文字知識処理環境 CHISE の基盤ソフトウェア

守岡 知彦、江渡 浩一郎

概 要

特定の符号化文字集合に依存しない自由な文字処理技術の実現を目指している CHISE (CHaracter Information Service Environment) プロジェクトで開発している文字知識処理のための基盤ソフトウェアについて述べる。このプロジェクトでは、当初、符号化文字技術に代わる文字表現方式である『Chaon モデル』(旧『UTF-2000 モデル』) を実証するために、XEmacs を基にした多言語文書編集系 XEmacs CHISE (旧 XEmacs UTF-2000) を実現した。XEmacs CHISE では高度な文字処理機能を実現することができたが、その機能の多くは XEmacs CHISE の内部世界に留まった。計算機環境全体への適用という CHISE プロジェクトの目標のためには、XEmacs CHISE の外に持って行くとともに Emacs Lisp 以外の言語処理系での利用が不可欠である。ここでは libchise をはじめとする XEmacs CHISE の機能の外部化の取り組みと Ruby/CHISE をはじめとする Emacs Lisp 以外の言語処理系への取り組みの現状と今後の展望について紹介する。

1 はじめに

CHISE プロジェクトでは文字に関するさまざまな側面を多角的にとらえようとする次世代の文字処理環境の実現を目指している。CHISE というのは “CHaracter Information Service Environment” の略で、文字に関する知識処理のための計算機環境のようなものを表している。この中核となるのは計算機環境で共有される文字データベース・システムであり、各種アプリケーションがこの文字データベースを参照して動作することにより計算機環境全体で文字に関する知識が共有されることになる。CHISE における文字処理は “Chaon” モデルに基づいている。Chaon モデルでは、符号化文字集合を用いた文字表現に代わり、文字オントロジーによって文字の性質や文字間の関係を表現することになる。このため、このモデルでは文字に関する知識や文字に対する視点や文字の置かれた文脈や状況などを特定の符号化文字集合に依存することなく表現することが可能となる。

このように、CHISE プロジェクトは、巨大な収録文字数を誇る符号化文字集合やエンコーディングを工夫した優れた文字コードを作ろうとするのではなく、符号化文字技術というもの自体に疑いの目を向け、それに代わるものを再構築することによって文字コードにまつわる問題を抜本的に解決することを目指している。

この Chaon モデルに基づく文字処理は文字オントロジーを格納する文字知識データベース・システムを用いて実現することができる。守岡が実装した最初の Chaon 実装である XEmacs UTF-2000 [7] [1] では当初 XEmacs が内蔵する Emacs Lisp 処理系の機能を用い、リスト構造によって文字知識データベースを実現していた。後には専用のデータ構造を用いて記憶効率を上げるという工夫を行ったが、それでも文字知識データベースが XEmacs の内部に閉じていることには変わらず、XEmacs の外から直接的に文字知識データベースを参照することができなかった。この問題を解決するには文字知識データベースを XEmacs の外部に持っていけば良いといえる。そして、守岡は文字知識データベースの外部化を行った XEmacs CHISE (旧 XEmacs UTF-2000) を実現した。これにより、XEmacs CHISE の文字知識データベースは異なる XEmacs CHISE のプロセス間や XEmacs CHISE 以外の Chaon 実装からも共有可能となった。そして、実際に、江渡による Ruby/CHISE と師茂樹氏による Perl/CHISE が実現された。また、守岡は XEmacs CHISE の文字知識データベース (以下ではこれを『CHISE 文字データベース』と呼ぶことにする) に関する操作を提供する libchise の開発を行い、XEmacs CHISE の機能の libchise へ移転を行っている。

本論文では、このような CHISE プロジェクトにおいて行われている XEmacs CHISE (旧 XEmacs UTF-

2000）の機能の外部化の試みと、CHISE 環境で共有される文字知識データベース（『CHISE 文字データベース』）を利用するための基盤ソフトウェアである XEmacs CHISE, libchise, Ruby/CHISE について述べる。

