ツリーバンク検索への「UNIX 的」アプローチ

窪田 悠介 (筑波大学)

Treebank research and the 'UNIX philosophy'

Yusuke Kubota (University of Tsukuba)

要旨

国語研 NPCMJ コーパスは、(ゼロ代名詞や関係節空所などを含む) きめ細かな統語構造を付与したツリーバンクとして日本語初のものであり、特に統語論や意味論など、今までコーパス利用があまりなされてこなかった分野でのコーパス活用を活性化させることが期待できる。一方で、木構造を検索し、そこから必要な情報を取り出す作業の (一見したところの) 複雑さのため、言語研究への活用は未だ模索段階を出ていない。本論文では、UNIX 系 OS での基本スキルである単純なコマンドを数珠つなぎにしてデータを加工する手法と、ツリー検索・加工に特化したコマンドラインツールを組み合わせて使うことによって、NPCMJ を用いて実際の言語研究に役立つ情報抽出が可能になることを示す。「(ガ/ノ交替の) ノ格でマークされた主語と共起する述語の頻度表を作る」というタスクを例に、コーパスからの情報抽出の具体的な手順を説明する。

1. はじめに: NPCMJ プロジェクトについて

国立国語研究所で開発中の NINJAL Parsed Corpus of Modern Japanese (NPCMJ) は、言語研究への利用を念頭において作られている、日本語を対象としたものとしては初の本格的なツリーバンクである。2016 年度よりプロジェクトを開始し、6 年間のプロジェクト期間中に計 5 万文以上のデータを公開する予定となっている (アラステア・バトラーほか 2016)。

何らかの統語情報を付与した日本語の言語資源で現存するものとしては、京都大学テキストコーパス (京大コーパス) が有名である。京大コーパスが主に自然言語処理研究のリソースとして開発されたものであり、また、句構造でなく係り受け情報をタグ付けしたものであるのに対し、NPCMJ コーパスは、言語学で一般的に用いられている句構造による統語構造情報を付与したコーパスである点に特色がある。[(1)]また、NPCMJ コーパスにおいては、統語構造のみならず、以下のような文法的に重要と考えられる情報も細かくタグ付けされている (詳しくは吉本 (2016)、Butler et al. (2017) を参照のこと)。

- ゼロ代名詞や関係節のトレースなどといった、理論言語学の空範疇に対応する情報
- 表層格のみならず、主語、目的語、受身文の論理的主語といった文法役割
- 副詞節、埋めこみ節、等位節など、従属節の下位分類

(1) 句構造文法のツリーバンクは、言語学研究への応用の他にも、自然言語処理で広く使われている文法理論であるカテゴリ文法のツリーバンクへの転用が容易であるというメリットもある。この点に関しては、例えば Uematsu et al. (2013) を参照のこと。

現在まで、統語論や意味論などのいわゆる理論言語学研究においては、コーパスを使った研究があまり活発に行われてきているとは言いがたい。[(2)] 周知のように、現在までの (特に生成文法系の) 理論研究は、コーパスでは非文の判定ができないということを理由に、もっぱら内省による手法に頼ってきた。このため、現在のところ、理論研究におけるコーパス利用の方法論というものがそもそも未開拓の分野として放置されている。そして、このようないわば理論研究者の「怠惰」を助長した要因の一つとしてコーパス開発者の側からも、理論研究者と共同で実際の理論研究に役立つような言語資源を構築しようという試みをそれほど積極的に推し進めてこなかったという事情があるように思われる。前川 (2013) などが明快に議論しているように、コーパスは決して内省に基づく理論言語学研究の意義を否定するものではなく、むしろそれに欠けているものを補う研究手法を提供する道具として非常に大きな可能性を秘めている。しかしながら、そのような資源が実際に使いやすい形で整備されていなければ、人 (この場合、理論言語学者) は、問題があると薄々気づきながらも慣れ親しんだ研究手法や固定観念の縛りにとらわれがちである。

