記 号 処 理 65－4
（1992. 6. 12）

Lisp 国際標準化のためのベース言語 KL について

湯浅太一（豊橋技術科学大学） 梅村恭司（日本電信電話）
橋本ユキ子（日本電気） 黒川利明（日本アイ・ビー・エム）
伊藤貴康（東北大学）

ISO の Lisp 標準化委員会においてベース言語として採択された日本案である KL を紹介する．KL は Common Lisp をベースにして設計されたコンパクトで効率のよい Lisp 言語である．本報告は，同提案に関する国内関係者の意見を求め，今後の国際標準化作業に役立てることを目的とする．

Design Considerations of the Base Language KL for International Lisp Standardization

Taiichi Yuasa (Toyohashi University of Technology)
Kyoji Umemura (NTT Software Laboratory)
Yukiko Hashimoto (NEC Corporation)
Toshiaki Kurokawa (IBM Japan Ltd.)
Takayasu Ito (Tohoku University)

This report gives design considerations of the Lisp language KL, which was designed by the Japanese SC22 Lisp Working Group. KL is a compact and efficient Lisp language, whose design is based on Common Lisp. This language has been selected as the base language for ISO international Lisp standardization.

1 はじめに

Lisp は人工知能・記号処理用の言語として広く使用されてきているが，実用的な Lisp 言語として現在最も広く使われているのは Common Lisp と Scheme である．これらの標準化は，人工知能・記号処理分野のソフトウエアの流通のために極めて重要視され，数年間に亘って標準言語設計の努力が行なわれてきている．

Scheme に関する標準化は，IEEE の MMSS (Microprocessor and Microcomputer Standards Subcommittee) において行なわれてきたが，1990 年 12 月に承認されたドラフト [10] が 1991 年 7 月から正式に IEEE 標準として流布されることとなった．

Common Lisp の標準化は ANSI で行なわれてきているが，Guy Steele の本 [6,7] に見られるように大規模な言語であるために，標準化作業が手間取っている．それだけでなく，Common Lisp が余りに大規模言語となっているために，米国における Common Lisp ユーザが減少傾向にあり，標準化に値する言語であるか否かの疑問も提起されるようになっている．このため，ANSI では標準化作業を急いでいる．

ISO における Lisp 言語の国際標準化は，1987 年に発足した ISO/IEC JTC 1/SC 22/WG 16 委員会（以下 WG16 と略す）において作業がすすめられてきた．日本においては，情報処理学会規格調査委員会 SC22 LISP ワーキンググループ（WG）が ISO の国際標準化作業に参加しており，1989 年 9 月に仙台で行われた WG16 の第 5 回会合では日本語文字も含めた国際文字処理方式に関する提案 [4,5] や，Common Lisp をベースとしたコンパクトで効率の良い核言語 [8] の提案を行うなど，活発な活動を続けてきた．

1992 年 1 月に米国 Cambridge で行なわれた WG16 の第 8 回会合で次のような決定（Action Item の採択）が行なわれた．

1. Lisp 核言語（Kernel Lisp）
   日本の提出した “The Kernel Lisp Language for ISO Lisp Standardization” [9] を Lisp 核言語設計のベース・ドキュメントとする．

2. オブジェクト指向の核言語（Kernel for Object Oriented Features）
   ANSI の CLOS（Common Lisp Object System）[1] をベース・ドキュメントとする．

これによって，Lisp の国際標準化作業を具体的にすすめるための基礎が確立した．今後，これらのベース・ドキュメントを検討・改訂し，1 つのワーキング・ドラフトとしてまとめてゆく作業が WG16 において行われることになる．

ベース言語として採択された日本案（以下 KL とよぶ）は，仙台会議において報告されたものを改訂したものであり，基本的な枠組みは同じである．KL にはオブジェクト指向機能が定義されていないが，近年の Lisp 言語の傾向を反映して，オブジェクト指向機能がぜひ必要であるとの観点から CLOS が採択された．しかし CLOS の仕様は巨大すぎるために，ANSI 案を大幅に縮小した仕様（ISO Lisp Object System）[2] がすでに米国から提出されている．

以下では，日本の提案した KL について，その設計の基本的な枠組みを報告する．KL の詳細については，その仕様書 [9] を参照されたい．その際に，本報告が KL を理解するための参考になるであろう．また本報告は，国内の専門家の意見を求め，今後の国際標準化作業に役立てることを目的とするものである．

2 設計方針

近年のLispの現状をみると，処理系の普及度，応用プログラムの量，ユーザ数，計算機メーカのサポート体制のいずれをとってもCommon Lispが圧倒的に他のLisp系言語より広範囲に利用されており，実質的な標準言語としての地位を築いている．したがって，国際標準言語においてもCommon Lispからあまりにかけはなれたものは受け入れられないであろう．

