Rubyにおける文字コード変換の最適化

水野 達也，松本 章代，Martin J. Dürst
青山学院大学理工学部情報テクノロジー学科

1 はじめに

本研究室では，3年前からプログラミング言語Rubyの国際化の一環として文字コード変換の研究に取り組んできた。専用のテーブル構造と小さいインストラクションセットに基づいて，多くの文字コードの変換を実装した[1]。しかし，文字コード変換における実行時間と使用するメモリに軽減の余地が残っている。本研究では膨大な文字コードGB18030を例として文字コードの最適化を行った。

2 GB18030

2.1 GB18030とは

GB18030は中華人民共和国（中国）が制定した文字コードの国家規格である。この文字コードはGB2312-80およびその拡張であるGBKをベースに，GB2312-80とGBKの互換性も維持している。この文字コードは，2000年3月17日に国家質量技術監督局により最初に発表された。現行の版は，国家質量監督検験検疫総局と中国国家標準化管理委員会によって2005年11月8日に発布され，2006年5月1日に実施されている。この規格のサポートは中国で販売されるすべてのソフトウェア製品に対して義務付けられている。

2.2 GB18030の構造

GB18030はASCII，GB2312-80を含むGBK，Unicode，将来拡張部の部分で構成されている。GB2312-80は94 × 94種類の文字を表す。ASCIIと重ならないために，2バイト文字コードである。バイトの範囲として1バイト目，2バイト目ともに`A1`から`FE`までを使用する。GB2312-80以外のGBKは1バイト目は`0x81`〜`0xFE`，2バイト目は`0x40`〜`0xFE`（`7F`は除く）でUnicodeからコードポイントを選択してUnicode番号に従って追加されている。GB18030での拡張は4バイト文字を追加し，GBKで対応してなかったUnicodeの文字をすべて追加した。4バイト文字は第1，3バイト目は`0x81`〜`0xFE`，第2，4は`0x30`〜`0x39`で表現する。このような拡張方法をとっているので，各バイトで対応する文字領域は決まっている（表1参照）。またUnicodeとの対応関係もこの拡張方法によって半規則的になっている。

表1: GB18030の主な文字領域とバイト数

文字	文字数	バイト数
ASCII	128	1バイト
GB2312-80	7,445	2バイト
GB2312-80以外のGBK	16,495	2バイト
Unicode BMP	50,400	4バイト
Unicode 第1-16面	1,058,400	4バイト
将来拡張	478,800	4バイト

3 Rubyの文字コード変換の仕組み

Rubyでの文字コード変換の実装は簡単な専用のインストラクションセットによる。変換に必要なデータはRubyのコンパイル時にオプコードを含めて4バイト長のインストラクションに変更される。実行時には入力1バイトごとに1個のインストラクションが評価され，それによって文字コードの変換が行われる。さらに，バイト値にかかわらず同じインストラクションが使われることが多いため，バイト値からインストラクションを選択する処理は2段階のテーブル参照によって行われる[2]。

1つ目のテーブルはoffsetsといい，バイト値256種類の長さと幅1バイトで構成され，次のテーブルへのインデックスとして使われる。

2番目のテーブルはinfosといい，インストラクションが格納されている。インストラクションの例としては変換結果の出力，入力の変更なしの出力，入力のバイト列の無効，変換の未定義，規則的な変換のための関数の呼び出しなどを指定するものがある。2つのテーブルの一例を図1に示す。

4 インストラクションの実装

本研究で実装したインストラクションは，GB18030とUnicode間の文字コードの差を利用することによって変換を実現するものである。GB18030とUnicodeはGB18030の由来から半規則的である。非BPMに相当

Optimization of Character Code Conversion in Ruby
Tatsuya MIZUNO, Akiyo MATSUMOTO and Martin J. DÜRST
Department of Integrated Information Technology, College of Science and Engineering, Aoyama Gakuin University
5-10-1 Fuchinobe, Sagamihara, Kanagawa 229-8558, Japan
mizuno@sw.it.aoyama.ac.jp, {akiyo, duerst}@it.aoyama.ac.jp