2 XEmacs CHISE

XEmacs CHISE（図 1）は守岡が開発した Chaon 実装で、拡張可能な多言語文書編集系 XEmacs [6] を基にしている。XEmacs CHISE に先立ち守岡が実現した XEmacs UTF-2000 では、前述のように文字知識データベースが XEmacs 内部の記憶空間に閉じていた。一方、XEmacs CHISE では文字知識データベースを XEmacs の外部に置くことが可能である。これにより、XEmacs CHISE の文字知識データベースは異なる XEmacs CHISE のプロセス間や XEmacs CHISE 以外の Chaon 実装からも共有可能となった。

図 1: XEmacs CHISE

2.1 文字

XEmacs CHISE では、Chaon モデルに基づき、文字（または、文字の集合を表す『文字状況』）を文字素性の集合によって表現している。文字素性は素性名と値の対であり、キー部が素性名を表すシンボルである cons 対で表現することができる。また、文字素性の集合は連想リストで表現することができる。この連想リストはこれによって表現される文字の性質を表現したものと考えることができる。そこで、このような連想リストを『文字指定 (char-spec)』と呼ぶことにする。

文字指定は概念的には文字であるが Lisp 的にはリストの一種であり、そのままでは XEmacs における文字型オブジェクトとは見做されないことになる。このため XEmacs CHISE では、文字指定の集合を文字データベースに格納するとともに、そこでの文字指定への参照である “char-id” を文字型オブジェクトの内部表現とし、文字データベースを介して文字素性ないしはその集合である文字指定と文字型オブジェクトを結びつけている。

以下に XEmacs CHISE が提供する文字関連の機能を示す：

関数 define-char (*char-spec*)

文字素性の集合 *char-spec* で表現される文字オブジェクトを定義し、その文字オブジェクトを返す。

[例]

```
(define-char
  '((name              . "CJK RADICAL SECOND TWO")
    (general-category symbol other)
    (bidi-category     . "ON")
    (mirrored          . nil)
    (total-strokes     . 1)
    (<-radical
     ((=ucs            . #x4E5A)
      ))
    (=big5-cdp         . #x8C5D)
    (=ucs              . #x2E83)
    ))
```

関数 get-char-attribute (*character feature* ***&optional*** *default-value*)

文字オブジェクト *character* の素性 *feature* の値を返す。

もし、値が定義されていない場合、*default-value* を返す。なお、*default-value* の既定値は nil である。

[例]

```
(get-char-attribute ?あ 'name)
→ "HIRAGANA LETTER A"
```

関数 put-char-attribute (*character feature value*)

文字オブジェクト *character* の素性 *feature* の値を *value* に設定する。

[例]

```
(get-char-attribute ?あ 'foo)
→ nil
(put-char-attribute ?あ 'foo 1)
→ 1
(get-char attribute ?あ 'foo)
→ 1
```

関数 remove-char-attribute (*character feature*)

文字オブジェクト *character* の素性 *feature* を削除する。

関数 find-char (*char-spec*)

文字素性の集合 *char-spec* で表現される文字オブジェクトを探索し、見つかった場合はそれを返す。見つからなかった場合は nil を返す。

関数 map-char-attribute (*function feature*)

文字素性名 *feature* を持つ各文字に対し、関数 *function* を適用する。

この関数は2引数の関数であり、この関数が呼ばれる時、第1引数に文字が渡され、第2引数に文字素性値が渡される。また、この関数が非-nil 値を返すと、関数 map-char-attribute の実行はその時点で停止される。

関数 char-attribute-alist (*character*)

文字 *character* に対応する文字指定を返す。

関数 char-attribute-list ()

現在までに使われた文字素性名の一覧を返す。

2.2 Coded-Charset

XEmacs CHISE は XEmacs Mule や GNU Emacs などの従来型 Mule 実装における “charset” 機能と同様な符号化文字集合 (coded character set; CCS) に関する機能を持つ。これをここでは “coded-charset” と呼ぶ。

