UNIX における日本語文書読み上げソフトの開発について

平成 12 年 2 月 4 日

情報電子工学科 竹野研究室
佐藤 健美

目 次

1 はじめに **1**

2 作成する日本語文書読み上げソフトについて **1**
2.1 UNIX とフリーソフトウェア 1
2.2 ソフトに求める品質 2
2.3 視覚障害者の使い道 2
2.4 晴眼者の使い道 3

3 全体の流れ **4**
3.1 kakasi の特徴 4
3.1.1 kakasi の変換速度 5
3.2 音声ファイルの出力 6
3.2.1 音声出力 7
3.2.2 音声データの連結 7

4 kakasi の問題点 **8**
4.1 kakasi のオプション 8
4.2 kakasi による処理の流れ 8
4.3 機能的な問題 8
4.4 文法的な問題 10
4.5 漢字の変換精度 10
4.5.1 難読漢字 10
4.5.2 複合語 11
4.5.3 長文読解 12

5 変更点と結果 **14**
5.1 分割モード 14
5.2 文字変換処理 16
5.3 変更後のプログラムの流れ 17
5.4 実行結果 19

6 まとめ **21**

参考文献 **22**

概要

現在、 UNIX 上で動作する日本語文書読み上げソフトが普及していないため、これをフリーソフトとして配布できるようにしようと思いその作成を考えた。そして、 UNIX における日本語文書読み上げソフトの開発を行っていく為に、今回は kakasi という漢字や仮名の混じった文章をローマ字やひらがなのみの文章に変換するソフトを使用することにした。しかし、 kakasi を今回のソフトで使用するには問題があるので、修正をしなければいけない部分がある。この論文では、 kakasi を修正するうえで調べた結果や、実際にソフトを作成していく為に必要なプロセスや問題点を報告する。また、修正を行った後の実行結果を見てどのように変わったか、その後にでてきた問題点について考察する。

1　はじめに

近年、企業や学校など様々な場所でコンピュータが使用されるようになっている。そして、身体に障害を持つ人もコンピュータを使用するケースが多くなってきているが、その中でも特に、視覚障害者がコンピュータを使うのは困難である。彼等が使う OS は、MS-DOS や UNIX といったテキストベースの物がほとんどであり、 MS-DOS を使用するユーザが数多く残っている。東京都内にある視覚障害者のためのパソコン教室では、NEC 社製の PC-98 用 MS-DOS を使用しているというような実例もある。

ディスプレイを見ることのできない人にとって音は重要なものであり、それはどの分野においても必要なものであるが、それに応じたコストもかかってしまう。そこで、コンピュータを使用する人の手助けになれば良いと思い開発を行うことにした。しかし、UNIX は MS-Windows の様に音声出力が OS で統一されていないため、日本語文書を音声化して出力できるフリーソフトが普及していない。MS-DOS では、株式会社リコーの「VC2」というフリーソフトが、 MS-Windows では、同じくリコーの「雄弁家 for Windows」という市販のソフトが出ている。UNIX 用として、例えば富士通から市販されているものなどがあるが、視覚障害者にとって使いやすいものであるとは言い難い。

今回目標とするソフトは、ひらがなに変換された文書に 1 つ 1 つ音声データを割り当て、それを連結して出力させるという方法で行う。この方法ではあまり音声の品質はよくないが、今回の主な使い道を考えると、あまり高い品質のものは必要無いと考え、このような形で作成を行う。そして、修正を行うときはユーザ自身が音声データも変更を行えるようにする等の処理を行い、最終的にはフリーソフトとして配布することを目標とする。

2　作成する日本語文書読み上げソフトについて

2.1　UNIX とフリーソフトウェア

フリーソフトウェアには、

- 自由に配布できる
- 無料で入手できる
- ソースが公開されていて、ユーザがそれを改良できる
- 色々なユーザが改良、開発に参加できる

というようなメリットがある。このようなソフトウェアを PDS (Public Domain Software) と言う。しかし、著作権によって保護されており、いくつかの制限のある項目を持つソフトもある。

また、フリーソフトは無料で使用できるかわりに、デメリットもある。例えば、ソフトの使用に際しては、すべて自分で責任を取らなければならない。例えば、ソフトの暴走によってファイルが破壊されても誰にも文句は言えない。そして、当然メーカーのサポートもないため、自分で処理を行わなければならなくなる。

UNIX において、フリーソフトウェアの位置づけは非常に高い。もともと UNIX は、最初に作られたときはそれ自体がフリーソフトウェアの一種であったこともあり、また、ユーザの多くが研究者、プログラマーであることなどからフリーソフトウェアの文化が根強く残っている。また、パソコン上で動作するフリーの UNIX が存在し、その入手、インストールも容易に行える。そして、その上で動作するソフトも多種多様で、最新のものが容易に入手できる。

プログラムをフリーソフトウェアとして配布すると、色々なユーザに使ってもらうことで忌憚のない意見が得られる、ソースレベルでのデバッグ、改良に、多くのユーザに参加してもらえる、などのメリットもある。

2.2 ソフトに求める品質

今回作成するソフトは、晴眼者向けという方向で作成しているが、いずれは視覚障害者にも使いやすいものへ修正できるようにすることを目標とする。本来、このようなソフトの開発を行うときは規則音声合成で音声データを作成したほうが品質のよいものができる。しかし、今回はあらかじめ録音した音声データを用いて作成する。これは、今回の主な使い方である文書やデータの読み合わせには、あまり高い品質のものは必要無いと考えたためである。また、品質を上げるとソフトの軽快性が失われる恐れがあるため、このような形で作成する。そして、いずれは使用者が自分の声を録音して使えるように修正を行い、データ、質などに不満があるとき、ユーザが自分で改良できるようにする。

2.3 視覚障害者の使い道

ここでは、視覚障害者にとってどのような使い方があるかを報告する。

視覚障害者にとって、コンピュータを使うことが非常に困難であるということは前に記した。そういう人達にとって、音声によるガイドというものは非常に重要になってくる。しかし、その目的によってどの程度のものが必要か、またどのような方法でプログラムを動かすのが最適か等、さまざまな問題がある。そこで、次にいくつかの具体例をあげ、最適な実行方法を考えてみる。

1. プロンプト上で打ったコマンドを読み上げる
2. コマンドを実行して表示された内容を読み上げる
3. テキストファイルを読む

まず、 1. や 2. のような使い方をする場合、このソフトはシステム上に常駐した状態にしておくのが最も適していると思われる。もちろん 3. も常駐した状態で使用してもそれほど問題はないと思われるが、常駐した状態であるということは常にシステムに負担をかけているということになる。そのため、 3. の使い方をする場合は常駐させておくよりも、使いたいときにその場で実行するほうが適していると思われる。