一方で，Common Lispについては様々な問題点が設計段階から指摘されており，日本のSC22 LISP WGにおいても，WG16発足当初からこれらの問題点の検討を行ってきた[3]．その最大の問題点は，言語の規模が大きすぎるという点である．特に国際標準言語として考えた場合，その大規模な仕様のために次のような障害が予想される．

- 仕様書の作成に時間がかかる．
- 厳密な仕様記述が困難である．
- 仕様の検討に多大の時間を要する．
- 効率のよい処理系を試験的に作成することが困難である．

これらが標準化作業にいかに深刻な問題であるかは，ANSIのCommon Lisp標準化作業が難航していることをみれば明かであろう．また，Common Lispが使用されている状況をみると，その言語規模のためにプログラマの教育に時間がかかる，処理系が巨大になり計算機資源を圧迫する，などの問題が生じている．

以上の理由から，KLの設計にあたっては，Common Lispをベースにし，その核となる部分を抽出することによってコンパクトな言語にするという方針を採用した．核の抽出にあたっては，特に次のような機能は可能な限り仕様から除去することとした．

- ほとんど利用されない機能．
- 他の機能の組み合わせで効率を低下させることなく実現できるもの．
- プログラムの静的な解析を困難にする機能．
- 現在の計算機では効率よく実現できない機能．
- その機能のために，他の機能の効率を著しく低下させるもの．

しかしながら，KLは核言語とはいえ，それ自体でLispの応用プログラムの大部分が記述できる完結した言語であることとした．このため，例えばCommon Lispの組み込みマクロは原理的にはユーザが定義できるのだが，非常に頻繁に使用されるdoなどはKLの仕様に含まれている．

3 Common Lisp との互換性

Lispの標準化作業にあたっては，試験的な処理系の開発が欠かせない．これは，作成段階の仕様書の不備を補う役割を果たすと同時に，仕様が現実に効率よく実現できるかどうかの貴重な情報を提供してくれる．しかし，いかにコンパクトなLisp言語であろうと，その効率のよい処理系を作成するにはそれなりの手間と時間を要する．そこで，広く普及しているCommon Lisp処理系を有効に利用できるように，次の方針を採用した．

KLで記述したプログラムは，Common Lisp処理系でコンパイルでき，かつ実行速度のペナルティなしに処理系本来の速度で実行できること．

これは標準化作業の期間のみならず，標準言語が実用化されるためにも重要である．もちろん標準言語をターゲットとする処理系であればさらに効率のよい実行が期待できるが，そのような処理系がでまわるまでには，（Common Lispの場合のように数年も要することはないであろうが）かなりの時間が必要だからである．

上の方針は，Common LispのインタープリタがKLのプログラムを必ずしも効率よく実行できることを意味するものではない．インタープリタによる実行では，KLのプログラムをシミュレートするためのオーバーヘッドが生じる可能性があるからである．また，上の方針は，KLがCommon Lispのサブセットであることを要求するものではない．KLをプログラミング言語として統一のとれたものとするためには，ある程度の機能の追加が必要である．また，標準言語として有用な機能はCommon Lispにとらわれずに採用できる余地は残しておくべきである．

KLがCommon Lispの完全なサブセットではないといっても，Common Lispと同じ名前でありながら異なった機能をもつものは，いたずらに混乱をまねくだけであり，避ける方針がとられた．すなわち，KLの組み込み関数，組み込みマクロ，特殊フォームのうち，Common Lispと同じ名前を持つものは，その機能はCommon Lispのサブセットであり，KLでの動作はCommon Lispでの動作と同一である．その差は，KLでの機能制限である．たとえば，Common Lispでの`defstruct`のオプションのいくつかはKLでは使えない．また，KLにもCommon Lispと同様に`format`関数があるが，フォーマットの指定に使えるフォーマット文字列はKLでは非常に制限されている．しかし，KLで許されるフォーマット指定は，Common Lispでも利用することができる．

現実には，上の方針は言語設計に対してかなり大きな制約となる．例えば，Common Lispでは`nil`という記号，空リスト，偽値はすべて同一のデータであり，これらを異なったデータとすることはCommon Lisp処理系によるシミュレーションを（不可能ではないかもしれないが）非常に困難にするであろう．この3者を同一のデータで表現することは，古い悪習慣をひきずっているという意見も日本のWGにあり，IEEEの標準Schemeのように3者を異なったデータ（IEEE Schemeではそれぞれ`nil`，`()`，`#f`と表記される）とする案も出されたが，上の方針のためにCommon Lispと同一とすることとした．

