記号処理 40－6
（1987. 1. 13）

Ｃｏｍｍｏｎ Ｌｉｓｐにおける日本語処理方式の提案

元吉文男
電子技術総合研究所

本報告ではＣｏｍｍｏｎ Ｌｉｓｐにおいて日本語文字を扱うための方法を提案する。この方法は日本語文字を従来の英数字と全く同等に扱えるものであり、さらに、英数字だけを扱う場合で効率を重視するときにも、宣言を入れることによりメモリ効率、実行効率とも上げるようにシステムを作成することを可能としているものである。しかも、現在のＣｏｍｍｏｎ Ｌｉｓｐの仕様からそれほどはずれることもないので、既存のシステムこの日本語処理機能を取り入れることも可能である。またこの方法は日本語に限らず一般の多文字種文字についても適用できるものである。

AN PROPOSAL FOR EMBEDDING JAPANESE CHARACTERS IN COMMON LISP

Fumio MOTOYOSHI
Information Science Division, ElectroTechnical Laboratory
1-1-4, Umezono, Sakuramura, Ibaraki, 305 JAPAN

This paper proposes an method for embedding Japanese characters in Common Lisp. Although the basic idea is to make the char-code-limit value large enough to include all the characters, it is possible to use strings whose elements are all byte-characters without loss of efficiency by declaration for that. This method is also applicative for any multi-byte characters like Chinese.

１．はじめに

Lispは従来から記号処理用の言語として、主として人工知能などの研究の道具として用いられてきたが、最近になりそればかりではなく、エキスパートシステムに代表されるようになってきた。このような流れのなかで、従来は様々な方言があったLispを標準化しようとする試みがなされ、そのなかでもCommon Lispがいちはやくまとめられ、多くの応用プログラムがそれに準拠して作成されている。

日本においても、Common Lispが広く使用される傾向にあり、また独自のCommon Lisp処理系も作成されている。しかしながら、処理系の多くは日本語文字の扱いが不十分であり、内部の構造を理解していないと使いこなすことができない。Lispの利用者が研究者である場合にはこれでもよいが、一般の利用者にとっては十分とはいえない。

そこで、以下においてCommon Lispに十分な日本語文字の処理機能を取り入れる方法の提案を行う。まず２節において日本語文字を取り込む際の原則を述べる。３節においては日本語文字以外の多種類文字にも共通してあてはまることを説明する。この節の内容はデータ構造の問題であり、これがCommom Lispに取り入れられれば、各国において独自の機能を追加することは容易に行える。４節では日本語特有の処理について述べる。

２．日本語文字を取り入れる原則

Common Lispに日本語を取り入れるにあたり、よりどころとした原則をこの節で述べる。

現在でも、Lispに限らず日本語を扱えるプログラミング言語は多く存在するが、そのほとんどは、いわゆる文字列として扱うものであり、変数や関数の名前としては使用できないものである。メッセージを日本語で出力する程度の目的にはこれでも十分であり、実際、データが数値を主として扱うならばこれで用が足りる。しかし、Lispのように、記号を扱いしかもそれらが構造を持っているような言語においては、日本語文字を英数文字と対等に扱える、すなわち文字列中はもちろんのこと、シンボル中にも使用でき、しかもそれらの中で両方の文字が混在することも可能となっている、ことが求められる。

日本語文字が英数文字と対等であるということは、日本語文字の

文字がデータとしてみた場合にも１文字として扱われることを意味する。現在のシステムの多くは日本語を扱えるといっても、内部ではその１文字が２バイトで構成されそれぞれが独立した文字として扱われている。あるいは両方の文字の混在を許さないような別のデータ型を使用してこの問題を避けているものもある。これでは、使いずらいシステムとなってしまうので、第一の原則として次の立場をとることにする。すなわち、**日本語文字も英数文字と対等に扱い、どちらの１文字もデータとしても１文字として扱う。**

この原則に従って日本語が取り入れられているシステムにおいては、次のことがいえる。**特に日本語を意識せずに作成されたプログラムに日本語を含むデータを与えた場合にも、期待したとうりに動作する。**このことは特に欧米からのプログラムを日本語も扱えるように移植するに際しては重要である。これにも問題がないわけではない、というのも印字の際に日本語文字の幅が英数文字の幅と異なっている場合に、エディタなどで印字幅を文字数で計算していると、思わぬ結果になることがある。これは日本語文字を内部で２文字として処理しているシステムのほうがうまく働くように思われるかもしれないが、これは次のような特殊な事情によっている。すなわち、日本語文字と英数文字のの内部表現での比２：１が、印字幅の比２：１と同じであることに依存している。