このようなことを考えた場合、視覚障害者がこのソフトを使用するときはシステム上に常駐しているほうが使用の幅も広がり便利である。今回作成しているソフトは常駐して使うようにはしていないが、いずれは視覚障害者でも使いやすいものになるように修正していくことを目標とする。

2.4 晴眼者の使い道

晴眼者にとって、このソフトはその状況、目的によってはあまり必要ではないかもしれない。しかし、例えば二人で行った方がはかどるような作業を一人でやらなければいけない場合、二つの作業を並行して行わなければならない場合など、その内容によってはこのソフトが役立つ場合もある。そのような作業の負担を軽減できるようにするのが今回作成を目標とするソフトである。

ここで、使用例を次に示し、具体的な使い方を説明する。

1. テキストエディタへの応用
2. 文書の読み合わせ
3. 学校、企業などのプレゼンテーションの場で使用
4. インターネットの音声ガイド
5. データの整理などをする時の助手的な役割
6. メールを読む、又は送信する時の確認
7. アプリケーションの起動や終了の確認
8. スキャナで取り込んだ手書きの文章などのチェック

1. は、常駐して 1 文ごとに音声出力させる方法や、ある程度の文書を作成してそのデータをソフトで音声出力させる方法のような使い方が考えられる。このことで、文書作成中の誤字脱字を減らす手助けができれば良いと考えた。

2. の文書の読み合わせは、内容的に 1. とほぼ同じで文書の中に変換ミスが無いかを調べるために使用する。

3. の使い方をする場合、かなり正確な変換と聞き取りやすい音声出力が要求される。今回作成したソフトの品質でこの使い方をするのは不可能だが、将来的にこのような分野においても使用が可能になれば良いと思われる使用例である。これは、 4. についても同様のことが言える。

5. も、 1. や 2. と同様の使い方ができる。これで、資料とプリントアウトしたデータなどを交互に見ながら間違いを発見するという作業の負担を減らすことが可能になる。

6 は、視覚障害者にとっても有効な使用法である。晴眼者と視覚障害者では使い方が若干異なるが、晴眼者の場合は、何か作業をしているときにメールを受信した場合、その作

業を中断しないでメールの内容を確認したいとき、その内容を音声出力することによって作業を行いながらにしてメールの内容を確認するというような使い方ができる。

7. の場合、このソフトはシステム上に常駐していなければならない。しかし、使い方次第では視覚障害者にも使える方法である。

8. は、文書をスキャナで取り込み、 OCR ソフトにかけたときに起こる変換ミスを、確認するときに使う方法である。

全体的に見ると、主に 1. 2. 5. 8. のような文書やデータの読み合わせに使うことが多いと思われる。しかし、ソフトの品質を考えるとこの程度の使い方が最も適当であると思われる。

3　全体の流れ

まず、ソフト全体の流れを Fig.3.1 に示す。

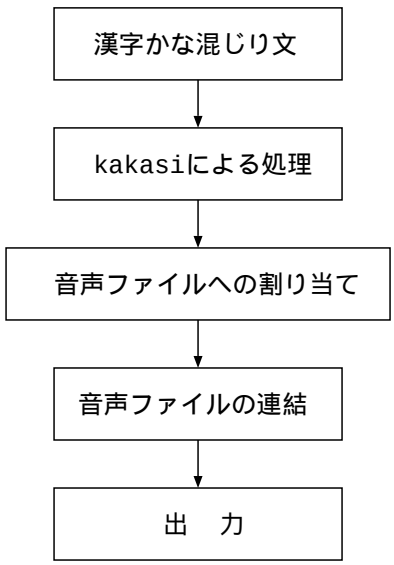


Fig. 3.1　全体の流れ

漢字とかなの混ざった日本語文書を kakasi でひらがなのみの文書に変換を行う。次に、変換された文書をあらかじめ作成した音声データに 1 音 1 音当てはめる。そのままの状態で音声出力を行うと途切れ途切れになってしまうため、音声ファイルの連結を行い音声出力する。この流れを、次節から詳しく説明する。

3.1　kakasi の特徴

ここでは、 kakasi の特徴などを説明する。

kakasi とは、高橋裕信氏が作成した漢字かな読み上げソフトである。そして、今回使用したものは kakasi Version 2.2.5 で、このプログラムはフリーソフトとしてソースが公開されている。

このソフトは漢字、仮名が混ざった文章をひらがな、またはローマ字のみの文章に変換し出力するソフトで、漢字の読めない端末を使ったときや、漢字に不慣れな外国人や子供に文章を紹介したい時に使うことを目的として作られた。

kakasi の特徴としては次のようなものがあげられる。

1. 文章をひらがな、ローマ字のどちらにでも変換できる
2. 熟語の途中に空白や改行が入っていても変換できる
3. 変換後の表示の方法を選べる

3.1.1 kakasi の変換速度

kakasi の速度を測るうえで、日本語形態素解析システム『茶筌』 Version 1.51 と比較した。茶筌とは、奈良先端科学技術大学院大学松本研究室で開発されたフリーソフトで、kakasi と同様に漢字かな混じり文をひらがなに変換する機能があり、主に形態素の解析に使用される。また、 kakasi 、茶筌ともにインターネットの全文検索システムのインデックスの収集として、文集の単語の分割などに用いられている。茶筌は kakasi に比べて、変換した後に出力する内容が多く、単語ごとに文法と活用形を表示する。茶筌と kakasi では、目的が多少異なっているが日本語をひらがなに変換するスピードは kakasi だけではわからないので、茶筌を使い比較を行っていく。

kakasi と茶筌の速度比較は、time というプログラムの実行中に消費した時間を表示する UNIX のコマンドで、実際の測定結果は以下のようになった。

```
コマンド % time kakasi -JH < test4.txt

結果 0.14u 0.22s 0:03.36 10.7%

コマンド % time chasen test4.txt

結果 0.55u 0.90s 0:10.50 13.8%
```

ここで、出た結果は左から順に

- ユーザが使用した時間
- システムが使用した時間
- 実時間
- cpu 占有率

を表している。
cpu 占有率は

$$\frac{\text{ユーザ時間} + \text{システム時間}}{\text{実時間}} * 100(\%)$$

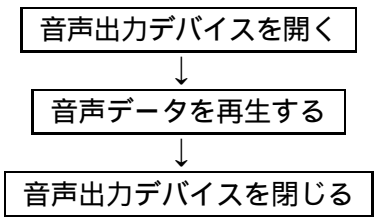
で計算される。速度の比較を行うときは、「ユーザ時間 + システム時間」を見れば良い。
この場合、茶筌は kakasi の約 4 倍の時間がかかっていることになる。これは、どちらも付属の辞書ファイルを参照して変換を行っているが、茶筌は kakasi よりも辞書ファイルのサイズが大きく、文字の変換以外にも文法の検索、表示を行っているためやらなければいけない処理の量が kakasi に比べて多いことが原因と考えられる。そのため、今回のソフトを作るうえでは、余計な処理を行わず変換に時間のかからない kakasi の方が良いと言うことになる。

