記 号 処 理 51－1
（１９８９．６．２）

ＬＩＳＰにおける国際文字処理方式に関する技術的諸問題

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

ＩＳＯ　ＬＩＳＰにおける国際文字処理規格として、国内のＳＣ２２／ＬＩＳＰＷＧにおいて審議され、ＩＳＯ　ＷＧに報告された提案を、技術的問題点を中心に報告する。本報告は、同提案に関する国内関係者の意見を聞くことを目的とする。

TECHNICAL ISSUES ON INTERNATIONAL CHARACTER SET HANDLING IN LISP

Toshiaki Kurokawa (IBM Research, Tokyo Research Laboratory)
Taiichi Yuasa (Toyohashi University of Technology)
Yukiko Hashimono (NEC Corporation)
Kyoji Umemura (NTT Software Laboratory)
Takayasu Ito (Tohoku University)

This report gives a brief description of the proposal for international character set handling to be included in the ISO Lisp Language standard. This proposal has been discussed at the Japanese SC22/Lisp Working Group and was presented at the ISO/IEC JTC11/SC22/WG16 LISP.

1　はじめに

ＩＳＯ／ＩＥＣでは、現在ＬＩＳＰ言語の標準化に向けての議論・検討が進められており、これに対応するために国内においても情報処理学会のもとにＳＣ２２／ＬＩＳＰ　ＷＧが活発な活動を行なっている。標準化作業にあたって考慮すべき項目は数多いが、その中でも文字の取り扱いは、従来ＬＩＳＰ処理系がサポートしてきた文字集合が国や応用領域、ＯＳ環境などに強く依存してきたために、その標準化には多くの技術的検討課題が残されている。

いうまでもなく、標準化活動は応用プログラムのポータビリティを向上させることを目的としたものであり、国際文字処理方式（International Character Set Handling、以下ＩＣＳＨと略す）もプログラムのポータビリティを高めるように決められるべきである。文字処理方式に関しては次の３つのポータビリティのレベルが考えられる。

レベルA　（国際）標準文字上に開発されたプログラムを考える。標準文字のみがプログラムに入力される限り、標準文字集合のスーパーセットをサポートする任意のＬＩＳＰ処理系において、このプログラムが変更なしに動作することを保証する。ここで、“文字集合上に開発されたプログラム”とは、その文字集合を使って記述され、入力としてその文字集合内の文字を受け付けるプログラムを指す。

レベルB　標準文字集合上に開発されたプログラムを考える。標準文字のスーパーセットＡをサポートする処理系上でこのプログラムが実行されるとき、集合Ａの任意の文字を取り扱えることを保証する。

レベルC　文字集合ＢとＣをそれぞれサポートしている２つの処理系ＸとＹを考える。Ｘ上でＢとＣに共通する文字のみを使って開発されたプログラムが、処理系Ｙ上でも変更なしに動作することを保証する。

ここで、標準文字集合として特定の文字集合を仮定することはしないが、従来大多数のＬＩＳＰ処理系がサポートしてきた文字集合の共通部分を取り、[ISO478]で勧告されているようにISO646のコード化文字集合 [ISO646] に適合することが望ましい。

上記３レベルのポータビリティに関しては、その優先順位はＡ、Ｂ、Ｃの順である。レベルＡが達成できないＩＣＳＨの仕様はまったく無意味であり、レベルＢは異国間のアプリケーションの交換のためには是非とも必要である。レベルＣは各国内における母国語の処理プログラムの普及には欠かせない。

ＳＣ２２／ＬＩＳＰ　ＷＧにおいては、ＬＩＳＰ言語標準化にあたって、これら３つのレベルのポータビリティを達成する（少なくとも向上させる）ために、ＬＩＳＰにおけるＩＣＳＨの技術的諸問題の検討を重ねてきた。以下ではその概

要を報告する。詳細については［ICSH］を参照されたい。

2　ＩＣＳＨに関する一般的注意事項

欧米各国では、人々は日常生活で比較的少ない数の文字（letter）を使用しており、これらの国の計算機システムで使用する文字（character）数も比較的少ない。これらの文字は１バイト（８ビット）コードで表現可能である。しかし、これは日本の場合を考えれば明かなように、決して一般的な場合とはいえない。日本人は日常生活で６０００以上の文字を使用しており、計算機システムにおいては２バイト表現が必要となる。そこで、ＩＣＳＨの設計に当たっては、文字が必ずしもバイトに対応しないということをまず心に留めておかねばならない。

