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)

はじめに

生成文法とコーパス研究の関係 (生成文法側の立場)

> また、生成文法研究者は、コーパスからは一例も見つからないような事例を、自身が提案する仮説の証拠として使うことも少なくなく、これについての批判を聞くこともある。しかし、これは、生成文法が統語構造を研究していることと、現在のところ統語構造についてのタグを付与されたコーパスが存在しないことから来る必然的な帰結であって、生成文法の問題というより、コーパスの問題といえるであろう。
>
> (小川・長野・菊地 2016, 13)

アウトライン

目的:

- 統語・意味解析情報付き現代日本語コーパス (NPCMJ) の紹介
- (特に文系の) コーパス研究における UNIX ツールの活用例の紹介

進め方:

1. NPCMJ コーパスの概要
2. いわゆる「UNIX 哲学」について
3. UNIX ツールを用いたツリーバンク検索の実例

アウトライン

目的:

- 統語・意味解析情報付き現代日本語コーパス (NPCMJ) の紹介
- (特に文系の) コーパス研究における UNIX ツールの活用例の紹介

進め方:

1. NPCMJ コーパスの概要
2. いわゆる「UNIX 哲学」について
3. UNIX ツールを用いたツリーバンク検索の実例

統語・意味解析情報付き現代日本語コーパス (NPCMJ)

NINJAL Parsed Corpus of Modern Japanese (NPCMJ)

- 国立国語研究所で開発中のツリーバンク
  (= 統語構造についてのタグを付与されたコーパス)
- 年間 1 万文を web 上で公開 (http://npcmj.ninjal.ac.jp/)
  - 検索インターフェイスも提供
  - 「`npcmj ninjal`」で Google 検索するとたどり着ける
- 昨年度末、最初の 1 万文をインターフェイス (Ver 1) とともに公開した。

ツリーバンクと言語研究

ツリーバンクとは

- 統語構造情報を付与したコーパス
- 文の階層的な構造に基づく検索ができるため、統語論や意味論の研究への活用が期待できる

とはいえ

- (通時的な言語研究を除いては) 今のところどちらかというと自然言語処理のリソースと考えられがち (cf. Penn Treebank)

NPCMJ の特徴: 言語学研究を念頭に開発

- ペン通時コーパスにならい、文法関係やゼロ要素など、言語学的に重要な情報を細かく付与している
- 言語学的な知識のあるアノテーター (主に言語学専攻の大学院生) による人手でのチェックを徹底している

ツリーバンクと言語研究

ツリーバンクとは

- 統語構造情報を付与したコーパス
- 文の階層的な構造に基づく検索ができるため、統語論や意味論の研究への活用が期待できる

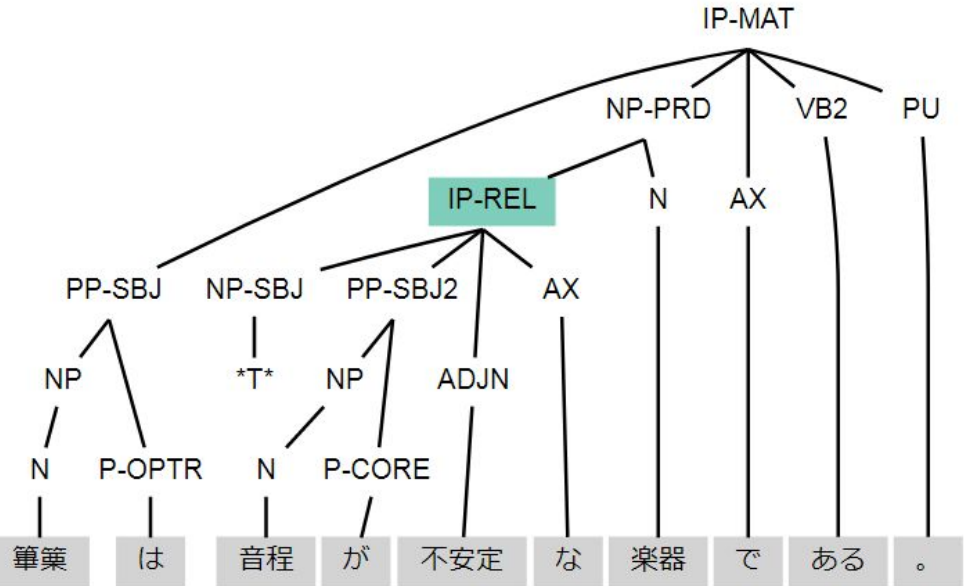
とはいえ

- (通時的な言語研究を除いては) 今のところどちらかというと自然言語処理のリソースと考えられがち (cf. Penn Treebank)