これは日本語だけを扱うならば欠点にもなり得るが、一般の多文字種文字を扱う場合のことを考えると、いつも内部表現の大きさの比と印字幅の比が等しくなるとはいえず、それよりも、外部表現での１文字が内部においても１文字として表現されていることの利益が大きいとかんがえられる。なお、印字幅の問題にかんしては文字数で数えるのではなく、べつに印字幅を求める関数を容易すれば済むことであり、このほうが一般性がある。ここで次の原則をあげておく。**日本語文字だけでなく、一般の多文字種文字について同じ方法で取り込めるようにする。**

機能の面からは以上で述べたことで十分であると考えられるが、システムとしてみた場合の効率を無視することはできない。そこで次のような原則を念頭にいれておくことにする。**日本語文**

字を全く使用しないプログラムにおいては、実行効率、メモリ効率ともに落とさないようにする。この原則は今までの原則と背反するところがあるが、実際のシステムの負荷を考えると無視することもできないので、両方の妥協点を探すことになるが、次節で述べるものは両方をなんとか満足させるものであると考えている。

3．多種類文字に共通の部分

この節では日本語文字に限らず一般の多文字種文字をCommon Lispに取り込む方法を述べる。従ってこの節で日本語文字といわれている部分はそこをほかの多文字種文字で置き換えてもそのまま通用する。

2節で述べたように日本語もじを英数文字とまったく対等に扱えるようにするには、character codeの中に日本語文字を取り込むことである。Common Lispにおいてはchar-code-limitという変数があり、この値を大きくとることによって、日本語文字までも扱うことができる。この場合にひとつ前提があり、それは英数文字と日本語文字とをcharacter codeのなかで一列に並べられるということである。すなわち、日本語文字のコードが英数字のコードと共通部分をもたなければよい。この前提はほとんどの処理系において容易に満足することが可能である。日本語文字とそのcharacter codeとの対応はそれぞれの文字種特有の問題であり、日本語については4節で述べる。

このようにすれば、フォント付きの日本語文字というものもCommon Lispの枠内で考えることが可能となり、英数字文字について行える処理は日本語文字についても何等変更することもなく使用できることになる。

基本方針は以上の通りであるが、効率のことを考えると以下のような問題がある。character codeの値が大きくそれが1バイトの大きさを越えている場合には（日本語文字の場合はこれにあてはまるが）単純なインプリメントでは英数字だけからなるストリングが1文字につき2バイト以上になってしまい、メモリ効率が悪くなる。なおシンボルはそのp-nameがストリングであるので、ストリングについて解決できればあとは問題なく実現できるので以下ではストリ

ングを中心に考えることにする。

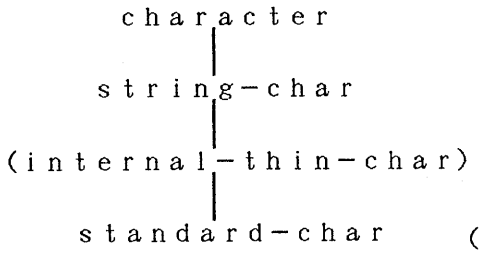
メモリ効率の問題を避けるには、内部では英数字だけからなるストリングを最適化して1バイトのストリングとして圧縮した表現にすることが考えられる。これはいわば内部で2種類のデータ型を持たせ、ユーザにはその型をみせないということである。このようにするとユーザからは単にcharacter codeが大きくなっただけのようにみえるので、よさそうに思われる。しかし、ユーザからみた場合に1種類のストリングしかないということは、ストリングを扱うすべての関数において、その定義どうりの機能が実現されなければならない。これもたいていの場合はそれほどの負担がなく実現けきるが、ただ1つ、最適化されて圧縮されているストリングにｃｏｄｅの値が2バイト以上の文字を代入しようとしたときに問題が生じる。すなわち、その場合でも定義どうりに機能するためには、もとの圧縮されてあるストリングを圧縮されていないストリングに変換しなけれｖばならない。このストリングが長い場合にはシステムの負荷が大きくなってしまう。

そこで、いっそのことユーザにも2種類のデータ型をみせることにして、システムの負荷が大きくなるものについては、エラーとして扱ってしまうことが考えられる。実際、ここで提案する方式はそのようなものである。以下で詳しい説明を行う。　日本語文字を効率よく扱うために、ストリングに2種類の型を設ける。その1つは従来のストリングと同様にbit･font属性が0の文字が全て要素となり得るもので、これをstringという呼ぶことにする。もう1つは最適化した文字列を表わすためのものでこれをinternal-thin-stringと呼ぶ。また、internal-thin-stringに入れられるという文字型の副型internal-thin-charを導入する。これを判定する述語をinternal-thin-char-pとする。

