3K-2

プログラミング言語 Ruby のための Unicode 正規化ライブラリの実装と評価

野島 歩 † 水野 達也 †† 塩澤 元 †† 松原 俊一 † Martin J. Dürst †

† 青山学院大学理工学部情報テクノロジー学科
†† 青山学院大学大学院理工学研究科理工学専攻

1 はじめに

地域に依存しない国際的な文字コードとして Unicode は今や欠かせない．Unicode では同じ表示の文字でも数通りの内部表現を一部許している．異なる内部表現が混在すると，テキストを処理する際に不都合が生じる．Unicode はこの問題を解消するために Unicode 正規化 [2] を定めている．例えば，国際化ドメイン名において見た目は同じでも異なるドメイン名が存在すると唯一性が失われてしまう．ゆえに，国際化ドメイン名では前処理または確認として Unicode 正規化を行う．

本研究では，プログラミング言語 Ruby 用の Unicode 正規化ライブラリを実装する．さらに既存の正規化ライブラリにはない機能として，文字の種類による互換分解の範囲を指定した文字列の正規化を図る．また，W3C (World Wide Web Consortium) で推奨されている [3] 正規化形式 NFC による変換の高速化を実現する．

2 Unicode 正規化

2.1 結合文字と合成済み文字

Unicode の主な特徴に，複数の文字を結合して一つの文字を構成する表現方式が一部存在する．

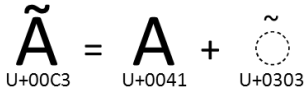

図 1: 合成済み文字と文字の合成

例えば図 1 のように Ã という文字は， Ã という一つの符号位置で表す方式と，A という文字に合成用のチルダ ˜ (U+0303) を合成して表す方式がある．後者の場合，A を基底文字，˜ を結合文字という．

Implementation of a Unicode Normalization Library for Ruby

Ayumu Nojima †, Tatsuya Mizuno ††, Hajime Shiozawa ††, Shunichi Matsubara † and Martin J. Dürst †
†Department of Integrated Information Technology, College of Science and Engineering, Aoyama Gakuin University
††Graduate School of Science and Engineering, Aoyama Gakuin University
{nojima, hajime, mizuno}@sw.it.aoyama.ac.jp, {matsubara, duerst}@it.aoyama.ac.jp

このように表示は同じでも，内部表現が異なるものが混在すると混乱が生じてしまう．しかし，Unicode 正規化を用いれば内部表現は統一できる．

2.2 互換分解可能文字

Unicode には，文字が持つ意味は等しいが幅や形，位置，角度など視覚的に異なる文字や，文字列を一つの符号位置にまとめた文字が収録されている．これらの文字は互換分解可能文字といい，全部で 16 種類に分けられている．

表 1: 互換分解可能文字の例

例	種類	正規化後
①, ⓐ	CIRCLE	1, a
ｱ	NARROW	ア
㍍	SQUARE	メートル

互換分解可能文字の例を表 1 に示す．互換分解可能文字は正規化の種類によっては，対応する一般的な文字または文字列に置換される．

2.3 正規化形式

Unicode 正規化は，文字列表現の統一方法として四つの正規化形式 NFD, NFKD, NFC, NFKC を定めている．NF は「Normalization Form」, D は「Decompose」(分解), C は「Compose」(合成)，K は「Kompatibilty」(互換) を各々意味する．「Kompatibilty」は「Compose」の C との重複を避けるため，頭文字が K になっている．

表 2: 正規化形式

形式	処理
NFD	正規分解
NFKD	互換分解
NFC	正規分解の後，正規合成
NFKC	互換分解の後，正規合成

各正規化形式の処理は表 2 のようになる．NFD, NFKD において合成済み文字は分解され，NFC および NFKC で結合文字は合成される．また，互換分解可能文字は NFKD，NKFC で正規化される．