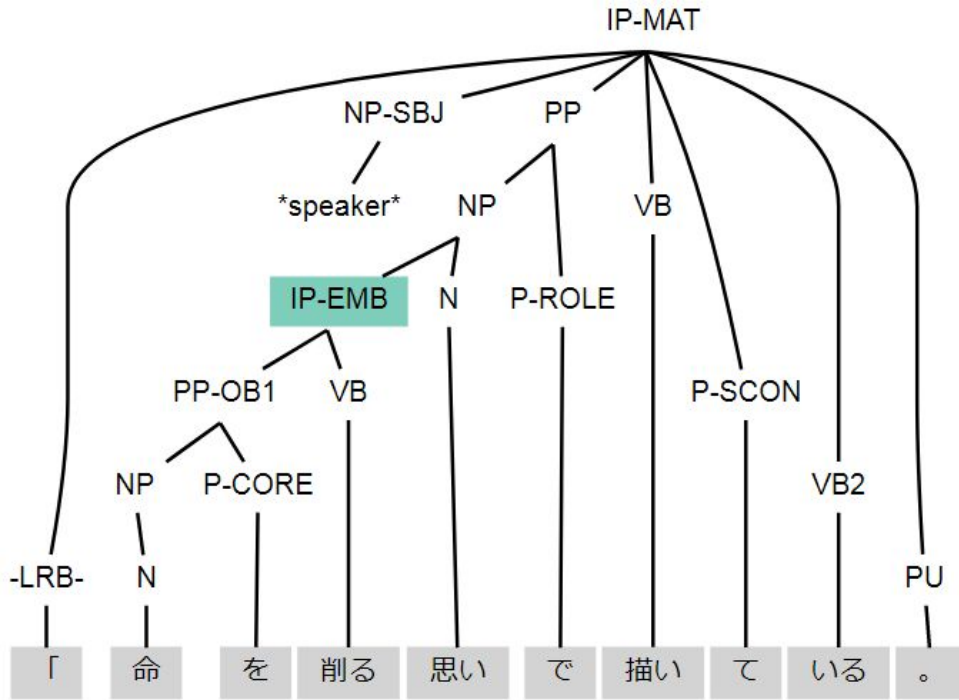
NPCMJ の特徴: 言語学研究を念頭に開発

- ペン通時コーパスにならい、文法関係やゼロ要素など、言語学的に重要な情報を細かく付与している
- 言語学的な知識のあるアノテーター (主に言語学専攻の大学院生) による人手でのチェックを徹底している