NPCMJ プロジェクトは、まだ始まってから比較的日も浅く、コーパス自体の構築やその検索インターフェイスの開発に関して様々な課題を抱えている。また、現在までに公開されているデータ (昨年度 1 万文を公開) は言語資源としての規模も非常に小さい。このように、様々な点で発展途上であるとはいえ、コーパス利用に対して否定的ないし及び腰であった理論言語学研究者を明確なターゲットとしているという点で画期的なプロジェクトである。とはいえ、分野の壁を崩し新たな学際的研究を活性化させるのは容易なことではなく、そのためには様々な工夫が必要となる。本論文では、そのような「工夫」の一つとして、文系研究者にも比較的習得が容易であると考えられる、本格的なプログラミングの一歩手前のスキルだけでどこまでツリーバンクを活用できるかということを考えてみることにする。

2. NPCMJ と言語研究

上述のように、NPCMJ プロジェクトは理論言語学とコーパス研究の間の架け橋となる可能性を持つプロジェクトであるが、比較的少人数でコーパス開発にあたっているため、コーパス利用の促進という点でやや出足が遅れている。準備期間を含めると現時点ですでにプロジェクトの始動から二年以上が経過しているが、NPCMJ コーパスを利用した言語研究は未だ模索段階の域を出ない (井戸ほか 2017)。

井戸ほか (2017) は、統語構造を付与したコーパスという生の言語資源と、実際の言語研究 (後者のサンプルは学術誌『日本語文法』に掲載された論文から選んだ) を突き合わせてみる作業を通して、この両者の間に現時点でどれくらいのへだたりがあるかを把握する予備的調査を行っている。この調査により、構造をタグ付けしたコーパスの言語研究への活用に関していくつか非常に有望な見通しが得られた一方で、コーパス開発と利用の促進における今後の課題もまたはっきりと見えてきた。特に大きな問題として分かったのが、単に構造を付与しただけのデータ (=現行版の NPCMJ) の意外なほどの使いにくさである。NPCMJ の開発の大きな動

(2) たとえば、小川ほか (2016) がこの辺の事情を理論研究者側の立場から整理している。

機は、言うまでもなく、BCCWJ のような現在すでに言語研究で利用されている、形態論・品詞情報のみを付与した平坦な構造のコーパスでは、言語の階層性が反映されていないために、実際の言語研究においては用途が限定的になってしまうという (それ自体は正しい) 認識である。このことを考えると、構造だけ付与してもやはり用途が限定的となってしまうというのはやや皮肉な結果といえるかもしれない。とはいえ、言語という研究対象の持つ複雑さや多面性、そして、言語学者の仕事というのがとりもなおさず、そのような複雑で多面的な対象の本質を解明することであるということを考えれば、これはある意味当たり前の結果でもある。

具体的には、実際の言語研究においては、単に「これこれしかじかの構造」というようなものに着目することはまずなく、「これこれしかじかの構造」に「これこれしかじかの性質を持った単語」は生起するかといった、構造的情報と語彙的情報とが複合的に関連する一般化を問題にすることが圧倒的に多い。この際、問題となる語彙的情報が、(たとえば、「取りたて詞」のような) 閉じたクラスである機能語に関するものであれば、該当する単語のリストを別に用意することでとりあえず用は足りる。[(3)] より問題が大きいのは、名詞、動詞、形容詞、副詞といった、内容語に関する語彙的情報が当該の一般化にとって重要となるケースである。たとえば、何らかの理由で様態の副詞が特定の統語環境に現れる例だけを抽出してきたい、というような場合がこれに相当する。この問題をある程度網羅的に解決するためには、語彙意味論的情報のデータベースをツリーバンクとリンクさせることが必要となる。このような試みはすでに英語などに関して行われているが、[(4)] これは機能語のリストを作るよりさらに大変な作業となり、それなりの精度と規模のものを作ろうとすれば、現在進行中のツリーバンク構築のプロジェク自体と同程度の規模の別個のプロジェクトを立ち上げる必要が生じる。