XEmacs CHISE における文字の表現は符号化文字集合に依存していない。このため、従来型 Mule 実装におけるような意味での charset は必要ない。しかしながら、既存の文字符号の世界と情報交換したり、既存のフォントを利用したり、従来型 Mule 実装との互換性を実現し従来型 Mule アプリケーションを利用するために、何らかの形で charset に相当するものを実現することは重要である。そこで、XEmacs CHISE では符号化文字集合の抽象として coded-charset を設けている。

XEmacs CHISE の coded-charset に関する機能には以下のものがある：

関数 encode-char (*character coded-charset* ***&optional*** *defined-only*)

文字 *character* が符号化文字集合 *coded-charset* で符号化可能な時、その符号位置を返す。符号化できない時は nil を返す。

関数 decode-char (*coded-charset code* ***&optional*** *defined-only*)

符号化文字集合 *coded-charset* の符号位置 *code* に対応する文字を返す。該当する文字が存在しない場合は nil を返す。

また、XEmacs Mule における charset に関する API の上位互換であり、関数 make-char や関数 split-char, 関数 make-charset なども存在する。

なお、XEmacs CHISE では coded-charset の継承機能が導入されており、関数 make-charset の第3引数で ‘mother’ という属性を指定することにより既存の coded-charset を継承した coded-charset を定義することができる。

2.3 Coding-System

XEmacs CHISE は XEmacs Mule や GNU Emacs と同様に入出力における文字コード変換機能である “coding-system” 機能が存在する。

XEmacs CHISE の coding-system 機能は XEmacs Mule の coding-system 機能の上位互換であり、従来から存在する coding-system に加えて、UTF-8 系

coding-system や Big5 の変種などが追加されている。UTF-8 系 では UCS を表す coded-charset として、GB に基づくもの (utf-8-gb), CNS 11643 に基づくもの (utf-8-cns), JIS に基づくもの (utf-8-jis), KS に基づくもの (utf-8-ks), Big5 に基づくもの (utf-8-big5), Unicode/UCS の例示字形に基づきなるべく包摂しないもの (utf-8-mcs) が用意されている。

また、ベースとなる文字コードで表現できない文字を SGML の実体参照形式で表現する機能もあり、例えば、utf-8-gb に実体参照機能を付けたものである utf-8-gb-er や CDP 外字付き Big5 である big5-cdp に実体参照機能を付けたものである big5-cdp-er などが用意されている。

2.4 文字知識データベースの外部化

XEmacs CHISE は文字知識データベース（CHISE 文字データベース）を Berkeley DB に格納しており、文字処理をする上で必要となる文字データを最初に必要となった時に記憶空間内に読み出すことができる（これを “lazy loading” 機能と呼んでいる）。これにより、XEmacs CHISE の文字知識データベースは異なる XEmacs CHISE のプロセス間や XEmacs CHISE 以外の Chaon 実装からも共有可能となった。そして、実際に、江渡による Ruby/CHISE と師茂樹氏による Perl/CHISE が実現された。

XEmacs CHISE は CHISE 環境で共有される CHISE 文字データベースとは別に、実行中の XEmacs CHISE のプロセスに固有の文字データベースを持っている。この文字データベースはそのプロセスの外部からは参照できず、プロセスの終了とともに消滅する。2.1 節で述べた関数 put-char-attribute や関数 define-char で定義した情報はこのプロセス内部の文字データベースに記録されており、そのままでは CHISE 文字データベースに反映されない。そのため、次のような関数が用意されている：

関数 save-char-attribute-table (*feature*)

各文字の文字素性 *feature* の値を CHISE 文字データベースに書き出す。

関数 save-charset-mapping-table (*coded-charset*)

coded-charset の decoding-table を CHISE 文字データベースに書き出す。

また、プロセス内部の文字データベースにおける情報を破棄して、CHISE 文字データベースの情報を読み直せるようにするための機能も用意されている：

関数 reset-char-attribute-table (*feature*)