欧米において使用文字数が少ないことは、計算機システムがＡＳＣＩＩやＥＢＣＤＩＣのような単一のコード化文字集合（coded character set）を使用することを可能にしている。これは、１バイト表現がメモリ効率および実行効率において無駄がないためであり、ほとんどの場合、効率向上のためにいくつかのコード化文字集合に文字を分割する必要は生じない。しかし、１バイト表現が効率的でないと考えられる場合は、英数字のような頻繁に使用される文字に対してはよりコンパクトな表現を必要とすることがあり、従って複数のコード化文字集合を持つ必要が生じる。

同じ状況が日本を含めたいくつかの国で生じる。２バイト表現はメモリ効率が良くないことがあり、計算機システムでは１つ以上のコード化文字集合を使用している。日本における典型的な例は、多くの計算機システムがコード化文字ＡＳＣＩＩとＪＩＳ　Ｘ０２０８の両方を使っていることである。そして、前者の集合内の文字と後者の集合内の文字とを区別するための約束事を伴っている。それゆえ、ＩＣＳＨの設計においては、２つ以上のコード化文字集合を扱うＬＩＳＰ処理系が存在するということも念頭に置いておかねばならない。

本節での最後の注意事項は、ＬＩＳＰ処理系がサポートする文字が一様に扱われるようにＩＣＳＨは設計されるべきであるという点である。文字の集合は各国での必要性を反映して注意深く選択されてきたものであり、各国のプログラマが特定の文字を特別扱いしなければならないことは非常に不便である。ＬＩＳＰのメカニズムに関して“文字を一様に処理する”例がいくつかある。

- ◆すべての文字が一様な方法で入出力できる。
- ◆記号名に任意の文字を持つ記号を受け入れることができる。
- ◆任意の文字を持つ文字列を受け入れることができる。文字列の「長さ」は、文字列のバイト数ではなく、文字列の文字数と等しい。
- ◆任意の文字が、文字列中の任意の文字と置き換えられる。
- ◆文字列の構文属性がすべての文字に対して定義できる。

これらの要求は１バイト圏で生活している欧米の人々には当然のことばかりであ

るが、これらの最小限の要求でさえも、複数バイト圏では必ずしも満足されていないのが現実である。

3　基本概念

ＩＣＳＨの議論をする前に、本節ではＩＣＳＨに関連する基本概念を明らかにしたい。

文字オブジェクト（character object）

文字オブジェクトは、使用されるコーディングシステムと独立にプログラムを書くことを可能にするために文字コードの抽象概念を提供する。従って、独立した文字型の導入は、異なるコーディングシステムを持つＬＩＳＰ処理系間におけるアプリケーションプログラムのポータビリティを高めることを目的とする。

文字コード（character code）

各文字オブジェクトは、個々のＬＩＳＰ処理系で一意に文字オブジェクトを決定する「内部文字コード」と呼ばれる整数と対応づけられる。ＯＳが使用するコードのように、ＬＩＳＰ処理系の外側では他のいろいろなコードが使用される。これらを、ＬＩＳＰ処理系内部で使用される内部文字コードと区別するために、「外部文字コード」と呼ぶことがある。文字オブジェクトはその内部コードによって一意に決定されるので、ＬＩＳＰ処理系内部では単一のコード化文字集合を使用しなければならない。これに対して、ＬＩＳＰ処理系が対応すべき外部のコード化文字集合は複数存在する場合がある。

文字列（character string）

文字列は文字オブジェクトの有限個（０個も可能）の並びである。文字列は内部文字コードのベクタとして実現されることが多い。

文字（character）

文字オブジェクトと内部文字コードは一対一に対応するため、両者を混同して使用すると便利なことがある。この混同した概念を表現するために「文字（character）」という用語を用いることができる。例えば、文字列は実際には文字オブジェクトのベクタとして実現されていなくても、「文字のベクタである」と言い切ることができる。さらに、日常の生活で使用する「文字（letter）」の概念も含めて「文字（character）」と呼ぶことがある。しかし、日常生活における「文字（letter）」は、必ずしも文字オブジェクト（あるいは内部文字コード）と一対一に対応しない。例えば、会話の中で「文字Ａ」といった場合は、大文字・小文字の区別をしていないのが普通である。しかし、近年の大多数のＬＩＳＰ処理系ではこの両者を区別し、異なる文字オブジェクトである。言い替えれば、日常の個々の「文字（letter）」には、一般的にいくつかの文字オブジェクトに対応

