｓｕｒｅの完全サポート
（ｖ）　その他

### （ｉ）ｌｅｘｉｃａｌ　ｓｃｏｐｅの採用

従来ｌｉｓｐがインタプリタ向きのｄｙｎａｍｉｃ　ｓｃｏｐｅを永い間採用してきたが、Ｃｏｍｍｏｎ　Ｌｉｓｐになって初めて（Ｓｃｈｅｍｅという例外を除く）、コンパイラ向きのｌｅｘｉｃａｌ　ｓｃｏｐｅを採用した。従来は、インタプリタとコンパイラの実行結果の違いが問題になることがあったが、ｌｅｘｉｃａｌ　ｓｃｏｐｅの採用でこの問題は解消された。他言語（たとえば、Ａｌｇｏｌ系の言語）では、ｌｅｘｉｃａｌ　ｓｃｏｐｅはいわば当然のことであったが、ｌｉｓｐでは今までインタプリタに負担がかかるのを恐れて採用してこなかったものと思われる。ただし、ｌｅｘｉｃａｌ　ｓｃｏｐｅを採用したとはいえ、後述するように、やや不徹底な点が気になる。

(ii) ｇｅｎｅｒｉｃの徹底

Ｃｏｍｍｏｎ　Ｌｉｓｐは従来ｌｉｓｐにも増して、ｇｅｎｅｒｉｃを徹底している。

まず、数値関数については、豊富なデータ型に対して、共通に一種類しか用意していない。数値型としては、ｉｎｔｅｇｅｒ（ｆｉｘｎｕｍとｂｉｇｎｕｍを含む），ｒａｔｉｏｎａｌ（ｉｎｔｅｇｅｒまたはｒａｔｉｏ）、ｆｌｏａｔ(ｓｈｏｒｔ,ｓｉｎｇｌｅ,ｄｏｕｂｌｅ,ｌｏｎｇの各ｆｌｏａｔ数)、ｃｏｍｐｌｅｘと多くの型があるだけでなく、たとえば、ｃｏｍｐｌｅｘの実部・虚部が、ｆｌｏａｔやｒａｔｉｏｎａｌ（ｂｉｇｎｕｍ，ｒａｔｉｏを含む）と複雑に組み合せられる。このように、豊富な数値型に対して関数が一種類のみなので、ユーザからは使い易く便利であるが、実現は必ずしも容易ではない。性能や精度を考えると、必ずしもｇｅｎｅｒｉｃな関数の組み合せだけですべての数値関数を記述できるとは限らないからである。

また、ｌｉｓｔとｖｅｃｔｏｒ（したがって、ｓｔｒｉｎｇも含まれる）を統一した型として、ｓｅｑｕｅｎｃｅを設け、ｓｅｑｕｅｎｃｅ関数をｌｉｓｔとｖｅｃｔｏｒの共通関数として用意している。これも、ｇｅｎｅｒｉｃの一種で便利なこともあるが、ｌｉｓｔ特有の関数も別にあるなど、完全に仕様がすっきりしたわけではない。

さらに、ｓｅｔｑの一般化として、ｓｅｔｆを設け、ｓｅｔｑの第一引数がシンボルだけであったのに対して、ｓｅｔｆでは、多くのアドレス相当のものを返すＳ式が書けるようになった。これも、ユーザからすれば便利な機能に違いないが、実現の点からすれば、対応する関数をまとめてマクロとして用意しなければならない。ところが、このマクロは、一般にソースとして与えたものと、実行時の内容とが一致しないという問題を抱えており、とくにデバッグ時などで困ることが起る場合が出てくる。

(iii) パラメタ種別の増加

関数のパラメタ種別がＣｏｍｍｏｎ　Ｌｉｓｐでは大巾に増えた。従来のｌｉｓｐでは、可変個の引数や、省略値をもつパラメタが書けるのもあったが、Ｃｏｍｍｏｎ　Ｌｉｓｐでは、ｒｅｑｕｉｒｅｄパラメタ（必須の引数）、ｋｅｙｗｏｒｄパラメタ（名前付の引数）、ｏｐｔｉｏｎａｌパラメタ（省略値指定の引数）、ｒｅｓｔパラメタ（可変個の引数）、ａｕｘｉｌｉａｒｙパラメタ（引数というよりむしろローカル変数）、ときわめて種類が多い。これは、ｓｐｅｃｉａｌ　ｆｏｒｍの数を減らす、ｆｓｕｂｒなどの特殊な関数の種類を減らす、といった仕様上の改善の他に、ユーザにとっては書き易さが増すことにもなる。個々のパラメタの機能は確かによいかもしれないが、これらのパラメタは組み合わせて使うことも可能であり、そのときの規定は複雑である。

(iv) ｃｌｏｓｕｒｅの完全サポート

Ｃｏｍｍｏｎ　Ｌｉｓｐでは、関数ｃｌｏｓｕｒｅが完全にサポートされており、このため、関数引数や関数が結果となる場合でもつねに正しく動作し、いわゆるｆｕｎａｒｇ問題という

ものが起らない。しかし、ｃｌｏｓｕｒｅを完全にサポートするためには、ｃｌｏｓｕｒｅ中の自由変数をｈｅａｐ上に置かなければならないなどのため、（少くともハードウェアの サポートがなければ）オーバーヘッドが生ずる。仕様上の整合性からは、ｃｌｏｓｕｒｅの完全サポートが望ましいが、ｃｌｏｓｕｒｅの使用頻度や、実行時のオーバーヘッドを考えると、完全サポートは疑問である。