プロセス内部の文字データベースから各文字の文字素性 *feature* の値を消去し、CHISE 文字データベースの情報を読み直せるようにする。

関数 reset-charset-mapping-table (*coded-charset*)

プロセス内部にある *coded-charset* の decoding-table を破棄し、CHISE 文字データベースの情報を読み直せるようにする。

また、文字素性に関する情報を “lazy loading” ではなく一気に読み込むための機能も用意されている：

関数 load-char-attribute-table (*feature*)

CHISE 文字データベースから各文字の文字素性 *feature* の値を読み込む。

この他、文字素性を陽に登録するための機能も存在する：

関数 mount-char-attribute-table (*feature*)

CHISE 文字データベースから文字素性 *feature* を読み込めるようにする。

2.5 CHISE 文字データベース

Chaon モデルに基づく文字処理システムを活用するためには文字知識のデータベース化が不可欠であり、我々はこうした文字データベース・コンテンツの開発にも力を入れている。現在、CHISE 環境用に提供されている文字データベースには

1. XEmacs CHISE 附属のデータベース
2. CHISE 漢字構造情報データベース (CHISE-IDS)

の 2 つがある。

前者は XEmacs CHISE に附属する文字データベースで、XEmacs CHISE の define-char 関数を用いた Emacs Lisp プログラム（define-char 形式）で表現されている。

後者は ISO/IEC 10646-1:2000 [2] で定義された IDS (Ideographic Description Sequence) を用いて漢字構造情報を表現したデータベースと XEmacs CHISE 用関連プログラムからなるパッケージである。

この漢字構造情報というのは漢字の部品の組合せ構造に関する情報のことである。ご存知のように、多くの漢字は偏と旁などの部品の組み合わせによって構成されているが、こうした漢字の部品の組合せ構造は形の抽象的表現となるだけでなく、字義や音価にも関係しており、字源に基づく文字構造の分析は「解字」と呼ばれ、そうしたデータは重要な辞書記述の１つである。そこで我々は、CHISE 文字データベースに収録されている全ての複合（会意・形声）漢字に対し漢字構造情報を付けることを目標に、漢字構造情報データベースの開発を計画した。そして、2001 年度には未踏ソフトウェア創造事業の予算を得て、漢字の部品の組合せ構造を表現した CHISE 漢字構造情報データベース (CHISE-IDS) の開発を始めた。現在、ISO/IEC 10646-1:2000 [2] 基本統合漢字（Unicode の例示字形）, 同 統合漢字拡張 A, および ISO/IEC 10646-2 [3] 統合漢字拡張 B に対する入力を一応完了しており、これを元に JIS X 0208:1990（の例示字形)、大漢和文字などに対する編集作業も行っている。

CHISE-IDS の開発以前から存在する漢字構造情報を含んだデータベースとしては、台湾中央研究院の CDP データベースと台湾の中華電子佛典協會 (CBETA) の外字データベースがある。また、この他、日本でも「今昔文字鏡」[4] がある。CDP のデータベースと CBETA の外字データベースはそれぞれ独自形式を採っており、そのままでは IDS 形式に変換することはできない。また、今昔文字鏡のデータは一般には公開されていない。今昔文字鏡は専有的 (proprietary) ソフトウェアであり GPL によって配布されている XEmacs CHISE で利用することはできない。一方、CDP のデータベースと CBETA の外字データベースは GPL で配布されるプログラムで利用可能であるので、可能な限り変換して利用することにした。

XEmacs CHISE 附属のデータベースと CHISE-IDS パッケージは今のところ別々に配布されているが、XEmacs CHISE が利用可能な環境では、CHISE-IDS パッケージ附属のインストーラーを使って、CHISE-IDS データベースを XEmacs CHISE 附属のデータベースに統合することができる。これは前述の関数 save-char-attribute-table の応用例のひとつである。