ツリーの例



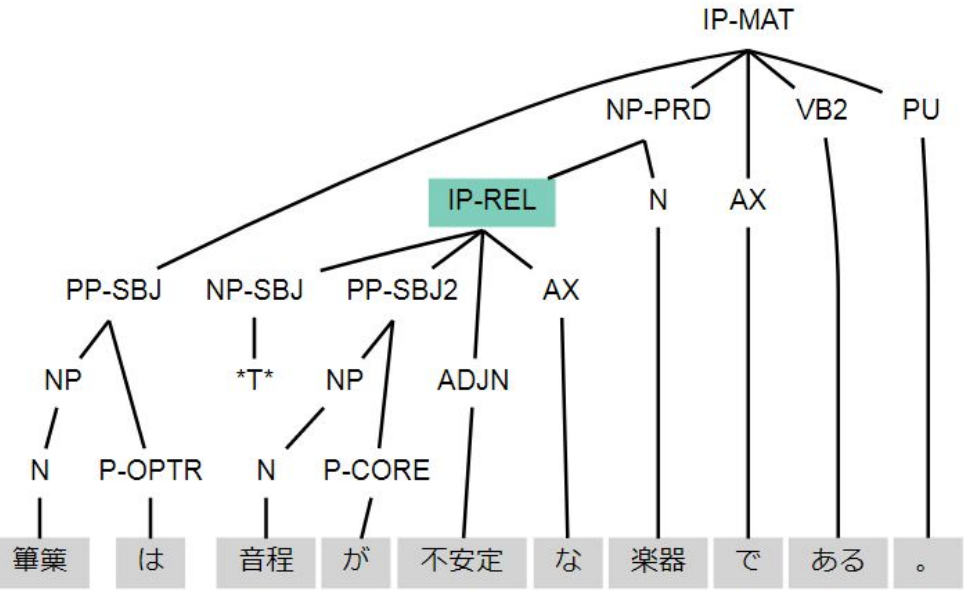
ID: wikipedia_KYOTO_12_CLT_00005_0100



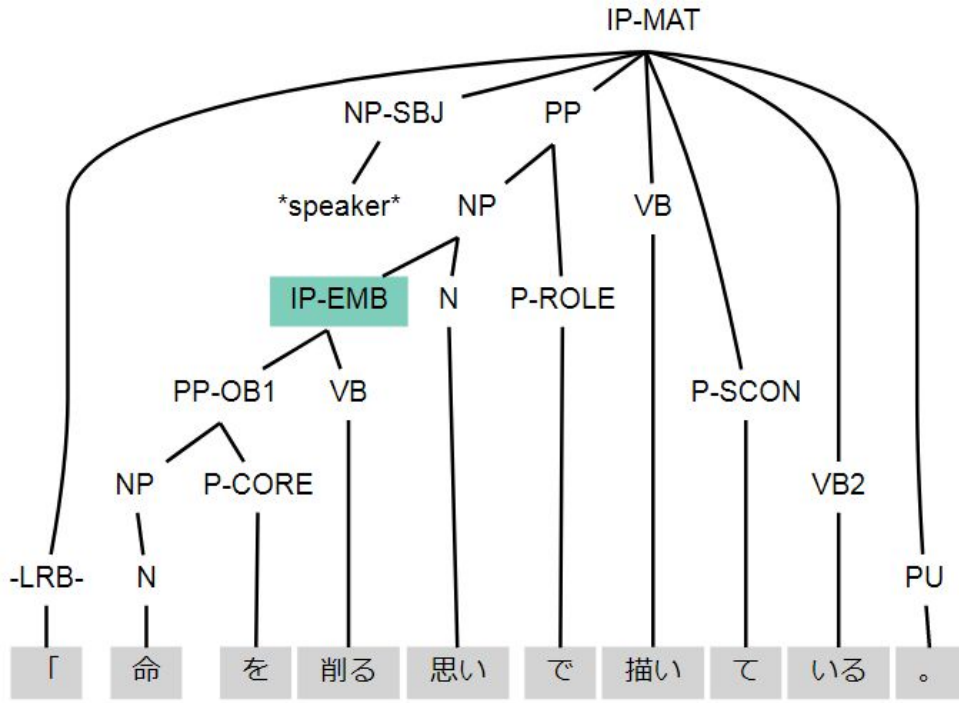
ID: newswire_KAHOKU_00073_K201401010A0F50XX00001_0082

- 関係節の内の関係と外の関係を区別
- トレースやゼロ主語などの情報も入っている

ツリーの例



ID: wikipedia_KYOTO_12_CLT_00005_0100



ID: newswire_KAHOKU_00073_K20140101OA0F50XX00001_0082

- 関係節の内の関係と外の関係を区別
- トレースやゼロ主語などの情報も入っている

NPCMJ コーパスの内訳

NPCMJ 公式版 (昨年度公開計 1 万文)

テキストジャンル	ツリー数
新聞記事 (河北新報)	4323
Wikipedia 記事	2745
新・旧約聖書	1652
教科書 (益岡・田窪)	1378

すべてのデータを
人間のアノテーター
が二重にチェック

けやきツリーバンク (Butler et al. 2017)

NPCMJ 公開分に加えて、

- 青空文庫 約 5200 文
- 毎日新聞 1995 年 約 1600 文 (京大コーパスの一部と同じ)
- BCCWJ のテキストの一部 約 4400 文

などを含んだ約 4 万文。(内 1.4 万文は復元に元テキストが必要。)