3 実装

3.1 UnicodeData

Unicode 正規化の実装において，分解および合成に用いるマッピングは欠かせない．これらのマッピングの生成に必要なのが，UnicodeData [1] である．UnicodeData はすべての Unicode 文字の情報が記載されているテキストファイルである．UnicodeData が持つ情報のうち，正規化に用いるのは分解マッピングと正規結合クラスである．分解マッピングは，Unicode の各文字が分解可能か否かを示す．そして可能ならば分解後の文字列が明示されている．一方，正規結合クラスは結合文字の種類を示す値である．本研究では，正規表現を用いてこれらの情報のみ抽出し，符号位置をキーとしたハッシュに記憶するようにした．

3.2 正規順序

一つの基底文字に二つ以上の結合文字が結合している時, 結合文字の順序を一意に定めなければいけない．このときに用いる順序を正規順序 (Canonical Ordering) という．正規順序の実装には, 正規結合クラスを用いる．

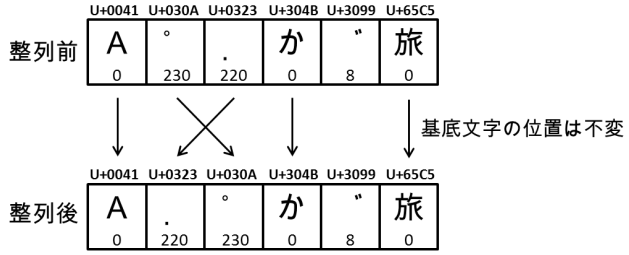

図 2: 正規順序への整列

図 2 に正規順序の例を示す．文字の下の数字が正規結合クラスである．正規順序では，各基底文字に対する結合文字は正規結合クラスの小さい順に並び，正規結合クラスが等しい文字の順番は変わらない．基底文字の正規結合クラスは 0 で，整列の前後で位置は変わらない．したがって，これらの条件を満たすため安定ソートを用いた．結合文字は多くても 10 文字以上続くことは無いのでバブルソートを用いて実装した．

3.3 ハングルの分解と合成

Unicode 正規化では，ハングルのみマッピングを使わずにアルゴリズム的に分解，合成が可能である．アルゴリズムを用いると分解マッピングや合成マッピングを生成する必要が無いので，メモリの使用量を大幅に削減できる．本研究では，Unicode の仕様に準拠して Ruby で実装した．

3.4 正規化メソッドの概要

本ライブラリが提供するメソッドとして，文字列を指定した正規化形式で正規化することができるメソッド `normalize` を実装した．正規化形式を指定しない場合は NFC で正規化される．また，本ライブラリは文字列が指定した形式で正規化されているかどうかの判定を行うメソッド `check_nf` も提供する．

3.5 種類別互換分解オプション

本研究では，互換分解可能文字の種類に応じた互換分解を実装した．互換分解可能文字の種類はメソッドのオプションとして指定する．

表 3: “㌔” を入力した際のオプションの使用例

オプション	出力
なし	“1 メートル”
CIRCLE	“1 ㍍”
SQUARE	“①メートル”
CIRCLE | SQUARE	“1 メートル”

表 3 は，オプションの使用例である．オプションを指定しない場合はすべての互換分解可能文字を正規化する．オプションはビットごと OR 演算子を用いて複数指定も可能である．

3.6 NFC の効率化

既存の正規化ライブラリは本来 NFC で正規化する必要が無い文字も含めてすべての文字に対して合成処理を試みるので効率が悪い．実際に，現在 Unicode は 109449 文字収録しているのに対し，合成パターンはハングルを除いて 931 種類しかない．

そこで本ライブラリは，`check_nf` を用いることにより，正規化する必要のない文字を合成処理の前に判定し，必要以上の処理はしない．

4 まとめ

本研究は Ruby における国際化の一環として Unicode 正規化を実装した．本ライブラリは Unicode のテストをすべてパスしており，Unicode 6.0.0 に収録している全文字に対応している．

参考文献

[1] Mark Davis and Ken Whistler. Unicode Character Database. Unicode Standard Annex, The Unicode Consortium, 2010. Unicode 6.0.0.

[2] Mark Davis and Ken Whistler. Unicode Normalization Forms. Unicode Standard Annex, The Unicode Consortium, 2010. Unicode 6.0.0.

[3] François Yergeau, Martin J. Dürst, Richard Ishida, et al. Character Model for the World Wide Web 1.0: Normalization. W3C Working Draft, The World Wide Web Consortium, 2005.