3.2 音声ファイルの出力

ここで行っているのは、 kakasi によってひらがなに変換された日本語文書を音声データに割り当てる作業である。ここで必要な音声データは以下の通りである。

- 50 音データ
- 濁音などのデータ
- 「きゃ」「きゅ」「きょ」等、小さい文字が入るデータ
- アルファベット
- 数字
- 無音データ

これを 1 文字ずつひらがなの部分に割り当てていき、無音データは、文書の中の句読点やスペースの部分に割り当てる。無音データを割り当てることで、文書を読み上げる際に疑似的に息つぎをさせて、出力された音声をできるだけ聞き取りやすくすることが可能である。
しかし、この方法では日本語文書を音声データに割り当てて出力するだけなので、このデータの状態で音声出力を行うと以下のようになる。



この状態で実行すると 3 つの作業を 1 音 1 音繰り返すことになり、音声出力デバイスを閉じるたびに雑音が入り、聞き取りにくくなる。また、この作業のスピードが出力方法により異なり、期待する出力にならない可能性がある。そこで、Fig.3.1 に書いたように文章ごとに音声ファイルの連結を行い、一つの音声データとして再生する必要がある。

3.2.1 音声出力

UNIX には、音声データを出力する方法が複数ある。例えば、 SUN や富士通のワークステーションの OS である Solaris では、次のような方法が考えられる。

1. 音声出力デバイス `/dev/audio` に直接流す
2. Solaris のコマンドである `audioplay` を使用する
3. NAS のコマンドである `auplay` を使用する

1. は UNIX に常駐しているデバイスに直接音声データを流して出力する方法で、これを使うと前に書いたように実行、再生、終了を繰り返すので雑音が入って聞き取りにくい。2. は Solaris 独自のコマンドで汎用性に問題がある。 3. は NAS(Network Audio System) という audio server のコマンドで、この NAS が動いている場合 `/dev/audio` に直接流すことができない。この 2 つのコマンドで 1 音 1 音つなげずに再生を行うと実行、再生、終了の繰り返しは無いものの音と音のつながりが不自然な感じがある。

実際に、音声出力を行う場合は `/dev/audio` に直接流す方法と、`audioplay` 等の音声出力外部コマンドに流す方法の両方で使えるようにする必要がある。そして、どの方法を使っても音声がうまく出力されるように、音声データをスムーズにつなげてから出力する必要がある。

3.2.2 音声データの連結

前に説明したように、 1 音 1 音割り当てて再生を行うと聞き取りにくい音声になってしまう。そこで、音声データの連結を行うわけだが、この場合どのような単位で連結すれば良いかが問題になってくる。

1 つの方法として考えられるのは、句点で区切る方法である。これだと、 1 つの文章が 1 データとして連結されて非常に区切り良く音声出力される。または、1 文が長くなることを想定して句点、読点の両方で区切る方法もある。しかしこの場合、短い文章のときには 1 つの音声データの長さが極端に短くなってしまうこともありえる。

他の方法として、変換する文章を全て 1 つのデータにしようとする方法があるが、そうするとデータの連結処理に時間がかかってしまうことが予想される。また、改行を判別してデータを連結するという方法もある。この方法は、表計算データのような場合には適していると思われるが、文書データを出力した場合、変な場所で間が空いてしまう恐れがある。以上のことを考えると、最初に書いた句点で区切る方法が最も良いのではないかと思われる。

4　kakasi の問題点

4.1　kakasi のオプション

kakasi には、標準で用意されているオプションがある。例えば、辞書ファイルを参照したときに単語の読みが複数ある場合に読み方を全て表示するオプション、変換前の漢字の脇にその読みを差し込むオプション。この 2 つは、 特徴の 3. に該当する。しかし、今回は kakasi で変換を行った後に、別の処理が残っているので、表示の方法はデフォルトの「ひらがなのみの文章で出力する」方法を使う。そして、今回のソフトで使用するオプションは以下の通りである。

-JH: 漢字をひらがなに変換する

-KH: 全角カタカナをひらがなに変換する

-kH: 半角カタカナをひらがなに変換する

-Ea: 全角の記号を半角の記号に変換する

-c: 熟語中に含まれる改行や空白を除いて読む

これらのオプションを使用することで、漢字、カタカナをひらがなに変換した状態の文章として出力される。

4.2　kakasi による処理の流れ

日本語文書をひらがなの文書に直す場合の kakasi の流れを書くと Fig.4.1 のようになる。

入力された漢字かな混じり文を句読点ごとに分割し、辞書ファイルを参照しながら単語ごとに分割を行い、漢字をひらがなに直す。そこで変換されなかった漢字は更に細かく分割され、すべての漢字がひらがなに変換されるまでその処理を繰り返し、最後にひらがなのみの文章として出力する。

4.3　機能的な問題

kakasi にはさまざまなオプションがあり、その中で今回のソフトに使用するオプションは前に説明した。そして、そのオプションを使用して文書の変換を行うと、次のようになる。

新潟工科大学は、柏崎市にあります。
↓
にいがたこうかだいがくは, かしわざきしにあります.

そして、これは次のように音声データが割り当てられる。

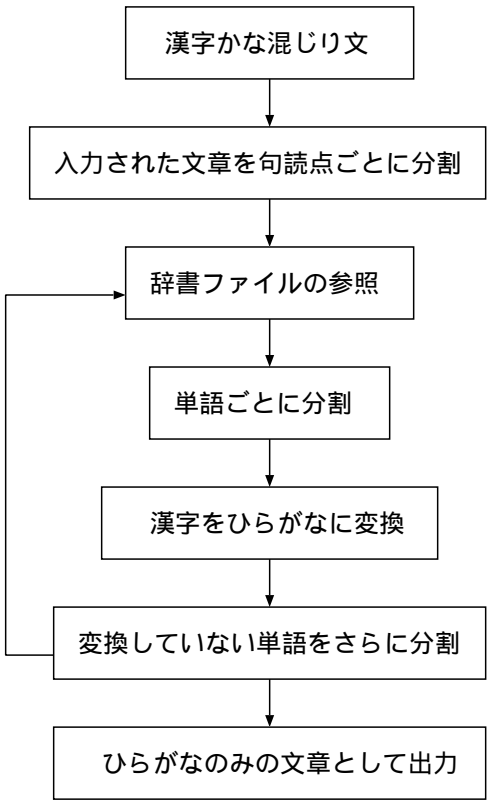


Fig. 4.1 kakasi による処理の流れ

```
{ni,i,ga,ta,ko,u,ka,da,i,ga,ku,ha,null,
  ka,si,wa,za,ki,si,ni,a,ri,ma,su,null}.au
```

この例の場合にはあまり問題は無いように見えるが、変換する文書が長くなると無音データ (`null.au`) が非常に少なくなり、長く連続した音声になって聞き取りにくい。そこで、

-s: 変換した文字の前後に空白を空ける

というオプションがあるが、これを追加して変換を行うと次のようになる。

にいがた こうかだいがく は , かしわざき し にあります .