- NPCMJ 公式版と比べると精度にばらつきがある
- web インターフェイスは付属しない

NPCMJ コーパスの内訳

NPCMJ 公式版 (昨年度公開計 1 万文)

テキストジャンル	ツリー数
新聞記事 (河北新報)	4323
Wikipedia 記事	2745
新・旧約聖書	1652
教科書 (益岡・田窪)	1378

すべてのデータを
人間のアノテーター
が二重にチェック

けやきツリーバンク (Butler et al. 2017)

NPCMJ 公開分に加えて、

- 青空文庫 約 5200 文
- 毎日新聞 1995 年 約 1600 文 (京大コーパスの一部と同じ)
- BCCWJ のテキストの一部 約 4400 文

などを含んだ約 4 万文。(内 1.4 万文は復元に元テキストが必要。)

- NPCMJ 公式版と比べると精度にばらつきがある
- web インターフェイスは付属しない

ツリーバンクを利用した言語研究に向けて

ガ・ノ交替

(1) a. 窓がない部屋

b. 窓のない部屋

(2) a. 太郎が書いた本

b. 太郎の書いた本

問: ノ格でマークされる主語と共起する述語には何らかの特徴があるか？

ツリーバンクを利用した言語研究に向けて

ガ・ノ交替

(1) a. 窓がない部屋

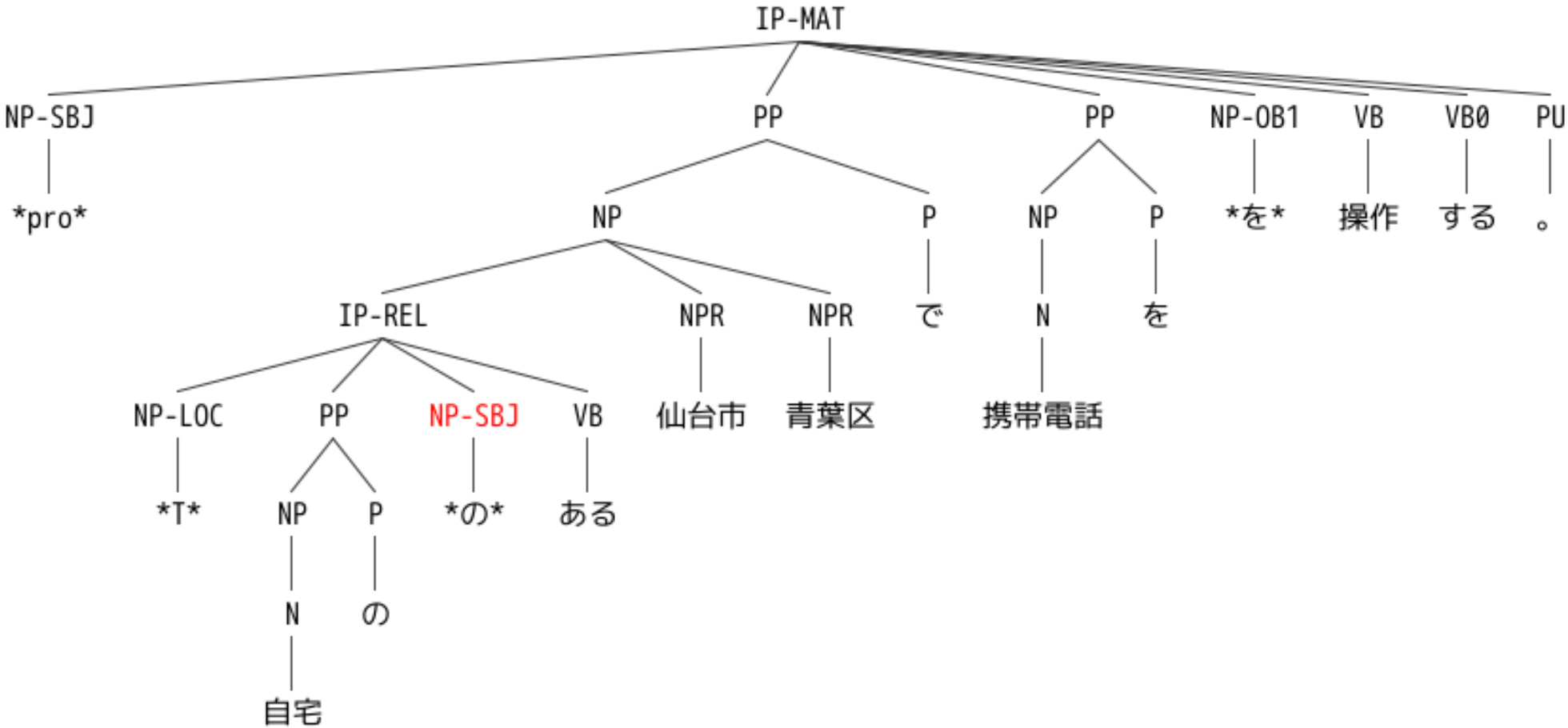
b. 窓のない部屋

(2) a. 太郎が書いた本

b. 太郎の書いた本

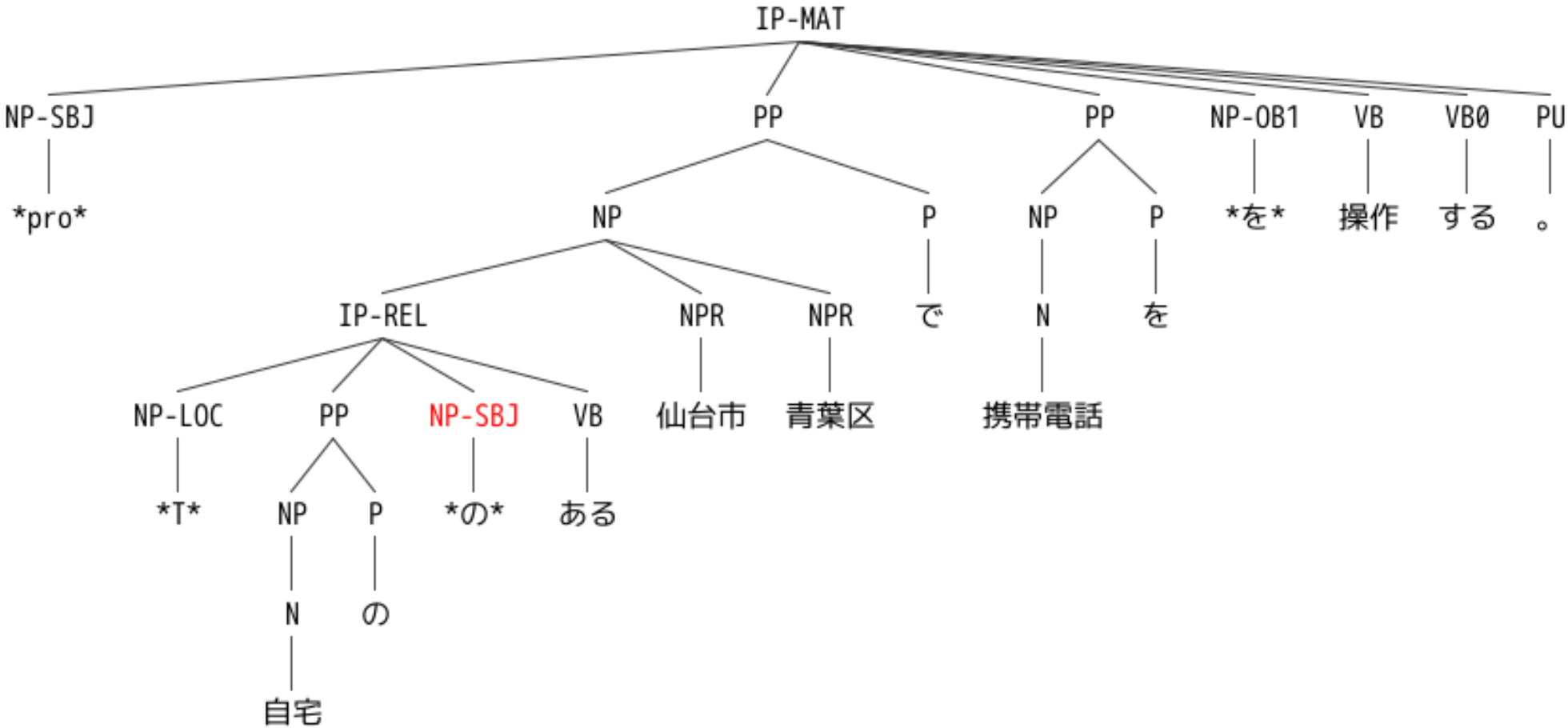
問: ノ格でマークされる主語と共起する述語には何らかの特徴があるか？

ガ・ノ交替のアノテーション



- ノ格主語を含む節を取ってくるだけなら簡単
- ノ格主語と共起する述語の頻度表を作るには、手で集計するか、さらにデータを加工する作業が必要

ガ・ノ交替のアノテーション



- ノ格主語を含む節を取ってくるだけなら簡単
- ノ格主語と共起する述語の頻度表を作るには、手で集計するか、さらにデータを加工する作業が必要

UNIX 哲学

The UNIX Programming Environment (Kernighan and Pike 1984)