（ｖ）　その他

その他、Ｃｏｍｍｏｎ　Ｌｉｓｐでは、ｍｕｌｔｉｐｌｅ　ｖａｌｕｅｓや、ｐａｃｋａｇｅ、ｓｔｒｕｃｔｕｒｅなど、多くの機能を提供してユーザに便宜を与えている。しかし、これらは一方では処理系を重くし、大きくする要因ともなっている。

また、たとえば、ｃｈａｒａｃｔｅｒ型というものがあって、文字の値の他に、ｆｏｎｔやｂｉｔといった属性も持てることになっている。これなどは、ｂｉｔｍａｐ付きの端末を想定しているようである。

３．他言語との比較

Ｃｏｍｍｏｎ　Ｌｉｓｐを他言語（たとえば、Ａｌｇｏｌ系の言語）と比較してみると、次のような点が浮び上ってくる。

まず、最初に、様々な機能が許容的に規定されているので、エラーの検出が難かしくなっている点である。先程述べた、パラメタの組み合せもその一つの例であるが、またたとえば、宣言は重複してもよいので、同一名称のパラメタを同じパラメタリスト内に書いても、エラーにはならない。

また、ｌｅｘｉｃａｌ　ｓｃｏｐｅの採用は、従来ｌｉｓｐから比べると進歩ではあるが、いくつかの点で不徹底である。たとえば、変数については、ローカル変数はごく普通に定義できるが、関数の場合、ローカル関数はｄｅｆｕｎではなくｆｌｅｔやｍａｃｒｏｌｅｔなど特殊な関数でないと定義できない。ｐｒｏｐｅｒｔｙ　ｌｉｓｔもグローバルな扱いになつている。　Ｃｏｍｍｏｎ　Ｌｉｓｐでもｓｐｅｃｉａｌ宣言により、変数毎に　ｄｙｎａｍｉｃ　ｓｃｏｐｅにすることもできるが、このｓｐｅｃｉａｌ宣言が定義のみならず参照に対しても別箇指定できること、および同一のシンボルでｌｅｘｉｃａｌ　ｓｃｏｐｅをもつものと　ｄｙｎａｍｉｃ　ｓｃｏｐｅをもつものとが同時に別々参照できること、などは不思議としかいいようがない。

さらに、Ｃｏｍｍｏｎ　Ｌｉｓｐでは数値関数についてもレパートリが増え充実しているようにみえるが、たとえば、Ｆｏｒｔｒａｎなどと比べると、（マニュアルどおりに関数を作るたとき）、精度や性能の点でまだ及ばないと思う。

## ４．おわりに

以上、Ｃｏｍｍｏｎ　Ｌｉｓｐの特徴を、従来ｌｉｓｐとの比較、および、他言語との比較の観点からみてきたが、外部仕様からすると、機能が増え統一的な仕様となってユーザにとって望ましい面もあるが、実現上からすると、ｉｎｔｅｒｐｒｅｔｅｒよりもｃｏｍｐｉｌｅｒ重視、汎用機よりもｌｉｓｐ　ｍａｃｈｉｎｅ等ハードウェアサポートを想定しており、全体としては、性能よりも機能を重視した言語である。とくに、みかけの上の関数の数に比べ、内部の実質的な関数の規模がかなり大きくなっている。

新しい言語を、とくに大勢の人間で設計するとしばしばこのような大型の言語になる傾向にある。Ａｄａの出現以来、プログラミング言語は、大艦巨砲時代に入った観すらある。（たとえば、拡張Ｂａｓｉｃや、Ｆｏｔｒａｎ８ｘの仕様等）。ユーザからすると、機能が増えて一見喜こばしいようであるが、反面、複雑さが増し全体の理解が困難になる、処理系が重くなる、といったしわよせがユーザの方にもくる。

現在、Ｃｏｍｍｏｎ　Ｌｉｓｐの拡張案の検討もすすんでいると聞く。それも大事かもしれないが、今のＣｏｍｍｏｎ　Ｌｉ'ｓｐの仕様の見直しを必要であろう。マニュアル自身も、厳密に読もうとすると、よく判らないところがでてくるし、細かい点で記述されていない点もある。従来のｌｉｓｐをよく知っている人間が、例題を見ながら想像力を働らかせて読むのに適したマニュアルのようである。言語仕様書と設計理由書とは分けて書き、仕様書のほうはもう少し厳密に書いてほしい。

Ｃｏｍｍｏｎ　Ｌｉｓｐの仕様についてやや批判がましい見解を述べたが、最初にも述べた通り、Ｃｏｍｍｏｎ　Ｌｉｓｐの出現そのものは、ｌｉｓｐの標準化という点で意義は充分あると思う。

### 参考文献


[１]　G. L. Steele Jr., Common Lisp： the language, Digital Press, 1984.

[２]　G. L. Steele Jr., An overview of Common Lisp, Conf. Record of the 1982 ACM symposium on Lisp and Functional programming, 1982.

[３]　R. A. Brooks and R. P. Gabriel, A critique of Common Lisp, Conf. Record of the 1984 ACM symposium on Lisp and Functional programming, 1984.