そして、ここに音声データを割り当てると、

```
{ni,i,ga,ta,null,ko,u,ka,da,i,ga,ku,null,ha,null,null,null,
    ka,si,wa,za,ki,null,si,null,ni,a,ri,ma,su,null,null}.au
```

のようになり、今度は非常に多くの無音データが入ってしまう。このように、句読点の間に無音データが入ってくるのは良いが、変換された単語ごとに音声が途切れると、聞き取

りにくくなる。特に「工科大学 は」のように、本来ならば間を空けて話さないような部分にスペースが空いていると非常に聞き取りにくい。そこで、「漢字 + ひらがな」の間にスペースを空けないようにすれば良いわけだが、 kakasi にはそういうオプションは存在しない。

4.4 文法的な問題

もともと kakasi は、日本語文書を変換してディスプレイ上に表示するソフトなので、そのままで使用すると問題が出てくる。それは、実際に記述されている文字と、読むときは発音が変化してしまうひらがなが存在するということである。それは、副助詞の「は」と格助詞の「へ」である。この文字は、実際に発音するときは「わ」、「え」と読まなければならない。

例えば、

「明日は海へ行こう。」

という文章を kakasi で変換すると、

「あしたはうみへいこう。」

のように出力される。そして、これに音声データを割り当てると当然「は」と「へ」はそのまま読まれてしまい変になる。これを正しく読むためには、

「あしたわうみえいこう。」

のような変換を行わなければならない。

この例のように、 kakasi では行えない変換をするためにプログラムの修正が必要になってくる。

4.5 漢字の変換精度

kakasi は、文章の変換を付属の辞書ファイルを参照しながら行っている。当然、そのファイルの中に登録されていない単語はひらがなに変換することができない。また、読み方が多い単語には優先順位がついていて、単語を変換したときに出力される読みはある程度決ってくる。そこで、今回 3 種類のテストを行い、1 つ目の難読文字ではどの程度の単語まで登録されているか、2 つ目の複合語では漢字変換がどのくらい柔軟性を持っているか、3 つ目の長文で総合的な変換能力のチェックを行った。

結果は、正しいひらがなに変換できたら正解として考える。

4.5.1 難読漢字

難読漢字のテストは、参考文献 [11] に載っていた難読漢字の一覧表から、良く知られている漢字を 50 個選び出して行った。

結果 (正解数/問題数)　　43/50

選んだ漢字は以下の通り。( ) の中が正しい読み、不正解の漢字には「 」でどのように変換を行ったかを報告する。

久遠「くどう」(くおん)　乞食 (こじき)　山茶花「やまちゃはな」(さざんか)
土筆 (つくし)　土龍 (もぐら)　欠伸 (あくび)　天晴 (あっぱれ)
五月蝿い (うるさい)　木霊 (こだま)　木魚 (もくぎょ)　白湯「しろゆ」(さゆ)
台詞 (せりふ)　布袋 (ほてい)　早乙女 (さおとめ)　老舗 (しにせ)
西瓜 (すいか)　吠面「ばいめん」(ほえづら)　牡丹 (ぼたん)　河童 (かっぱ)
刹那 (せつな)　岩魚 (いわな)　相槌 (あいづち)　海老 (えび)　南瓜 (かぼちゃ)
剃刀 (かみそり)　為替 (かわせ)　祇園 (ぎおん)　胡麻 (ごま)　重宝 (ちょうほう)
海苔 (のり)　紅葉「こうよう」(もみじ)　殺生 (せっしょう)　梅雨 (つゆ)
疾風 (はやて)　浴衣 (ゆかた)　悪戯 (いたずら)　悪寒 (おかん)　健気 (けなげ)
添削 (てんさく)　梯子 (はしご)　閏年 (うるうどし)
漁火「りょうひ」(いさりび)　境内 (けいだい)　撫子 (なでしこ)　暴露 (ばくろ)
頭布「あたまぬの」(ずきん)　曖昧 (あいまい)　下手物 (げてもの)
天秤 (てんびん)　向日葵 (ひまわり)

難読漢字と言っても、そのほとんどが誰でも知っているような漢字なので、正解率はかなり高いと言える。不正解の漢字を見ると、「紅葉」のように他の読み方があるものは、よく使われる読みが優先されている。しかし、その単語自体に読みが与えられていないと、例えば「山茶花」の変換結果のように 1 文字づつ読みが与えられ、意味の通らない読みが与えられる。

このような難読漢字変換の正解率を上げるためには、辞書ファイルに登録されている単語の数を増やせば良い。しかし、よく使われる難読漢字が登録されており、今回のテストで使用しなかったものでも、登録されている漢字は多いと思われる。そこに今回変換を失敗した漢字を登録しようとしても、その漢字の使用頻度などを考えると、それほど重要なこととは思えない。そこで、今回は頻繁に使われる常用漢字さえ正確に変換できれば日常で使用する分には問題ないと考えた。

4.5.2　複合語

ここでは、複合語の変換テストを行った。複合語とは、

「株式 (かぶしき)」+「会社 (かいしゃ)」=「株式会社 (かぶしきがいしゃ)」

のように、二つの単語を組み合わせたもので、それによって後ろの語の 1 文字目または前の語の最後の文字の読みが変化する語のことを言う。この複合語を 15 個選び出して変換テストを行った結果、以下のような数字になった。

結果 (正解数/問題数)　　09/15

次に変換テストに使用した語の一覧を表示する。左側の漢字が使用した語、ひらがなは実際に kakasi で変換したときに出力された結果である。

(正解した複合語)
「株式」+「会社」=「株式会社」「かぶしき」+「かいしゃ」=「かぶしきがいしゃ」
「紙」+「袋」=「紙袋」　「かみ」+「ふくろ」=「かみぶくろ」
「骨」+「組」=「骨組」　「ほね」+「くみ」=「ほねぐみ」
「土曜」+「日」=「土曜日」　「どよう」+「にち」=「どようび」
「土産」+「話」=「土産話」　「みやげ」+「はなし」=「みやげばなし」
「白」+「百合」=「白百合」　「しろ」+「ゆり」=「しらゆり」
「壁」+「紙」=「壁紙」　「かべ」+「かみ」=「かべがみ」
「底」+「力」=「底力」　「そこ」+「ちから」=「そこぢから」
「眉」+「毛」=「眉毛」　「まゆ」+「け」=「まゆげ」
(変換ミスした複合語)
「残り」+「火」=「残り火」　「のこり」+「ひ」=「のこりひ」
「彼岸」+「花」=「彼岸花」　「ひがん」+「はな」=「ひがんはな」
「機械」+「仕掛け」=「機械仕掛け」　「きかい」+「しかけ」=「きかいしかけ」
「卵」+「酒」=「卵酒」　「たまご」+「さけ」=「たまござけ」
「紙」+「鉄砲」=「紙鉄砲」　「かみ」+「てっぽう」=「かみてっぽう」
「大和」+「魂」=「大和魂」　「やまと」+「たましい」=「やまとたましい」