しかしながら、語彙情報データベースの完成を待っていてはいつまでたっても肝心のコーパスを用いた言語研究を始めることができない。そこで本論文では、現行の NPCMJ を用いてすでにどのようなことができるかを具体的に考えてみる。自然言語の統語構造をタグ付けしたコーパスというのは、顕在的に表示している階層構造だけでなく、その階層構造が反映している潜在的な情報も (隠れた形ではあるが) 保持している複雑なデータである。そして、これらの潜在的な情報のうちの少なくともいくつかは、将来的に語彙意味論的情報のデータベースを作る際にも役に立つものであると考えられる。ただし、このような隠れた情報をツリーバンクから抽出するには、単に指定した文字列や構造にマッチする文や部分木をとってくるだけでは難しいことが多く、取ってきたデータを何らかの形で見やすい形にさらに加工することがしばしば必要となる。以下では、そのような作業のためには UNIX 系の OS で古くから使われている技法である、コマンドラインで単純なコマンドを数珠つなぎにしてデータを加工する手法が特に有効であるということを具体例を通して説明する。

(3) とはいえ、言語学者が興味を持つかもしれない機能語の分類すべてに対応するような語彙リストを作るのは事実上不可能なので、機能語の語彙情報を付加する作業ですら決して単純な問題ではない。

(4) たとえば Propbank プロジェクト (https://propbank.github.io/) を参照。なお、言語学者向けにこの手の自然言語処理の言語資源を解説した概説としては、福原ほか (2016) がよい。

3. いわゆる「UNIX 哲学」

木構造のような再帰的データ構造を対象に複雑な検索を行い、その結果をさらに加工することが必要であるというと、高度なプログラミングの知識なしにはほぼ不可能なことのように思われるかもしれない。確かに、自然言語処理の研究にツリーバンクを用いる際にはそのような知識が必要となるかもしれない。だが、言語研究で必要となるたぐいの検索やデータの加工ならば、プログラミングの一歩手前のスキルを持ち合わせているだけでほとんどの場合十分に対応できる。以下ではこの点を具体的に説明するために、まず本節で「プログラミングの一歩手前のスキル」とはそもそも何なのかを説明し、次節でそれをどのように活用・応用することでツリーバンクを言語研究に活用する道が開けるかを説明する。

UNIX 系の OS の古くからの利用者であれば誰でも知っている基本的スキルとして、単機能のコマンドをどんどん数珠つなぎにしていってデータを加工するという方法がある。以下で例を示すが、このアプローチの基本的な考え方は、単純な機能に特化したコマンドほど部品としての汎用性が高いので、他のコマンドと柔軟に組み合わせることができ、それによって様々な複雑なタスクを簡単に手早く処理できるというものである。このような方法で計算機を使うことを「UNIX 哲学」と呼んだりする(5)。大変有用なスキルであるにも関わらず、ある種の人々にとってはあまりにも当たり前すぎるものであるためか、最近までよい入門書がなかった。このためか、文系の言語研究者の間ではあまり浸透していないように思われる。最近出版された上田 (2015) は、コマンドラインをあまり使ったことがない人向けにこの手のスキルを説明した良書である。

以下ではまず、コマンドを数珠つなぎにしてデータを加工するという手法でどんなことができるかを「テキストから単語の頻度表を作る」という古典的な例に即して説明する。(以下で説明する事柄は、この分野の名著として名高い Kernighan and Pike (1984) で紹介されている例ほぼそのままである。) 単語の頻度表を作るというのは、スクリプト言語などを使ったテキスト処理の入門書などでも初歩的内容の到達点の一つとして設定されていることが多い基本タスクである。淺尾・李 (2013) などで説明されている、ループを使ってハッシュテーブルを更新していくたぐいのより典型的なプログラミング手法を知っている人は、それと以下の方法とを比較してみると面白いだろう。