同様に，変数と関数が異なる名前空間を使用する点もCommon Lispに従うこととした．このために，変数の値が関数データである場合でも，その関数を呼び出すには，`funcall`や`apply`などを使用するように言語を定めた．

これらの例でわかるように，言語の基本部分においてはKLはCommon Lispのサブセットに近いものとなっている．

4 プログラムの静的解析

KLでは，Common Lispのいくつかの機能を削ることで，Common Lispよりもプログラムの可読性をあげ，コンパイラやツールなどがプログラムの静的な解析を容易に行えるように工夫されている．Common Lispでは言語のシンタックスさえも動的に決定せざるをえないことが多い．例えば，プログラム・テキストに現れた`ABC`という文字列が環境によって記号となったり16進数となったりする．このような静的解析の困難さは，プログラムの解読を困難にし，バグの原因となることはいうまでもないが，さらに，コンパイラとインタープリタの不整合を招くことになる．

この問題を解決するために，KLではreaderの振る舞いを変化させる大域的な環境を設定する機能を可能な限り削除した．例えば，

1. リード・マクロ，ディスパッチ・マクロを定義する機能．

2. 数値入力時の基数を指定する大域変数.

3. パッケージをグローバルに変更できる機能.

などは削除されている．リード・マクロやディスパッチ・マクロを定義する関数はそっくり削除されており，言語に組み込みのマクロだけが許される．これらの関数は，Lisp で他の言語の処理系を作成する場合には便利であるが，必ずしも必要ではない．数値の基数指定については，Common Lisp にも定義されている `#X`（16 進指定），`#O`（8 進指定）などの組み込みのディスパッチ・マクロで明示的に指定する．

パッケージについては，その本来の目的であるプログラムのモジュール化をサポートすることで代用する．つまり，Common Lisp では個々のファイルがプログラム単位であるのに対して，KL では複数のファイルから構成されるモジュールをプログラム単位とする．このモジュール化機能は Common Lisp のパッケージ機能を使って実現できるように設計されており，そのために「モジュール」を「パッケージ」とよぶことにした．KL ではパッケージのインターフェイスをパッケージごとに与えることが要求されている．このインターフェイス部には，パッケージごとに外部から参照できる名前とその種類（関数名か，マクロ名か，変数名か，など），およびパッケージ内部から参照する外部の名前とその種類を記述する．これにより，パッケージごとの名前の管理が可能となり，プログラムの静的解析が容易となり，様々な最適化の可能性が生まれる．例えば，パッケージ内の情報だけによって，ある大域関数が実行中に再定義される可能性の有無が判断できる．その可能性がない場合は関数名を表す記号を経由して関数定義を取り出す必要がなくなり，関数の本体を直接呼び出すことによって高速な関数呼び出しが可能となる．また，その関数名を表す記号や関数オブジェクトさえも不要になることがあり，メモリ効率の向上にも役立つことがある．

5 高性能処理系の可能性

前節の議論は KL のための高性能処理系の可能性を示唆するものであるが，KL の設計にあたっては，この他にも高性能の処理系を比較的容易に実現できるような配慮がなされている．

その 1 つとして，汎用の関数をできるだけ削減し，専用の関数群で代用するという方針があげられる．例えば，`memq` という関数の追加があげられる．(`memq` x y) は Common Lisp における (`member` x y `:test #'eq`) と等価である．多くの Common Lisp コンパイラでは，`member` がこのパタンで呼び出されることが多いことを知っており，その場合は特別の関数（`memq` に相当する内部関数）を呼び出すことによって効率よく実行するようになっている．効率のよい実行のためには，同様の処理を Common Lisp の多くの汎用関数に対して行う必要があり，これが Common Lisp コンパイラの開発コストを上昇させ，コンパイラ自体を肥大化される大きな要因となっている．むしろ（多くの Common Lisp 処理系が実際に行っているように）用途を限った `memq` のような関数を言語仕様に含めることによって，効率のよいコンパクトなコンパイラを容易に開発することができる．また，これらの特化された関数の提供は，とかく初心者がおかしがちな効率の悪いコーディング（例えば (`member` x y `:test #'eq`) と書けばよいところを，(`member` x y) と書く）を防止する上でも有益である．

Common Lisp コンパイラを複雑で巨大にしている要因の 1 つに，型チェックの必要性があげられる．汎用の関数が多いために，プログラム・テキストに与えられた情報から型情報を取り出し，特化された内部関数の呼び出しを行うのである．上の議論がこの問題を解消するのにも役立つことは明かである．