このinternal-thin-charはstring-charより下位で、standard-charより上位にあるものとする。したがって、ストリングと文字型の型階層は図1のようになる。しかし、これはinternal-thin-charがstring-charとおなじ集合を示すことを妨げるものではない。すなわち、あるシステムにおいては両者が同じ文字集合を表わしていることも有り得る。これは上の話でユーザからみえるデータ型が1種類しかないものと同様であり、どちらのインプリメントも可能とするものである。

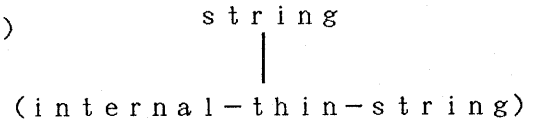
文字列を比較する場合には、それぞれの文字列から要素を取り出してそれについての比較を行うことにする。このことからも分かる

ように、internal-thin-string中にあろうがstring中にあろうが、コードが同じ文字は同一のオブジェクトとして扱われる。従って型



characterの型階層

図1．a



stringの型階層

図1．b

としては異なるものであろうとも、文字列としては等しくなるものも生じて来る。

次に入力に関する扱いについて述べる。なお、ここではファイル表現や端末との間の処理のことについては規定しない。これは、システムによって大きく異なることが考えられ、この部分については自由にできるものとする。ここではデータがLispに渡された後のことを考える。読み込みの際に読まれた文字列をどちらの型にするかであるが、安全性を考慮して全てstring型になるものとする。ただし、シンボル中のストリングについては書き換えられることがないので、最適化してもよいものとする。

このようにすると、英数字だけを扱うプログラムでは効率が落ちるので*default-string-read-type*という変数の値がinternal-thin-stringのときには読み込むデータ中に日本語文字が現れない、すなわち読み込まれたものをinternal-thin-stringとしてもよいことにする。ただし、これは必ずそのようにすべきであるというわけではなく、あくまでも、行なってもよいというものである。

4．日本語特有の部分

日本語文字のコードについては特に規定しないものとする。ただし、平仮名、片仮名についてはそれらの内部での順序はＪＩＳと同じであることが望ましい。

日本語文字（standard characterを除く）に対する構文的な意味は普通のアルファベットと同じ意味をもつのを標準とする。これはＪＩＳの３区の文字についてもあてはまる（それがstandard characterと定められていないならば）。しかし、それらについて特別な構文的な意味を持たせるモードを持たせてもさしつかえない。この意味のもたせかたとしては次のようなものが考えられる：

１．　（ｌｉｓｔ　ａｂｃ　／ｆｇ　”ｘｙ“）

２．　（ｌｉｓｔ　ａｂｃ　／ｆｇ　”ｘｙ“）

３．　（ｌｉｓｔ　ａｂｃ　／ｆｇ　”ｘｙ“）

この例で下線を引いた部分でそれに対応するstandard characterがある場合にはその対応するもので置き換えてから処理を行うものである。ただし、このようなモードを設ける際には全体として統一がとれている必要がある。たとえば、ｆｏｒｍａｔ中における文字列の解釈も同じ原則に従うべきであろう。

さらにＪＩＳ３区の文字については（それがstandard characterと異なる場合に）問題がある。すなわち、alpha-char-pなどの述語に対する態度である。これはシステムにより扱いが異なってもよいものとする。というのも現在では様々な意見があり標準として定めるのは尚早であると考えられるからである。だだし、alpha-char-pに対してＴを帰すようなシステムにおいてはそれ以外の関数、たとえばlowe-case-pなどについて統一がとれて、矛盾がないようになっている必要がある。

また日本語文字の中での種類を判別する関数についても用意することにする。これには、Japanese-char-p、katakana-p、hiragana-p、kanji-pなどが考えられる。さらに文字相互の変換katakana-hiragana、hiragana-katakanaなども加える。これらの詳細については別途定めるものとする。

５．おわりに

本提案は電子協マイコン技術委員会Ｌｉｓｐ技術専門委員会漢字ＷＧにおける討論をもとに作成されたものであり、日本語をＣｏｍｍｏｎ　Ｌｉｓｐに取り入れるさいの目安になれば幸いである。

各関数の細部については本提案では触れていないが、それらについては上記の委員会より報告集が出される予定である。

６．参考文献

Ｓｔｅｅｌｅ：Ｃｏｍｍｏｎ　Ｌｉｓｐ：ｔｈｅ　ｌａｎｇｕａｇｅ．
Ｄｉｇｉｔａｌ　ｐｒｅｓｓ，１９８４．

以下の文献はいずれも情報処理学会記号処理研究会８６－ＳＹＭ－３６のものである。

伊、高木、牛島：日本語テキストにおけるパターンマッチング手法の比較と改善．

松尾、牛島：Ａｄａにおける日本語テキスト処理パッケージの構築とその使用

杉村、奥乃、竹内：ＴＡＯにおける日本語文字列処理

前端、川合：日本ＤＥＣのＡＩ製品における日本語化について