テキストに現れるそれぞれの単語の頻度を数えるというのは、テキストを単語の集合と見なせば、要はそれを各単語ごとの集合にまとめ直して、そのそれぞれの集合に関して要素の数を数えればよいということである。これは、

1. テキストを一行一単語の形式に変換する
2. 行をアルファベット順に並べ替える
3. 同じ単語の行が連続で何回出現するかを数える

という手順で実現できる。`file.txt` が対象のテキストのファイルだとして、UNIX の計算機

(5) 日本語で「哲学」というと何やら仰々しいので、もともとの英語の ‘Unix philosophy’ (= UNIX 的考え方/流儀) という語のほうが名前と中身が一致しているが、日本語としてすわりのいい適当な訳語が思いつかないので、とりあえず「UNIX 哲学」と呼んでおく。

では、これはコマンドラインから以下のようにタイプすることで実現できる。($はプロンプト文字を表すのでタイプしなくてよい。)

```
$ cat file.txt | tr -sc A-Za-z '\012' | sort | uniq -c | sort -nr
                                 ①          ②      ③        ④
```

ここでは 4 つ (最初のファイルの内容を出力するコマンド cat も含めれば 5 つ) のコマンドを | (「パイプ」と呼ばれる) でつないでいる。パイプでつながれたコマンドは順次「前のコマンドから入力を受け取って次のコマンドに出力を渡す」という形で連繋して動作する。具体的には、①〜③がそれぞれ上の 1〜3 のステップに対応する。①のステップでは、アルファベット以外の文字を単語の区切りとみなし、それらを (連続する場合はまとめて) 一つの改行文字に変換している。また、③の uniq -c は重複行を削除し、重複の回数を行頭に印字するコマンドである。最後に、④の sort -nr で結果を数字の降順に並べ替えている。

たとえば、Project Gutenberg から入手したルイス・キャロルの *Alice's Adventures in Wonderland* のテキストファイルの頻出単語上位 5 件を出力すると以下のようになる。

```
$ cat alice.txt | tr -sc A-Za-z '\012' | sort | uniq -c | sort -nr | head -5
   1686 the
    869 and
    799 to
    672 a
    606 of
```

なお、上の方法は、説明の便宜上単純化しているので、大文字、小文字の表記の揺れを統一していない点やアルファベット以外の文字をすべて単語の一部とは見なさずに一律に除去している点など、実際の単語の頻度カウントとしては不正確なところがある点に注意されたい。sed や grep などのコマンドラインツールを利用すれば、これらの点をさらに作り込んで上のコマンドの完成度を高めることは比較的容易である。この種の作業のコツは、はじめから完璧を目指さず、「結構不完全・場当たり的な処置だけど、目指すところに少しづつ近づいている」というステップを積み重ねて少しづつ完成度を高めていくところにある。

このコマンド数珠つなぎの手法は、本格的なプログラミングと比べると、繰り返し処理やハッシュテーブルのような、プログラムの動作自体に関する制御構文や複雑なデータ構造の管理が不要であるという点に特徴がある。また、一つ一つのステップごとに結果を確認しながら徐々に欲しい形にデータを加工していくことができるので、プログラムの効率性などがそれほど問題とならない状況で、試行錯誤しながら作業を進めたい場合には非常に便利な方法である。次節で具体的に説明するが、ツリーバンクを利用した言語研究というのはほぼ完全に未開拓の分野であるので、このように試行錯誤しながら自分の目的に合うツールを手早く組み上げていく方法が特に適している。

4. パイプラインコマンドとしてのツリー検索ツール

本節では、前節で説明したコマンドラインツールとツリー検索・加工用のコマンドを組み合わせることで、実際にどのようにしてツリーバンクから有用な情報を抽出することができ