UNIX を効果的なシステムとしているのは、プログラミングに対するある種のアプローチであり、コンピュータを使用する際の考え方 (philosophy) である。その考え方を一つの文で言い表すことは出来ないが、その核心にあるのは、システムの全体の力は、個々のプログラム単体よりもプログラム同士の連携に、より大きく依存しているという考え方である。多くの UNIX プログラムはそれ単体では非常に瑣末なことしかできないが、他のプログラムと組み合わせることで一般的で有用なツールとなる。

例: 単語の頻度表の作成

頻度表:

単語	出現回数
the	1686
and	869
to	799
a	672
of	606

例: 単語の頻度表の作成

Python (淺尾・李 2013)

```
freq = {} # からっぽのディクショナリを用意

datafile = open('h.txt')
for line in datafile:
  line = line.rstrip()
  words = line.split()

  # ディクショナリに頻度を反映させる
  for word in words:
    if word in freq:
      freq[word] += 1
    else:
      freq[word] = 1

# ディクショナリの中身を表示
for word in sorted(freq, key=freq.get, reverse=True):
  print(word + '\t' + str(freq[word]))
```

シェルコマンド

```
$ cat file.txt | one_word_per_line | sort | uniq -c | sort -nr
```

例: 単語の頻度表の作成

Python (淺尾・李 2013)

```
freq = {} # からっぽのディクショナリを用意

datafile = open('h.txt')
for line in datafile:
  line = line.rstrip()
  words = line.split()

  # ディクショナリに頻度を反映させる
  for word in words:
    if word in freq:
      freq[word] += 1
    else:
      freq[word] = 1

# ディクショナリの中身を表示
for word in sorted(freq, key=freq.get, reverse=True):
  print(word + '\t' + str(freq[word]))
```

シェルコマンド

```