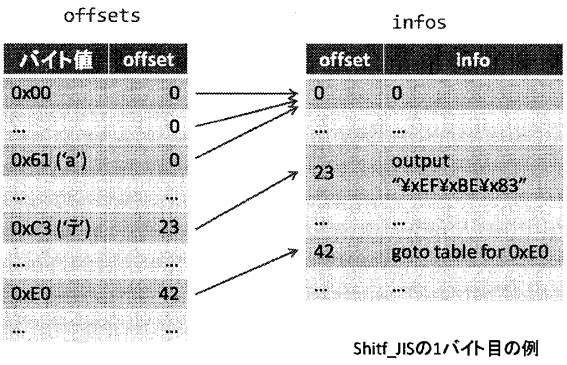

図 1: 文字コード変換二つのテーブル

する部分は完全に規則的であるので，GB18030 のコードとオプコードによって UTF-8 に変換している [1]。しかし，局所的に連続しているところにそれぞれ関数を適用すること [3] は現実的でない。そこで，連続している箇所において GB18030 と Unicode の差が一定になることを利用し，差をインストラクションのデータにし，それを引数として変換関数を呼ぶようにした。

本研究ではインストラクションの効果が期待できる条件でのみ導入した。1 つ目は GB18030 が 2 バイトで UTF-8 が 3 バイトの場合に適用する。2 つ目は GB18030 が 4 バイトで UTF-8 が 3 バイトの 2 つの場合について適用する。その結果によってテーブルの大幅な削減が実現された。

差は GB18030 が 2 バイトで UTF-8 が 3 バイトの場合はそのまま使用する。GB18030 が 4 バイトで UTF-8 が 3 バイトの場合は，GB18030 の構造を踏まえて数値を線型にしてから差をとった。さらに 2 つの場合共に求められた差を Ruby で利用するうえで適当な値に調整したものを利用している。

5 評価

本研究では以下の条件を設定する。条件 1 としてすべてにインストラクションを適用する。条件 2 として 2 つ以上連続した場合にのみインストラクションを適用する。条件 3 としてインストラクションを全く適用しない。この 3 つにテーブルサイズ，実行時間において評価を行う。

評価の対象としたものは文字コード変換の際にメモリを使用する gb18030.c および gb18030.o のファイルのサイズである。加えて各条件における中国語の長文を UTF-8 から GB18030 に変換するテストの実行時間である。テストは 5 回繰り返した平均をとっている。

実行環境は Windows XP, *Intel*® *Xeon*® 32bit CPU 2.66GHz, Cygwin gcc -g なし -o なしである。

各条件で Ruby をコンパイルした結果，インストラクションを適用することによって.c ファイルおよび.o ファイルのファイルサイズは大幅に減少した。テストの実行時間は，条件 3 ＞条件 2 ＞条件 1 となり，関数を使用することによる実行時間の増加が認められた (図 2 参照)。

表 2: 各条件におけるサイズとテスト結果

条件	gb18030.c	gb18030.o	テスト
条件 1	951 KB	194 KB	25.3940 秒
条件 2	963 KB	196 KB	23.6124 秒
条件 3	3,965 KB	602 KB	19.0688 秒

6 おわりに

本研究ではプログラミング言語 Ruby の国際化の一環として，変換効率の向上に取り組んだ。その一例として中国の国家規格である膨大な文字コードの GB18030 に対し行った最適化を取り上げた。GB18030 とユニコードの変換テーブルは半規則的で，先行研究ではそれに少数の関数の場合分けで対応した。我々は規則的部分でコードポイント同士の差が一定であることに注目し，新しいインストラクションの導入により今まで以上にテーブルの削減に成功した。

本研究によって実装されたインストラクションは最新の Ruby1.9.2 において公開済みである。

7 今後の課題

本研究では GB18030 を対象として最適化を行った。しかし，他の文字コードを含め文字コード変換をコンパクトにし，実行時間をより速くするという改善点が残っている。

参考文献

[1] 神林義博, 松本章代, Martin J. Dürst. Ruby における文字コード変換環境の向上. 情報処理学会第 71 回全国大会, 2009

[2] 成瀬ゆい, Martin J. Dürst. Ruby m17n. http://www.sw.it.aoyama.ac.jp/2008/pub/RubyKaigiM17N.html

[3] GB18030: A mega-codepage, 2001. http://web.archive.org/web/20080129171631/http://www.ibm.com/developerworks/library/u-china.html から入手可能