数字を見る限り、正解率は 60% 程度であまり良いとはいえない。また、一覧を見ればわかるが、大きな変換ミスは特に見当たらず主な間違いは単語と単語が結合したときに読みが変化する部分が、変化しなかった場合である。全ての変換ミスがそのパターンで、漢字自体は難しいものではなかったので、読みを間違えて変換したものはなかった。

この場合の解決法も難読漢字と同様に登録単語を増やすしかない。しかし、難読漢字よりも使用頻度が高いとはいえ、辞書ファイルに登録するのはどうかと思われる。確かに、辞書ファイルには初期の段階で数万個の単語が登録されているので、そこに 100 ~ 200 個の単語が増えたところで kakasi の処理速度が極端に遅くなるようなことは無い。しかし、この程度の変換ミスは晴眼者、視覚障害者ともに聞けば何のことを言っているのか理解できる範囲であるので、この問題はそれほど重要であるとは思えない。よって、このようなことを考えると、今回は辞書ファイルには手を加えずに kakasi のシステム上で起こる問題に焦点を合わせて変更を行っていくのが良いと思われる。

4.5.3　長文読解

また、これとは別に参考文献 [5] より引用した漢字・カタカナを含む約 500 文字程度の長文を変換させてみたところ、それほど大きなミスも無く良い結果を出すことができたが、若干気にある部分もあったので、次にその文章と変換後の文章を掲載する。

―――――――――― 以下、漢字かな混じり文 ――――――――――

西洋の言葉は、支那の言葉と同じように動詞が先に来て、次に目的格が来る。

またテンスの規則があって、時間的に細かい区別をつけることが出来、前の動作と後の動作とがはっきりと見分けられる。

また、関係代名詞と云う重宝な品詞があって、混雑を起こすことなしに、一つのセンテンスに他のセンテンスを幾らでも繋げて行くことが出来る。

その他、単数複数、性の差別等、いろいろな文法上の規定がある。

そう云う構造なればこそ、多くの語彙を積み 重ねても意味が通じるのでありますが、全然構造を異にする国語の文章に彼等のおしゃべりな云い方を取り入れることは、酒を盛る器に飯を盛るようなものであります。

然るに現代の人々は深くこの事実に留意しないで、とにかく言葉を濫費する癖があります。

彼等の書く文章はいずれかと云うと、古典文よりは翻訳文の方に近い。

小説家、評論家、新聞記者等、文筆を業とする人の文章ほどなおそう云う傾きがある。

西洋人は、上に挙げた英文 (略) を見ても分かるように「総べて」 ”all” とか「最も」 ”most” とか云う言葉を惜しげもなく並べ立てますが、現代の日本人もいくつかその真似をして、その必要のないところに最上級の形容詞を使う。

かくて我々は、我々の祖先が誇りとしていた奥床しさや慎しみ深さを、日に日に失いつつあるのであります。

———————————— ここまで ————————————

これを、ひらがなのみの文章に変換すると次のようになる。

———————————— 変換後の文章 ————————————

せいようのことばは、しなのことばとおなじようにどうしがさきにきて、つぎにもくてきかくがくる。

またてんすのきそくがあって、じかんてきにこまかいくべつをつけることができ、まえのどうさとのちのどうさとがはっきりとみわけられる。

また、かんけいだいめいしというちょうほうなひんしがあって、こんざつをおこすことなしに、ひとつのせんてんすにほかのせんてんすをいくらでもつなげていくことができる。

そのほか、たんすうふくすう、せいのさべつなど、いろいろなぶんぽうじょうのきていがある。

そういうこうぞうなればこそ、おおくのごいをつみかさねてもいみがつうじるのでありますが、ぜんぜんこうぞうをことにするこくごのぶんしょうにかれらのおしゃべりないい ほう をとりいれることは、さけをもるうつわにめしをもるようなものであります。

しかるにげんだいのひとびとはふかくこのじじつにりゅういしないで、とにかくことばをらんぴするくせがあります。

かれらのかくぶんしょうはいずれかというと、こてんぶんよりはほんやくぶんのほうにちかい。しょうせつか、ひょうろんか、しんぶんきしゃなど、ぶんぴつをぎょうとする にん のぶんしょうほどなおそういうかたむきがある。

せいようじんは、うえにあげたえいぶん (りゃく) をみてもわかるように「すべて」”all” とか「もっとも」 ”most” とかいうことばをおしげもなくならべたてますが、げんだいのにほんじんもいくつかそのまねをして、そのひつようのないところにさいじょうきゅうのけいようしをつかう。

かくてわれわれは、われわれのそせんがほこりとしていた おくとこしさ や しんしみふかさ を、にちににちに うしないつつあるのであります。

———————————— ここまで ————————————

変換後の文章の中で下線部が変換を失敗した部分である。変換を失敗した部分の多くは読み方が多い文字で、辞書ファイルの優先順位が高いもので表示されたためであると考えられる。また、「奥床しさ」のように漢字で表現することの少ない単語、「慎しみ」のように送りがなのつけ方が複数あるような漢字は変換ミスをしてしまうことが多いようだ。

変換した内容を見ると、約 500 文字の長文の中で変換ミスをしたところは 5 箇所だけである。このようなミスを無くすためには、読みの優先順位の変更を行わなければならない。しかし、それを行うと今度は今まで正しかった文字が正しく変換されなくなる恐れがある。このような問題はいくら変更を施しても、変換ミスは減らないと考え、単語の登録や変更は行わないこととする。

5　変更点と結果

kakasi で文書の変換を行ったときに起こる問題を前で説明したが、今回 kakasi で変更を行った部分は、-s オプションをつけたときの空白の空け方の改良と、出力して音声に割り当てるときに問題となる副助詞の「は」、格助詞の「へ」をそれぞれ「わ」、「え」に置き換えて変換し、出力するためのオプションの作成を行い、 kakasi の中で使えるようにした。その流れを示したものが Fig.5.1 である。ここでは、変更を行って出力がどのように変わったか、そして、変更を行っているうちに新たにでてきた問題点をあげながら詳しく説明していく。

5.1　分割モード

kakasi は、漢字、ひらがななどの区別を、大半の日本語対応 UNIX ワークステーションの内部コードとして使われている EUC (Extended Unix Code) コードで行っている。通常の -s オプションは変換する単語と単語の間や、分割の単位が小さくなったときは EUC コードを判断して前の語と後の語が異なっている場合にスペースを空けるため、熟語の前