図 2: M-x ideographic-structure-search-chars [CR] 木 金 [CR] の出力結果

また、CHISE-IDS パッケージは漢字構造情報を XEmacs CHISE で利用するためのプログラムを含んでいる。これは、現在の所、

ids.el IDS 形式の構文解析器など

ids-read.el CHISE-IDS データベースを XEmacs CHISE に読み込むプログラム

ids-dump.el XEmacs CHISE 内部で用いている ideographic-structure 形式のデータを CHISE-IDS データベース形式で書き出すプログラム

ids-util.el 漢字構造情報を Unicode 例示字体風や大漢和辞典風に変換するプログラム

ids-find.el 部品によって漢字を検索するためのプログラム

からなる。

ids-find.el は現在 ids-find-chars-including-components（別名 ideographic-structure-search-chars）と ids-find-chars-covered-by-components という２つの検索用コマンドを提供している。前者は M-x ideographic-structure-search-chars [CR] 部品列 [CR] で指定された部品列の各部品を少なくとも１個は含んでいる漢字の一覧を表示するコマンドである（図 2)。後者は M-x ids-find-chars-covered-by-components [CR] 部品列 [CR] で指定された部品列中の任意の部

品を０回以上用いて全体として２個以上の部品から構成される漢字の一覧を表示するコマンドである（図 3）。

図 3: M-x ids-find-chars-covered-by-components [CR] 木金 [CR] の出力結果

3 Ruby/CHISE

Ruby/CHISE は江渡によって実現された Ruby [5] における Chaon 実装である。CHISE の文字データベースを利用する機能を中核とし、漢字構造情報を扱うための機能や文字データベースの可視化のため作成されたいくつかの関連ライブラリが存在する。

3.1 Character class

Ruby/CHISE は Chaon モデルにおける文字オブジェクトを Character クラスのオブジェクトとして扱っている。この点では、オブジェクト指向言語ではない Emacs Lisp に基づく XEmacs CHISE における文字オブジェクトよりも Chaon モデル的な言語インターフェースを与えていると考えられる。

以下に、Ruby/CHISE におけるプログラム例を示す:

```
require 'chise'
include CHISE
p "字"                      # "字"
p "字".ucs                  # 23383
p "字".total_strokes        # 6
p "字".inspect_all          # 保有する素性情報を全て出力する
p "字".daikanwa             # 6942（大漢和番号）
```

つまり文字そのものが自分の持つ素性を知っていて、そこにメソッドとしてアクセスできる。

上記の例における "字" は実際には String オブジェクトであるが、Ruby/CHISE は１文字のみからなる文字列を自動的に Character オブジェクトに変換する。なお、

```
p "字字".ucs # 例外
```

のように１文字ではない文字列に対して文字素性を求めようとする場合には例外が発行される。

また、Character class のオブジェクトを生成し、それを直接用いることもできる。

```
# Character class のインスタンスを生成
char = "字".char

# 上記とまったく同じ
char = Character.get("字")
```

Ruby/CHISE では flyweight パターンを用いているので、同じ字の場合は同じインスタンスとなる。

3.2 実体参照

```
# 数値参照化する
p "字". to_er       # "&#x5b57;"

#数値参照を復号
p "&#x5b57;".de_er # "字"

# JIS X 0208-1990 による実体参照を復号
p "&J90-3B7A;".de_er

# 大漢和番号による実体参照を復号
p "&M-06942;".de_er
```

このように、実体参照の復号処理が可能となっている。

3.3 String の拡張

String#each_character
: String の一文字毎にイテレータを実行。引数として Character が入る

String#map_character
: 同様に map した結果の文字列を返す

String#char_length

UTF-8 的な文字列の長さを返す。

他にも to_er, de_er など Character と共通の要素を各文字毎に摘要するためのメソッドがいくつか用意されている。

3.4 素性の定義

```
"木".mydepth = 1
"林".mydepth = 2
"森".mydepth = 3
```

実行すると自動的にデータベースに保存されるため、プログラム終了後も素性は保存される。

```
p "木".mydepth
p "林".mydepth
p "森".mydepth
```