るかを説明する。なお、コマンドライン環境が整っていれば、本節で説明する作業はすべてWindows や Mac でも行うことができる。Windows の場合、Cygwin をインストールすれば`sort` や `uniq` などのコマンドラインツールを利用できるようになり、また以下で説明するツリー検索・加工用のコマンドとコマンドラインツールを組み合わせて利用することもできる。

4.1 準備作業

まずツリーバンクを入手する必要がある。NPCMJ は検索インターフェイスの都合などでxml 形式でエンコードしているが、コマンドラインツールでツリーを加工する目的では、プレーンテキスト形式のほうが扱いが便利である。NPCMJ と関連する言語資源であるけやきツリーバンク (Butler et al. 2017) では、プレーンテキスト形式でデータを配布している。そこで、以下ではけやきツリーバンクを用いて作業を行う。

けやきツリーバンクは NPCMJ の母体であり、現時点で NPCMJ 公開分 (2016 年度のもので 1 万文) のデータをサブセットとして含む 4 万文 (65 万語) 程度からなる。このうち 25,000文程度が著作権フリーのデータ、残りが毎日新聞 95 年度版 CD-ROM や BCCWJ などの有償のリソースを用いて復元できるデータとして提供されている。NPCMJ のデータが人間のアノテーターによりすべての文を二重にチェックしてあるのに対し、けやきツリーバンクはアノテーションの精度にばらつきがあるが、現時点ではデータ量が多いという利点があり、実際の言語研究に利用する場合でも、この利点が欠点を上回ることが多い。以下ではけやきツリーバンクの著作権フリーのデータを素材として用いる。また、検索用ツールなどもすべてフリーで入手可能なものを用いる。

けやきツリーバンクでは、以下のようにプレーンテキスト形式でブラケット構造で木構造を表している。アノテーション基準は、ペン通時コーパス (Santorini 2010) の方式を日本語に合わせて調整したものを用いている (Butler et al. 2017)。

```
( (IP-MAT (PP (NP (IP-REL (NP-SBJ *T*)
                          (PP (NP (PP (NP (PRN (NP (N 超大型加速器)))
                                              (-LRB- 「)
                                              (NPR 国際リニアコライダー)
                                              (PRN (-LRB- （)
                                                   (NP (NPR ＩＬＣ))
                                                   (-RRB- ） ))
                                              (-RRB- 」 ))
                                          (P の))
                                      (N 誘致))
                                  (P を))
                          (NP-OB1 *を*)
                          (VB 目指す))
                      (NPR 東北))
                  (P にとって))
          (PU 、)
          (PP (NP (NUMCLP (NUM ２０１３)
                          (CL 年)))
              (P は))
          (NP-SBJ *)
          (NP-PRD (PP (NP (N 節目))
                      (P の))
                  (N 年))
          (AX だっ)
          (AXD た)
          (PU 。))
  (ID 4_newswire_KAHOKU_00089_K201401040A0F70XX00001;JP))
```

コーパス検索ツールと加工ツールとしては、それぞれ tgrep2 と tsurgeon を用いる。それぞれのツールのインストール方法に関しては、論文末尾の URL を参照されたい。

コーパスを入手し検索ツールをインストールしたら、最初に準備作業としてコーパスを一つのファイルにまとめ、tgrep2 用の検索データベースを作る必要がある。コーパスファイルはけやきツリーバンクのアーカイブを解凍してできるフォルダの中の treebank という名前のサブフォルダに入っているので、これをとりあえず作業用フォルダ (ここではホームディレクトリ直下の~/work と仮定する) に一つのファイルとしてまとめる。(tgrep2 では空文字列のノードを許さないので、sed で TOP という文字列を根ノードに足している。)

```
$ cat ~/KeyakiTreebank/treebank/* | sed 's/( /(TOP /g' > ~/work/Keyaki.psd
```