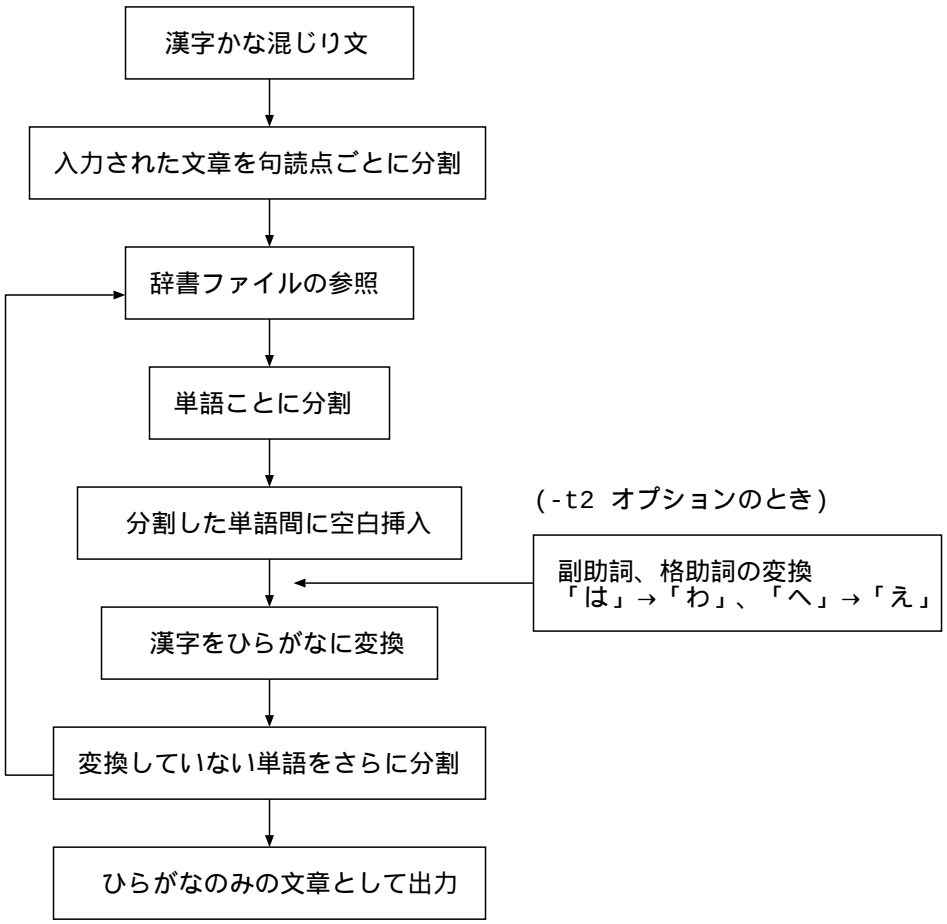


Fig. 5.1 変更後の kakasi の処理の流れ

後や漢字からひらがなに変わる部分の間などにスペースが空いている。それでは文章中に空いているスペースが多くなってしまうため、そういう部分を減らす必要がある。変換する単語ごとにスペースが空くため、熟語を変換したあとにも当然スペースは挿入される。それが、例えば「熟語 + 助詞」で音声に変換した場合、非常に聞き取りにくい。そこで今回は、「漢字 + ひらがな」のスペースを取り除き、できるだけ音声に間を空けないようにした。

ソース中では、文章の中で単語間にスペースを空けるかを判断するサブルーチンは、 -s オプションを指定したときに、外部変数である bunkatu_mode を 1 にすると実行され、初期状態の 0 である場合は実行されない。そこで、 bunkatu_mode に入れる番号を増やし、その番号で判断の方法を変えられるようにする。新たに作成するオプションを -t1 、 -t2 とし、それぞれ bunkatu_mode を 2 、3 としてプログラムを変更する。ここで、-t1 オプションとは漢字とひらがなの間には空白を空けないようにし、 -t2 オプションは、その機能とあわせて、漢字のあとにくる副助詞の「は」や格助詞の「へ」を「わ」、「え」に変更

するオプションである。この-t2 オプションについては次に説明する。

5.2 文字変換処理

文字変換処理は、前で説明した -t2 オプションにあたる部分で、文字を音声に変換したときに起こる読み方の違いを解消するためのサブルーチンである。

副助詞、格助詞を見分ける方法の一つとして、漢字の後ろにくる「は」、「へ」はほぼ確実に「わ」、「え」と読まれるため、それを判断して変換する方法がある。それ以外の方法を使う場合、形態素の解析を行って副助詞、格助詞の判断をしなければならないため、その場合は前に紹介した形態素解析システム『茶筌』を使用して行わなければならない。よって、 kakasi を使った、前者のやり方で処理を行う。

しかし、この方法で処理を行う場合、例外がでてくる。例えば、

- 官製はがき
- 水はけが良い
- 仲間はずれ
- 中途はんぱ

のような場合、「は」を「わ」に変換すると逆に変な読み方になってしまう。また、それとは逆に「漢字 + ひらがな」で無くても文字の変換を行わないと読み方が変になってしまう場合がある。

- ~ では
- ~ には
- どこへ

このような場合にも「は」や「へ」の処理を行わなければならない。これらの処理に関しては、 kakasi とは別の部分で新たにプログラムを作り、処理を行わなければならなくなる。例外処理のプログラムに関しては、時間の都合で作成が間に合わなかったため、説明だけにとどめた。

さて、実際に文字変換処理をするためには、分割モードを行うときに文字の種類を判断するのに利用している EUC コードを読みとり漢字の後の文字が「は (ECU code : A4CFh)」のときは「わ (EUC code :A4EFh)」、「へ (EUC code : A4D8h)」のときは「え (EUC code : A4A7h)」に置き換えるようにプログラムを書き加えた。この処理は前の空白を空ける判断の処理と一緒に行うと EUC コードの判断も続けて行うことができ、短いサブルーチンで処理を行うことが可能になる。