このようにして、自分独自の必要な素性を作り、その素性に基づいたプログラムを書くことができる。

3.5 漢字構造情報関連

Ruby/CHISE には、CHISE 文字データベースを扱うための一般的な機能に加え、漢字構造情報（字形分解・合成）を扱うための機能が強化されている。

3.5.1 解字

String には decompose, decompose_all という 2 つのメソッドがある。decompose は一段階だけ分解する。decompose_all はそれを再帰的に行う。

```
p "字".decompose
p "字".decompose_all
p "榊".decompose
p "榊".decompose_all
p "終了".decompose
p "終了".decompose_all
p "鬱".decompose
p "鬱".decompose_all
```

3.5.2 部品による文字探索

IDS 形式で表現された漢字構造情報に対応する文字オブジェクトを生成するには compose メソッドを用いる。

```
p "#x2ff0;木木".compose
```

find メソッドを用いることである部品が漢字の一部として含まれている漢字群を探すことができる。

```
p "日雲".find
```

ここでは、日と雲という部品が含まれている全漢字を出力する。

4 libchise

XEmacs CHISE や Ruby/CHISE, Perl/CHISE といった Chaon 実装は Chaon モデルに基づいて文字を処理するという点と同一の CHISE 文字データベースにアクセスするという点で極めて似通った部分を持っており、実際に同様な処理を行っている。しかしながら、こうした部分の部品化がなされていなかったために、各言語処理系はそれぞれ別々に実装する必要があった。これは実装効率とデータ形式の一元管理の点で問題が多いので、こうした部分のライブラリ化を行うことになった。そうして誕生したのが libchise である。

4.1 目標

libchise の開発は CHISE 環境で共有される機能をライブラリ化することを目標としている。これは言い替えれば XEmacs CHISE の機能を徐々に libchise 側に移して行くことだといえる。この際、問題となるのが複雑なデータの扱いである。

現在の CHISE 文字データベースでは Lisp を前提とした複雑なデータ構造が使われている部分がある[1]が、こうしたものを Lisp 以外の環境で扱うのは容易ではない。もちろん、そうしたものを libchise の中で隠蔽し、libchise を利用するプログラムに提供することは可能であるが、記憶管理や実装の手間を考慮すればな

[1] XEmacs CHISE は文字素性値として任意の Lisp オブジェクトを許している。

るべく複雑なデータ構造を使わないことが望ましいといえる。

しかしながら、こうした複雑なデータ構造が現れる背景には、現在の文字使用状況やこれまでの文字使用の歴史の複雑さがあり、こうした複雑さを切り捨ててまで単純化することは望ましくない。少なくとも、複数の解釈やその出典を十分に記述できることが望ましい。

その一方で、そうした詳細なメタデータを必要としない場合も少なくない。こうしたことを鑑みれば、CHISE環境は用途に応じた適切な複雑さを提供すべきであるといえる。そこで、我々は表 1 に示すような階層モデルを考えている。

第 4 層	オブジェクト層 文字に対する抽象的なサービスの提供 （関数、message 送信式、TopicMaps 等）
第 3 層	構造データ層 複雑なデータ構造の実現（S 式、XML 等） （記憶管理や型システムの実現）
第 2 層	文字層 素性名による文字データへのアクセスの実現 （C の文字列レベルに限定、記憶管理無し）
第 1 層	データソース層 （Berkeley DB や PostgreSQL などの データベース・システム等）

表 1: CHISE 階層モデル

現状の libchise は第 2 層の基本部分が実装されており、当面は第 2 層までの実装を目標としている。

4.2 Data Source

“Data Source” というのは CHISE 第 1 層を抽象化したもので、文字データを格納するためのストレージを抽象化したものである。

型 CHISE_DS

Data Source を表現する構造体。

CHISE_DS*
CHISE_DS_open (CHISE_DS_Type *type,*
char **location,* DBTYPE *subtype,* int *modemask*)