そして、tgrep2 用の検索用インデックスファイルを作成する。

```
$ tgrep2 -p Keyaki.psd Keyaki.tgrep
```

これで検索の準備は完了である。

4.2 tgrep2 と tsurgeon によるデータの検索と加工

準備が整ったので、 以下では「(ガ/ノ交替の) ノ格でマークされた主語と共起する述語の頻度表を作る」というタスクを tgrep2 と tsurgeon を用いて行う手順を示す。言語学の文献ではガ/ノ交替に関して、述語が状態述語である場合にノ格が出やすい、というような観察を行っているものもあるので (金 2009)、述語の頻度リストが作れるとその種の観察の妥当性を調べる際などに便利である。

けやきツリーバンクでは、ノ格の主語は以下の例のように、終端文字列が「*の*」という文

字列であるような NP-SBJ または NP-SBJ2 ノードを伴って現れる。

```
(IP-REL (NP-OB1 *T*)
        (PP (NP (PRO 私))
            (P の))
        (NP-SBJ *の*)
        (VB 知ら)
        (NEG ぬ))
```

したがって、tgrep2 で以下のような検索式を書いて、この空範疇を直接支配する IP ノードを取ってくれば、ノ格主語が現れる節をコーパス全体から取ってくることができる。tgrep2 では < は直接支配を意味する。検索式の書き方の詳細に関してはマニュアルを参照のこと。

```
$ tgrep2 -c Keyaki.tgrep '/IP/ < (NP-SBJ|NP-SBJ2 < *の*)' > Keyaki-no.psd
```

ここでは検索結果を一旦 Keyaki-no.psd というファイルに保存している。

さて、この作業によって得られたツリーを一つ一つ目で見ていって述語のリストを作ってもよいわけだが、データ量が多い場合、このような方法は現実的ではない。そこで、それぞれのツリーを加工して、文末の述語以外の要素を取り除くことにする。

ツリーの加工のためには tsurgeon というツールを用いる。tsurgeon は、基本的に

検索式

操作

の形で書いたコマンド (の連続) をファイルに保存して、それをコマンドラインから呼び出す形で利用する (「検索式」と「操作」の間に一行空行が入ることに注意)。

簡単な例を考えてみよう。まず、上で取ってきた主語がノ格でマークされた節だが、ここでの最終的な目標は述語の部分だけを取り出すことなので、埋めこみ節はすべて除去する必要がある。埋めこみ節というのは、要は、IP の下に IP がある構造なので、これは、以下のように書ける。

```
/^IP/ < /^IP/=x

delete x
```

このコードは、IP-ADV、IP-REL などにマッチするノードの直下に同じく IP-ADV、IP-REL などにマッチするノードがあった場合、後者 (x という変数でこれを同定している) を削除 (delete) する、という意味である。この 3 行だけを書いたファイルを、適当な名前、たとえば test.tsurgeon で保存し、コマンドラインから以下のようにタイプすると、先程のツリーから埋めこみ節だけすべて削除したものが作成される。

```
$ tsurgeon.sh -treeFile Keyaki-no.psd test.tsurgeon
```

最終的には、文末の主述語である VB または ADJ ノードだけを含む形、つまり、
(IP (ADJI 濃い))

のような形にまで持っていきたいので、さらに不要なノードを取り除く必要がある。

一番単純な解決は、力業で消すノードをひたすら列挙していくという方法だが、今回のケースでは、消したいノードは IP、VB、ADJI、ADJN 以外すべてであるので、これらすべてを列挙するというのはあまりエレガントな方法ではない。そこで、「IP、VB、ADJI、ADJN 以外を全部消す」というのをどう実現するか、というのを考えてみる。tgrep2 と tsurgeon では正規表現でノードを指定できるので、正規表現のマッチしない文字列を指定する機能を使って「I または V または A で始まらないノードだったら、それを消す」というふうに指定すれば、だいぶ目指すところに近づけそうである。そこで、test.tsurgeon に以下の 3 行を付け加えてみる。

```
/^[^IVA]/=x < __

delete x
```