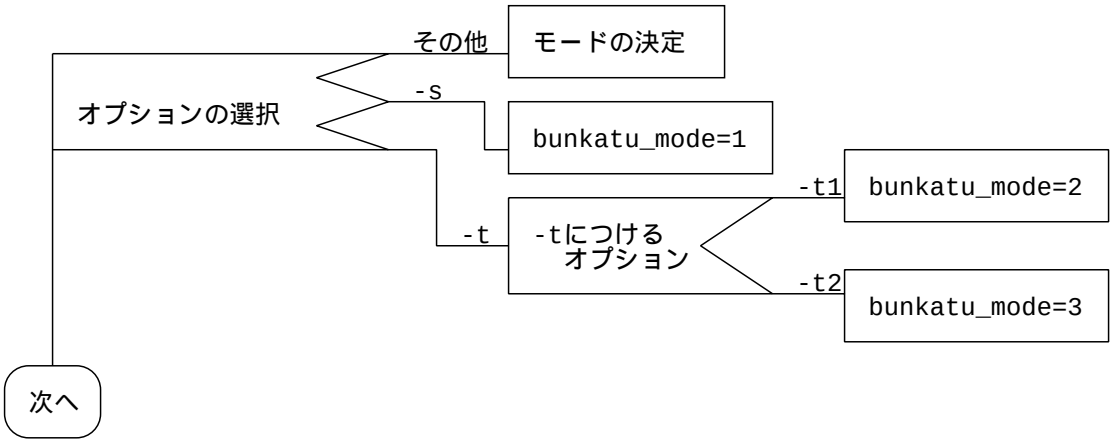


Fig. 5.2 追加オプションの PAD

5.3 変更後のプログラムの流れ

Fig.5.2 に示すように標準オプション -s が前後にスペースを空けるのに対して「漢字 + ひらがな」の部分にはスペースを空けないオプションを -t とした。ここで、 -t を -t1 、-t2 と区別することで文字変換処理を行うか判断させている。そして、その処理を行うために挿入したプログラムは次のようになる。

```
case 't':
 switch((*argv)[2]) {
   case '1':
     bunkatu_mode = 2;
     break;
   case '2':
     bunkatu_mode = 3;
     break;
 }
 break;
```

このプログラムを挿入した部分は switch ～ case 文で入力されたオプションを判断する部分であり、指定されたオプションによって別のプログラムに移動または、モードの決定を行う。 bunkatu_mode は、標準で入っている-s オプションで 1 と指定されているため -t1 のときは 2 、 -t2 のときは 3 として次の命令に移る。

Fig.5.3 は、 Fig.5.2 の続きにあたる部分で、後から追加した -t1 、-t2 オプションまたは -s オプションが指定された場合に実行されるプログラムである。bunkatu_mode は初期状態では 0 か 1 しか無いため -t オプションを指定したときに使えるように変更を行った。そして、この処理を行なうためのプログラムは次のようになる。

```
 if (bunkatu_mode == 1) {
  if (ptype != pctype) {
    put_separator();
    pctype = ptype;
  }
}
else if (bunkatu_mode == 2) {
  if (ptype != pctype && (ptype !=6 || pctype !=7)) {
    put_separator();
    pctype = ptype;
  }
}
else if (bunkatu_mode == 3) {
  if (ptype != pctype && (ptype !=6 || pctype !=7)) {
    put_separator();
  }
  if(ptype == 6 && pctype == 7){
    if (c[0].c2 == 0xcf) {
      c[0].c2 = 0xef;
    }
    else if (c[0].c2 == 0xd8) {
      c[0].c2 = 0xa7;
    }
  }
  pctype = ptype;
}
```

bunkatu_mode=1 は -s オプション、bunkatu_mode=2 は-t1 、 bunkatu_mode=3 は -t2 に対応する。 pctype 、ptype はそれぞれ文字の判断に使い、 1 バイト目を見て、ひらがななら ば 6 、漢字ならば 7 を代入する。 pctype は直前の語、ptype は現在の語を表している。サブルーチンの put_separator()) でスペースを空ける処理を行う。

-s の場合は ptype と pctype が異なる場合に put_separator()) でスペースを空ける。

-t1 の場合、スペースを空ける条件を -s の条件に加えて ptype=6 (ひらがな) 以外または pctype=7 (漢字) 以外のときにのみスペースを空ける処理を行う。

-t2 の場合は、 -t1 の後に if 文を設け、 pctype (前の文字) が漢字で、 ptype (後の文字) がひらがなのときに、 2 バイト目の文字の EUC コードを読みとり「は」のときは「わ」、「へ」のときは「え」にそれぞれ変換する。各オプションごとの処理が終了したら、ptype の値を pctype に代入して次の文字の判断を行う。