どうせ内部関数を呼び出すのであれば，どのような内部関数が存在するかを言語仕様に明記することによって，処理系に依存しない効率のよいプログラムを作成することが可能になる．このアイデアを KL は generic 関数の形で取り込んでいる．ここでいう generic 関数とは，CLOS のそれに似たものである

が，オブジェクト指向とは直接関係しない．KLでは組み込み関数かユーザの定義した関数かに関係なく，すべての関数はgenericである．個々のgeneric関数は，特定の型の引数が与えられたときに呼び出される特化されたサブ関数を内臓している．サブ関数はCLOSのメソッドに対応する（なぜメソッドという用語にしかなったかという質問がWG16でも出たが，これはKLのgeneric関数がオブジェクト指向とは直接関係しないことを明示するためである）．

サブ関数をプログラムで指定するには，Common Lispにおいてフォームの値の型を明示するために使用される`the`を用いる．例えば，加算を行う関数`+`は，浮動小数点数用の加算関数と整数用の加算関数をサブ関数としてもっている．KLの仕様書にはそのように明記されている．単に(`+` x y)とすれば，汎用の加算関数が呼び出され，その型によって適切なサブ関数が選択され呼び出される．浮動小数点数用のサブ関数を直接呼び出したければ，`the`を使って(`+` (`the float` x) (`the float` y))とするのである．このような`the`の用途は，Common Lisp本来の用途とは異なるが，`the`を使ったKLのフォームはそのままCommon Lispでも正当なフォームである．しかし，KLにおける`the`は，その取扱いが処理系依存であったCommon Lispの`the`より優れた特徴であり，かつ互換性も損なわない．

6 計算機アーキテクチャとの相性

Common Lispの機能のうち，現在の計算機アーキテクチャによるサポートがなく，それを実現するとオーバヘッドが現れるものは言語機能から削る方針とした．

例えばキーワードパラメータは，現在の計算機では効率のよい実現が絶望的であり，KLの仕様からは削除されている．実際のCommon Lisp処理系では，引数とキーワードとの一致検索を何回かのパスを通して行なうものが大多数であり，その非能率性ゆえにユーザはキーワードパラメータを使った関数定義を避ける傾向にある．KLでは`&rest`パラメータを使って任意個数の引数を受け取る関数が定義できる．オーバヘッドを覚悟で敢えてキーワードパラメータの機能を使用したければ，`&rest`パラメータを使ってユーザが実現することは可能である．

数値データのratio（分数）とcomplex（複素数）に対しては，それらの演算にハードウェアのサポートを期待できず，また利用価値も低い（Common Lisp処理系が世の中に登場した当時は，デモを見ている人を驚かせるという用途があったにはちがいないが）ので，削除した．Bignum（適切ではないかもしれないが，任意長整数と訳されることがある）の採用の是非は議論の残るところである．Bignumはその内部構造が複雑であり，ハードウェアのサポートを期待できないのは事実である．しかし，（大きな整数の階乗を計算して初心者を驚かせる以外に）bignumを必要とする数式処理などの応用がないわけではない．このためKLではbignumの排除を明記せず，整数の有効桁数は処理系依存とし，最低限24ビットの符合つき整数であることのみを要求している．

7 文字処理機能

多くのプログラミング言語において，1文字を1バイト（8ビット）で表現できない，いわゆる多バイト文字（multioctet）の取り扱いが問題となっている．漢字を含む日本語文字は多バイト文字であり，この問題はわが国においては切実である．

日本のSC22 LISP WGでは，国内のLisp処理系における日本語文字取扱いの実例を考慮し，1バイトの英文字と2バイトの日本語文字をプログラマが区別する必要のない言語仕様が望ましいと判断している．一方，ISO SC22において多バイト文字の扱いに関するプログラミング言語共通の国際基準を設定するための作業がすすめられており，Lispにおいても基本的にはその作業の進展を待つことが賢明であろう．

KLでは，文字データを1バイト文字に限定する記述はいっさい避けている．KLでは文字は256種類以上で，その数の上限は処理系依存としている．また，多バイト文字と1バイト文字を区別するような機能もまったく含んでいない．したがって，例えば，すべての文字が文字列の要素となることができ，要素の置き換えに際しても任意の要素を任意の文字で置き換えられることとしている．

全角（2バイト）の英字と半角（1バイト）の英字といった，異形同字の扱いに関しては，現在のKLはまったく触れていない．Common Lispではreaderが大文字と小文字を同一視する場合が多い．例えば，記号を読み込む時や，ディスパッチ文字（数値読み込みの際の基数指定などを含む）の取扱いの場合は大文字と小文字は同一視される．異形同字についてもこれと同様に対応することが考えられるが，既存の処理系でそのような機能を持つものは知られておらず，慎重に検討を行っている．