この結果を見ると、助動詞などを消しきれていないことが分かる。そこで、以下の指定をさらに加える。

```
/^(AX|VB2|ADV|INTJ)/=x

delete x
```

これでほぼ完成である。

最後に、IP-REL、IP-EMB などの IP ノードの下位分類の情報は、今回のタスクにおいては不要な情報なので、これらをすべて IP に統一する。最終的に、論文末尾のプログラム・リストに示したコード ga-no-conv-pred-extract.tsurgeon が得られる。

4.3 コマンドラインツールとの連繋による集計

上の作業によって、以下のような形のツリーのリストが得られた。

```
(IP (ADJI 遠い))
(IP (VB 知ら))
(IP (VB 生れ))
(IP (ADJI 深い))
...
```

今回の作業の目的は述語の頻度リストを作ることなので、目的達成まであと一息である。このあとは、3 節で説明した単語の頻度リストを作る作業と同じ手順をふめばよい。具体的には、動詞ごとに並べ替えて数を数えればよいので、3 節で単語の頻度リストを作った時と同じように、パイプで sort と uniq を以下のようにつなげばよい。(\は二行にまたがるために入れただけで、実際にコマンドラインで打つときは、\を入れずに全部一行に書いて問題ない。)

```
$ tsurgeon.sh -treeFile Keyaki-no.psd ga-no-conv-pred-extract.tsurgeon \
| sort | uniq -c | sort -nr | head -10
    516
     51 (IP (VB ある))
     34 (IP (ADJI ない))
```

```
20 (IP (ADJI 高い))
13 IP
 8 (IP (VB 言う))
 8 (IP (ADJI いい))
 7 (IP (VB とれ))
 7 (IP (VB い))
 7 (IP (ADJI 多い))
```

最初の行と 5 行目にゴミが混じっているが、上位 3 件の「ある」「ない」「高い」が多数を占め、これらを合わせると 105 件で、全体のデータ 503 件の約 1/5 を占めるなどといったことがわかる。

上では tsurgeon のスクリプトを微調整していくところを意図的に冗長に書いたが、この種の作業のポイントは、小さなステップに分けて試行錯誤を繰り返すことにある。まず最初に単純な方法で大まかなものを作ってみて、そのあとで少しずつステップを足して不具合を取り除いて精度を高めていき、最終的には、単純な加工の連鎖で生データから欲しい加工結果を得るようなコマンドに作り込む。このような方法でコマンドを作ると、見通しのよいコマンドを作ることができ、必要に応じて後で手直ししたり、他の目的に流用したりすることが容易になる。

5. おわりに

本論文では、ツリーバンクを言語研究に利用する際の手法として、木構造の検索や加工のためのコマンドラインツールである tgrep2 と tsurgeon を (Linux のみならず、Mac や Windows 上の Cygwin を含む広義の) UNIX 系の環境に標準で備わっているコマンドラインツールと組み合わせて使う方法を提案した。ツリーバンクの言語資源としての最大の特徴は、言語の (主に) 構造的な側面に関する、顕在的、また潜在的な様々な情報を、一つの単純なデータ構造 (再帰的なツリー構造) で表現している点にある。このような言語資源の持つ可能性を最大限に引き出すには、既存のインターフェイスに頼るだけではどうしても限界があり、本格的な研究のためには研究者が自分の目的に合ったツールを作る能力をある程度身につけることが必要となる。本論文では、文系の研究者にも比較的習得が容易であると考えられる「プログラミングの一歩手前」の知識を説明し、これを身につけるだけで実際の研究に役立つツリーバンク検索が可能になることを論じた。ツリーバンクを用いた言語研究はまだまだ始まったばかりであり、本論文で示したこともあくまで最初の一歩でしかないが、これをきっかけにツリーバンクを利用した様々な言語研究が行われるようになることを期待して筆をおくことにする。

謝　辞

本研究は JSPS 科研費 JP15H03210 の助成を受けたものである。

文　献

アラステア・バトラー・吉本啓・岸本秀樹・プラシャント・パルデシ (2016).「統語・意味解析情報付き日本語コーパスのアノテーション」　言語処理学会第 22 回年次大会 発表論文集, pp. 589–592.