することが少なくない。

処理系がサポートする文字（system-supported character）

ＬＩＳＰ処理系がサポートする文字の集合は処理系に依存する。ここで、「サポートする」とは「正確に処理できる」ことを意味し、必ずしも「処理系内に存在しうる」ということを意味しない。すべての文字が文字コードであるわけではなく、文字コードの集合が整数のある範囲を構成していることを仮定することはできない。ある整数が文字コードであるかどうかの判定は一般には困難（処理系の実行効率の点において）である。その結果、文字コードではない整数を受け取って、処理系が意味のない文字オブジェクトを生成することがあり、処理系はそのような文字オブジェクトを正しく処理できない可能性がある。サポートする文字のみを想定することによって、効率的なＩＣＳＨを実現する方が得策である。

標準文字（standard character）

ＬＩＳＰ処理系間でアプリケーションの最低のポータビリティ（前述のレベルＡ）を保証するには、すべてのＬＩＳＰ処理系が、その処理系がサポートする文字の中で共通集合を持つことを仮定する必要がある。その集合に属する文字を（国際）標準文字（（international）standard character）と呼ぶ。標準化に当たっては、標準文字の集合を規定する必要があるが、これまでの大多数のＬＩＳＰ処理系がサポートしてきた文字集合の共通部分であり、[ISO478] の勧告を満足するような標準文字集合ができるだけ早期に決定されることが望ましい。

4　ＩＣＳＨの枠組み

本説では、ＩＣＳＨに関する主要なメカニズムをあげ、ＩＣＳＨに対する要求や実現課題を整理する。

リードテーブル

リードテーブルは処理系がサポートするすべての文字に対してエントリを持つべきである。サポートする文字集合が巨大であれば、頻繁に使用される文字に対してのみあらかじめエントリを用意しておき、必要に応じて他の文字に対するエントリを追加することが可能である。

文字オブジェクトと文字列の構文

処理系がサポートするすべての文字は、（プレフィックス“#＼”をつけるといった）同じ形式で入出力できなければならない。また、処理系がサポートする文字からなるすべての文字列は、（ダブルクオートで囲むといった）同じ形で入出力できなければならない。

文字列操作

文字列中の文字は、すでに述べた意味で一様に処理されるべきである。文字列の「長さ」はその文字列中の文字数と等しくなければならない。また、文字列の任意の文字は任意の文字で置き換えられなければならない。

この要求は、ＬＩＳＰ処理系の日本語化に関してこれまでさんざん議論されてきた実現上の問題を生じる。メモリ効率の良い１バイト文字と、１バイトで表現できない２バイト文字の混在の可能性にいかに効率よく対処するかという問題である。実行効率と実現の容易さからは、すべての文字を２バイトで表現する方法が考えられるが、この方法は１バイト文字のみからなる文字列が多数存在する場合にメモリ効率の点で問題が残る。この問題に関する実現上の解決策の議論に関しては、[ICSH] を参照されたい。

記号の印字名

記号の印字名中の文字もまた一様な方法で処理されるべきである。

表示装置上で同じ印字表現を持つ記号は同一のオブジェクトであることが望ましいが、この性質をすべての処理系に対して要求することはできない。２つの異なる文字が同じ印字表現を持つことがあるかもしれないからである。一般に文字の印字表現は表示装置に依存し、ＬＩＳＰ処理系が２つの異なる文字が同じ印字表現を持つかどうかを決定することは困難（あるいは不可能）である。
従って、記号の同値性は記号の印字名の同値性によって（Common Lisp のようにパッケージ機能を有する場合は、その記号が属するパッケージと記号の印字名によって）決定されるべきである。