Data source を開き、その管理情報を返す。

type は data source の種類を表す。現在の所 CHISE_DS_Berkeley_DB のみが指定可能である。

location は data source の場所を指す。CHISE_DS_Berkeley_DB の場合、データベース・ファイルのファイル名を指定する。

subtype は data source 中の各素性の表現方法の種類の既定値を表す。その解釈は *type* に依存する。

modemask は data source 中の各素性のパーミッションの既定値を表す。その解釈は *type* に依存する。

処理が失敗した場合、NULL が返る。

int **CHISE_DS_close** (CHISE_DS **ds*)

ds で示される data source を閉じる。

処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

4.3 Feature

“Feature” というのは文字素性名（あるいは、素性名を同じくする文字素性の集合）を表すものあり、第 2 層における文字素性を記録するための表へのポインターとして実現されている。

型 CHISE_Feature

文字素性名を表現する型。

CHISE_Feature
chise_ds_get_feature (CHISE_DS **ds,*
const unsigned char **feature*)

Data source *ds* において、文字列 *feature* で指定される文字素性を開き、その管理情報を返す。

処理が失敗した場合は NULL を返す。

int
chise_char_load_feature_value
(CHISE_Char_ID *char_id,* CHISE_Feature *feature,*
CHISE_Value **valdatum*)

文字 *char_id* の素性 *feature* の値を *valdatum* で指される場所に格納する。

処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

unsigned char*
chise_char_gets_feature_value
(CHISE_Char_ID *char_id,* CHISE_Feature *feature,*
unsigned char **dst,* size_t*size*)

文字 *char_id* の素性 *feature* の値を *dst* で指される場所に格納するとともに、C の文字列として返す。

size は *dst* で指される記憶領域の大きさを示す。

処理が失敗した場合は NULL が返る。

int
chise_char_feature_value_iterate
(CHISE_Feature *feature,*
int (**func) (CHISE_Char_ID char_id,*
CHISE_Feature feature,
*CHISE_Value *valdatum)*)

文字素性名 *feature* の値を持つ各文字に対し、関数 *func* を呼び出す。この時、その関数の第 1 引数には文字 ID, 第 2 引数には文字素性名、第 3 引数には素性値が渡される。

関数 *func* が 0 以外の値を返すと、その時点で処理は終了する。

4.4 CCS

"CCS" というのは符号化文字集合の抽象である。

型 CHISE_CCS

符号化文字集合を表現する型。

CHISE_CCS
chise_ds_get_ccs (CHISE_DS **ds,*
const unsigned char **ccs*)

Data source *ds* において、文字列 *ccs* で指定される CCS を開き、その管理情報を返す。

処理が失敗した場合は NULL を返す。

CHISE_Char_ID
chise_ccs_decode (CHISE_CCS *ccs,* int *code_point*)

符号化文字集合 *ccs* において符号位置 *code_point* に対応する文字の char-id を返す。

処理が失敗した場合は -1 を返す。

int
chise_ccs_set_decoded_char (CHISE_CCS *ccs,*
int *code_point,* CHISE_Char_ID *char_id*)

符号化文字集合 *ccs* において符号位置 *code_point* に対応する文字の char-id を設定する。

処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

int **chise_ccs_sync** (CHISE_CCS *ccs*)

符号化文字集合 *ccs* に関する（書き込み用）キャッシュをストレージ上に書き出す。

処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

4.5 その他

型 CHISE_Char_ID

文字オブジェクトの ID を表現する型。

型 CHISE_Value

結果の文字列を格納するための構造体。

int **chise_value_size** (const CHISE_Value **s*)

CHISE_Value に格納された文字列の長さを返す。

char * **chise_value_data** (const CHISE_Value **s*)

CHISE_Value に格納されたデータを返す。

char *
chise_value_to_c_string (const CHISE_Value **s*)