Sumire Uematsu, Takuya Matsuzaki, Hiroki Hanaoka, Yusuke Miyao, and Hideki

Mima (2013). “Integrating Multiple Dependency Corpora for Inducing Wide-coverage Japanese CCG Resources.” *Proceedings of ACL 2013*, pp. 1042–1051.

吉本啓 (2016).「統語・意味解析情報付き日本語学コーパスの構築に向けて: アノテーション方式とコーパスの特色」 日本言語学会第 153 回大会予稿集, pp. 434–439.

Alastair Butler, Stephen Wright Horn, Kei Yoshimoto, Iku Nagasaki, and Ai Kubota (2017). “The Keyaki Treebank Manual.”. 国立国語研究所

小川芳樹・長野明子・菊地朗 (2016).「概観: 言語変化・変異の研究とコーパス」 小川芳樹・長野明子・菊地朗 (編)『コーパスからわかる言語変化・変異と言語理論』 開拓社, 東京 pp. 1–28.

前川喜久雄 (2013).「コーパスの存在意義」 前川喜久雄 (編)『コーパス入門』 朝倉書店, 東京 pp. 1–31.

井戸美里・鈴木彩香・窪田悠介 (2017).「NPCMJ コーパスの言語研究への活用: 展望と課題」 . 東海意味論研究会口頭発表資料、http://npcmj.ninjal.ac.jp/wp-content/uploads/2017/03/20170723_tokaiimiron_1.pdf,http://npcmj.ninjal.ac.jp/wp-content/uploads/2017/03/20170723_tokaiimiron_2.pdf

福原裕一・松林優一郎・乾健太郎 (2016).「自然言語処理における意味・談話情報のコーパスアノテーション」 小川芳樹・長野明子・菊地朗 (編)『コーパスからわかる言語変化・変異と言語理論』 開拓社, 東京 pp. 423–442.

上田隆一 (2015).『シェルプログラミング実用テクニック』 技術評論社, 東京.

Brian W. Kernighan, and Rob Pike (1984). *The UNIX Programming Environment.*: Prentice Hall.

淺尾仁彦・李在鎬 (2013).『言語研究のためのプログラミング入門: Python を活用したテキスト処理』 開拓社, 東京.

Alastair Butler, Kei Yoshimoto, Shota Hiyama, Stephen Wright Horn, Iku Nagasaki, and Ai Kubota (2017). “The Keyaki Treebank Parsed Corpus, Version 1.0.”. http://www.compling.jp/Keyaki/, accessed 2017/07/26

Beatrice Santorini (2010). “Annotation manual for the Penn Historical Corpora and the PCEEC (Release 2).”. Department of Linguistics, University of Pennsylvania

金銀珠 (2009).「現代語の連体修飾節における助詞「の」」 日本語科学, 25, pp. 23–42.

関連 URL

NPCMJ	http://npcmj.ninjal.ac.jp/
けやきツリーバンク	http://www.compling.jp/keyaki/
tgrep2	https://tedlab.mit.edu/~dr/Tgrep2/
tgrep2 (バイナリ版)	https://www.acsu.buffalo.edu/~droland/tgrep2/
Tregex/tsurgeon	https://nlp.stanford.edu/software/tregex.shtml
tsurgeon ドキュメント	http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/trees/tregex/tsurgeon/Tsurgeon.html

プログラム・リスト: ga-no-conv-pred-extract.tsurgeon

```
/^IP/ < /^IP/=x

delete x

/^[^IVA]/=x < __

delete x

/^(AX|VB2|ADV|INTJ)/=x

delete x

/^IP-/=x

relabel x IP
```