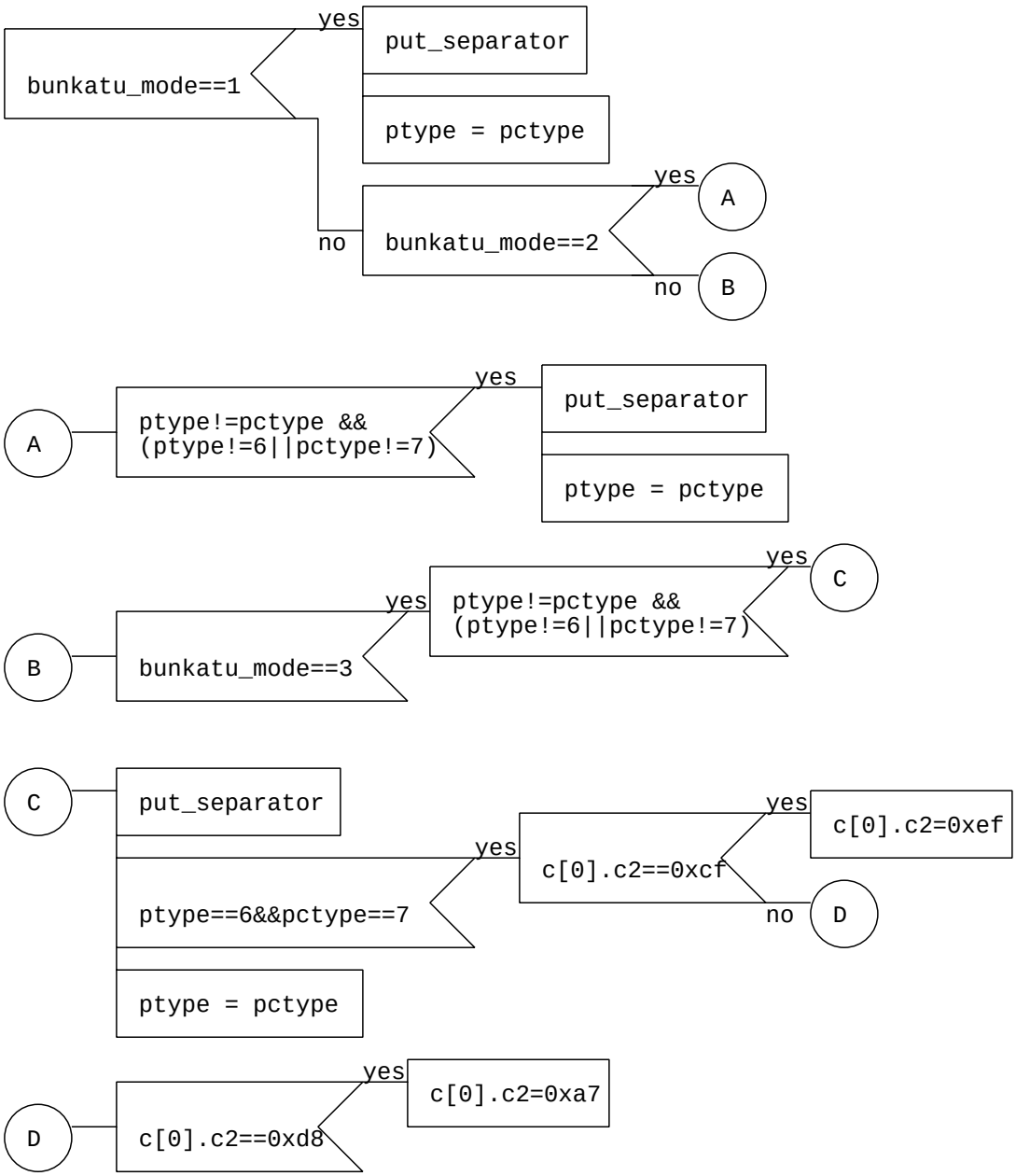


Fig. 5.3 変更部分の PAD

5.4 実行結果

新しいオプションを使用して、長文の変換テストで使用した文章の最初の 1 文を変換する。

西洋の言葉は、支那の言葉と同じように動詞が先に来て、次に目的格が来る。

まず、「漢字 + ひらがな」にスペースを空けないようにするオプション -t1 を使用する。次に表示するのが -t1 オプションを使って変換した結果と、音声データを割り当てたときの出力結果である。

```
せいようの ことばは , しなの ことばと おなじように どうしが さきに きて, つぎに もくてきかくが くる .
```

「西洋の言葉は」の部分を見ると「漢字 + ひらがな」になっているためスペースが空いていない。そのため、音声データの割り当て後も無音データが入っていないため、この部分は間を空けずに連続して音声出力がされる。

次に -t1 オプションの機能に加えて副助詞、格助詞の変換を行う -t2 オプションを使用した場合、結果は以下のようになる。

```
せいようの ことばわ , しなの ことばと おなじように どうしが さきに きて, つぎに もくてきかくが くる .
```

この結果を見てわかるように、「西洋の言葉は」の部分の「は」が漢字の後ろにある文字なので「わ」に変換されている。実際に読み上げソフトのプログラムを実行するときは、この -t2 オプションを使うことになる。

-s オプションを使った場合の出力結果は次のようになっている。

```
せいよう の ことば は , しな の ことば と おなじ ように どうし が さきにきて , つぎに もくてきかく が くる .
```

ここで、ソフトを作成するうえで実際に使用する -t2 オプションで出力した文章に、音声データを割り当てた結果を以下に示す。

```
{se,i,yo,u,no,null,ko,to,ba,wa,null,null,null,si,na,no,null,
  ko,to,ba,to,null,o,na,zi,yo,u,ni,null,do,u,si,ga,null,sa,
  ki,ni,null,ki,te,null,null,null,tsu,gi,ni,null,mo,ku,te,ki,
  ka,ku,ga,null,ku,ru,null,null}.au
```

上の -t1 、 -t2 オプションを使ったときの出力結果と -s オプションを使ったときの結果を比較してみると、明らかにスペースが多くなっており、音声データを割り当てたときの無音データの数が多くなることが予想できる。追加したオプションを使い、副助詞、格助詞の変換も行った状態で音声出力をすると、kakasi に標準で入っていた -s オプションを使用するよりも出力の間が減り、聞き取りやすくなった。 -t1 オプションは、このソフトの中では使わないが kakasi を単体で使用するときに -s に変わるオプションとして使用が可能である。

6　まとめ

今回、主に kakasi の処理の流れの解析、変換精度と処理速度のテスト、出力方法の調査と決定、問題点の抽出と、それに伴い修正を必要とする箇所の特定を行った。そして、kakasi の修正を行い、変換した単語の前後にスペースを空ける処理の改良、副助詞、格助詞の変換処理を行って、それらを新しいオプションとして追加した。そして、そのオプションを使って処理を行い、出力結果と問題点の解決状況を考察した。また、音声データの出力コマンドと音声ファイルの出力方法に関する調査を行った。しかし、音声データを扱う部分に関してはあまり力を入れることができず、最後までできなかったのは反省すべき点である。

今回の修正プログラムを kakasi に書き加えて実行した結果を見ると、まだ多くの問題が残っている。もう少し手を加えれば「漢字 + ひらがな」だけでなく「漢字 + カタカナ」にもスペースを空けないような処理が可能であると考えられる。また、副助詞、格助詞を判断するための他の条件を調べることができれば、変換の精度はさらに良くなる可能性がある。副助詞、格助詞の例外処理に関しても、例外になる単語の調査を行ったが、他にも例外が存在する可能性はある。この点に関しては、まだ調査の必要がある。そして、kakasi の後の処理を行う別のプログラムの作成がほとんどできなかったが、どのような形で処理を行えば良いかを考えていく必要がある。

今後の課題としては、分割モードのさらなる改良、副助詞、格助詞の変換精度の向上、そして今回行うことができなかったが例外処理のサブルーチンの作成、そして、音声データ自動作成プログラムの作成などがあげられる。

参考文献

[1] 高橋 裕信 : kakasi Version 2.2.5 付属ドキュメントファイル

[2] 山崎信英 北川博雄 : "特集 音声合成" , 月刊 C マガジン 1994 年 3 月号 , pp24–58 , ソフトバンク社

[3] 松本裕治 北内啓 山下達雄 平野善隆 今一修 今村友明 : 日本語形態素解析システム『茶筌』 version 1.5 使用説明書 , 奈良先端科学技術大学院大学 , 松本研究室 (1997)

[4] 荒井美千子 : "連載 プログラマー入門 11 速度の測定" , UNIX MAGAZINE 1995 年 12 月号 , pp74–84 , アスキー

[5] 国語教育研究会 編 : 2000 年度版 論作文に役立つ国語常識 , 有紀書房 (1998)

[6] よしだともこ : "よしだともこのルート訪問記" , UNIX USER 1999 年 11 月号, pp99–104 , ソフトバンク社

[7] 株式会社リコーホームページ URL `http://www.ricoh.co.jp/index.html`

[8] The UNIX Super Text 上 , 技術評論社 (1992)

[9] The UNIX Super Text 下 , 技術評論社 (1992)

[10] Ken Lunde 著 , 春遍雀來 鈴木武生 訳 : 日本語情報処理 , ソフトバンク社 (1995)

[11] 永岡書店編集部 編 : 実用ことわざ小事典 , 永岡書店 (1985)

[12] 小山裕司 斎藤靖 佐々木浩 中込知之 : UNIX 入門 , 株式会社トッパン (1996) pp24–58 , ソフトバンク社