CHISE_Value に格納されたデータを C の文字列として返す。

5 おわりに

CHISE プロジェクトのこれまでのあゆみと現状、そして、今後の展望について、CHISE プロジェクトの基盤システムに焦点を当てて紹介した。このプロジェクトでは、当初、XEmacs に基づく Chaon 実装である

“XEmacs CHISE” の開発が先行した。後に、Chaon モデルに基づく文字処理機能を計算機環境全体への適用するために、XEmacs CHISE の内部にあった文字知識データベースを XEmacs CHISE の外部に置くことが必要となり、Berkeley DB を用いた『CHISE 文字データベース』とその XEmacs CHISE からの利用が実現された。そして、Ruby に基づく Chaon 実装である “Ruby/CHISE” や Perl に基づく Chaon 実装である “Perl/CHISE” が実現され、CHISE 文字データベースを操作するプログラムを Emacs Lisp, Ruby, Perl で記述できるようになった。

3 節で述べたように、Ruby/CHISE は CHISE 文字データベースを扱うための機能を一通り備えており、これらは Ruby が提供するオブジェクト指向の枠組を用いて Chaon モデルを自然に実装しているといえる。

また、4 節で述べたように、CHISE 文字データベースを操作するための基本的な処理をまとめたライブラリである libchise の開発もはじまり、C でも記述可能となりつつある。そして、XEmacs CHISE や Ruby/CHISE を libchise を用いて書き直す試みも行われている。

しかしながら、Ruby/CHISE や libchise は現在の所 CES 関連の機能を欠いている。Ruby/CHISE に関しては暫定的な Shift_JIS と EUC-JP の相互変換を備えているが、libchise レベルでの実装は存在していない。XEmacs CHISE （あるいは、各種 Mule 実装）の有用性の１つが coding-system であり、未だ Chaon モデル的なテキスト交換形式が十分に整っていない現状を考えれば、こうした機能を libchise レベルで実装することは重要な課題である。よって、今後、CHISE 文字データベースと連係した CES 機能を提供したいと考えている。

また、CHISE 文字データベースにグリフ関連情報を収録するとともに、グリフ管理機能を整備し、グリフ合成システムやレンダリング・エンジン、各種ツールキット等と連係して、定義した文字が自由に表示できるようにすることも重要な課題の１つである。

なお、CHISE プロジェクトの情報は

- http://cvs.m17n.org/chise/
- http://kanji.zinbun.kyoto-u.ac.jp/projects/chise/
- http://mousai.as.wakwak.ne.jp/projects/chise/

で公開されている。これらの WWW 頁群を含め、各種成果物は CVS で管理されており、最新の開発状況を知ることができる。日本語用と英語用の２つのメーリングリストも用意されており、参加方法は上述の WWW 頁で説明されている。CHISE プロジェクトに興味を持たれた方は、是非、お気軽に御参加願いたい。

参考文献

[1] bit 別冊「インターネット時代の文字コード」, 第 9 章「文書編集系における文字コード」. 共立出版, 2001.

[2] International Organization for Standardization (ISO). *Information technology — Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane (BMP)*, March 2000. ISO/IEC 10646-1:2000.

[3] International Organization for Standardization (ISO). *Information technology — Universal Multiple-Octet Coded Character Set (UCS) – Part 2: Supplementary Planes*, November 2001. ISO/IEC 10646-2:2001.

[4] 今昔文字鏡. http://www.mojikyo.com/.

[5] The object-oriented scripting language Ruby. http://www.ruby-lang.org/.

[6] XEmacs. http://www.xemacs.org/.

[7] 守岡知彦. UTF-2000 — 汎用文字符号に依存しない文字表現系の展望. アジア情報学のフロンティア — 全国文献・情報センター人文社会学学術セミナーシリーズ No.10, 全国文献・情報センター人文社会学学術セミナーシリーズ, 第 10 巻, pp. 13–24, 2000.