文字の占めるファイル領域

各文字がファイル中に占める領域の大きさは、文字とファイルの種類（およびファイルを処理するＯＳ）に依存する。さらに、ある文字が占めるファイル上の領域は、その文字が現われる分脈に依存する場合がある。特に、処理系がサポートする文字集合が複数のコード化文字集合からなる場合に起こりうる。従って、文字のファイル中に占める領域の大きさを規定するわけにはいかない。

このことは、ファイルから１文字読み込んだ後、ファイルポインタが一定の数だけ増加することを仮定するわけにはいかないことを意味する。ファイルポインタ（もしそのような概念が標準案に含まれるなら）がどれだけ増加するかは処理系依存である。

文字の表示幅

同様に、表示装置上の文字の占める幅は文字に依存する。特に、すべての文字が表示装置上で１表示単位を占めることを仮定するわけにはいかない。これは、ビットマップディスプレイや漢字端末機を考えれば当然のことである。その結果、文字の表示幅に依存したプログラムはポータブルでなくなってしまう。しかしながら、多くの重要なＬＩＳＰ応用分野のプログラム（テキストエディタ、プリティプリンタ、数式処理システムなど）は文字の表示幅に依存する。これらのプロ

グラムのポータビリティを向上させるためのメカニズムが望まれる。

外部コーディングシステムの仕様

異なるＯＳあるいはマシン間のクロス処理のためには、現在のＬＩＳＰ処理系が使用しているコーディングシステムとは異なるコーディングシステムに対応する必要が生じることがある。そのような必要性が非常に高いなら、どのコーディングシステムを各ファイルストリームに対して使用すべきかを指定するメカニズムがあると便利である。

文字の同値性

Common Lisp では、英小文字は次の場合に英大文字と同一視される。

- ◆ 文字の構文属性を取り出すとき。
- ◆ フォーマット指示子として使用されたとき。
- ◆ CHAR-EQUAL や CHAR-LESSP のような文字比較用関数において。
- ◆ STRING-EQUAL や STRING-LESSP のような文字列比較用関数において。

同様な同値性のメカニズムがＩＣＳＨに取り入れられるならば、次の３つの可能性が考えられる。

- ◆ 標準文字の間の同値性を固定する。同値関係の追加をすることはできない。
- ◆ 標準文字の間の同値性を固定する。ユーザが自分で同値性を追加できるようなメカニズムを提供する。
- ◆ 同値性を予め定義することはしないで、ユーザがすべて指定できるようにする。

ユーザが同値性を定義できる場合は、指定された同値性をすべての関連するメカニズムに適用する方法と、メカニズムをいくつかのグループに分類し、グループごとに適用される同値性を指定する方法が考えられる。例えば、フォーマット指示子と構文属性に対して異なった同値関係を指定したい場合には後者が望ましい。

同値性のメカニズムは、処理系が提供する文字が複数のコード化文字に現われる文字からなる場合に必要となる。ＪＩＳ　Ｘ０２０８とＡＳＣＩＩが使用される日本のＬＩＳＰ環境での英文字はそのよい例である。従って、日本国内におけるＬＩＳＰ処理系に関しては柔軟な同値性のメカニズムが強く要求される。しかしながら、ユーザが同値性を指定できるメカニズムは未知のものであり、いくつかの可能な仕様を実際の処理系に試験的に実装し、使用経験を積むことが必要であろう。

5　おわりに

ＩＣＳＨに関しては、検討すべき項目がいくつか残されている。例えば、文字の同値性のメカニズムは本文でも述べたように未知のものであり、今後の研究課題である。本稿を機会に、広く国内関係者の意見を集め、今後の検討に役立てることができれば幸いである。

本稿は、ＳＣ２２／ＬＩＳＰ　ＷＧにおける検討結果をベースにしたものである。議論に参加された同ＷＧの委員の方々に感謝したい。

参考文献

[ISO478] Proposed Draft Technical Report on: Guidelines for the Preparation of Programming Language Standards and Letter Ballot, ISO/IEC JTC1/SC 22 N478 (1988).

[ISO646] ISO R646, 6 and 7 bit coded character sets for information processing interchange, 1st edition (1967).

[ICSH] SC22/LISP WG, Technical Issues on International Character Set Handling in LISP, ISO/IEC JTC1/SC22/WG16 LISP N49 (1989).