さらに日本語に関しては，一般にJIS，シフトJIS，EUCと呼ばれる複数のコード系が現実に使用されており，ファイルや端末機との入出力の際にどのコード系を使用するかが常に問題となっている．この問題については，ISO SC22における統一的な基準の作成を待つしか方法がなさそうであり，KLではこの問題には触れていない．

8 まとめと今後の課題

Lisp国際標準化のためのベース言語として採択されたKLの概要を報告した．KLはCommon Lispをベースにし，Common Lispとの互換性を強く意識して設計されている．しかしその仕様はコンパクトであり，効率のよい処理系が容易に実現可能である．

現在のKLには定義されていなが，標準案には望ましい機能として次のものがあげられる．

- オブジェクト指向機能
- エラー処理機能
- 国際文字処理機能

オブジェクト指向機能については，はじめにもふれたようにANSIのCLOSをベースにした比較的コンパクトな仕様が米国から提案されている．Common LispをベースとするKLは，Common LispをターゲットとしたCLOSとは相性がよさそうである．ANSIのCLOSはCommon Lispと共通する問題点をかかえているが，米国の提案ではそれらがかなり除去されており，KLへのマージの可能性を検討する作業をすでに開始している．エラー処理に関しては，ANSIのCommon Lisp標準案が大いに参考になると思われる．国際文字処理機能については先に述べた通りである．

現在，日本のSC22 LISP WGではオブジェクト指向やエラー処理機能を含んだKLの仕様書を作成中である．この新しい仕様書では，標準化作業が円滑に行えるように，言語の核になる部分をより明確にした次のような構成をとる予定である．

1. Introduction
2. Lisp Language Definition
   - 2.1 The Kernel
   - 2.2 Enhanced Features
3. System Interfaces
4. Object System
5. Digressions

A. BNF

伝え聞くところでは，ISO SC22 の国際標準言語のベース・ドキュメントとして日本案が採択されたのは，KL が初めてとのことである．KL をより良いものにするために，国内の多くの専門家からの御意見を期待するものである．KL の最新の仕様書（dvi 形式または PostScript 形式）は，次の電子メイル・アドレスに申し込むことによって入手可能である．

jis@rudolph.ntt.jp

謝辞

KL は，情報処理学会規格調査委員会 SC22 LISP WG における標準化作業の一部であり，同委員会の多くの委員の協力のもとに設計されたものである．また，日本電子工業振興協会 JIS 言語標準化調査研究委員会 Lisp 言語 WG における Lisp 言語の現状の報告は，KL の設計に大いに参考となった．これらの委員会の方々に感謝の意を表したい．

参考文献

[1] Daniel Bobrow and Gregor Kiczales, Common Lisp Object System Specification, Draft X3 Document 87-001, ANSI, 1987.

[2] Richard Gabriel, ISO Lisp Objct System Specification, ISO/IEC JTC 1/SC 22/WG 16 LISP N105, 1992.

[3] Takayasu Ito and Taiichi Yuasa, Some Non-standard Issues on Lisp Standardization, Proceedings of the First International Workshop on LISP Evolution and Standardization in Paris, IOS, 1988.

[4] Toshiaki Kurokawa, Taiichi Yuasa, Yukiko Hashimoto, and Takayasu Ito, Technical Issues on International Character Set Handling in Lisp, ISO/IEC JTC 1/SC 22/WG 16 LISP N49, 1989.

[5] 黒川利明, 湯浅太一, 橋本ユキ子, 梅村恭司, 伊藤貴康, LISP における国際文字処理方式に関する技術的諸問題, 記号処理 51-1, 1989.

[6] Guy Steele, Common Lisp the Language, Digital Press, 1984.

[7] Guy Steele, Common Lisp the Language, Second Edition, Digital Press, 1990.

[8] Taiichi Yuasa, Kyoji Umemura, Yukiko Hashimoto, and Takayasu Ito, An Overview of the Kernel Language for the Long-term ISLisp Design – A Compact and Efficient Lisp Specification –, ISO/IEC JTC 1/SC 22/WG 16 LISP N62, 1989.

[9] Taiichi Yuasa, Kyoji Umemura, Yukiko Hashimoto, Toshiaki Kurokawa, and Takayasu Ito, The Kernel Lisp Language for ISO Lisp Standardization – The Japanese Proposal –, ISO/IEC JTC 1/SC 22/WG 16 LISP N98, 1991.

[10] IEEE Standard for the Scheme Programming Language, IEEE Standard 1178-1990, 1990.