$ cat file.txt | one_word_per_line | sort | uniq -c | sort -nr
```

フィルタによるテキストの加工

```
                                  ①                ②      ③        ④
$ cat alice.txt | one_word_per_line | sort | uniq -c | sort -nr
   1686 the
    869 and
    799 to
    672 a
    606 of
    ...
    ...
```

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
4. 結果を降順に並べ替える

フィルタによるテキストの加工

```
                                  ①                ②      ③        ④
$ cat alice.txt | one_word_per_line | sort | uniq -c | sort -nr
   1686 the
    869 and
    799 to
    672 a
    606 of
    ...
    ...
```

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
4. 結果を降順に並べ替える

シェルコマンドとプログラミング言語

Python:

- プログラムはデータの処理の仕方を記述する
  (繰り返しや条件分岐を含む)
- 処理対象のデータはタスクに応じて様々な型をとる
  (リスト、配列、...)

シェルコマンド:

- 単純なコマンド (フィルタと呼ばれる) の連鎖でデータを加工
  (処理の流れは常に一直線)
- 処理対象のデータはプレーンテキスト

シェルコマンドを用いた手法が効果的な状況

- プログラムの効率性よりも開発の効率性が重視される場合
  (とりあえず動くものを作って試してみたい時など)
- 高度なプログラミングの知識なしである程度複雑なデータの
  加工などをしたい場合

シェルコマンドとプログラミング言語

Python:

- プログラムはデータの処理の仕方を記述する
  (繰り返しや条件分岐を含む)
- 処理対象のデータはタスクに応じて様々な型をとる
  (リスト、配列、...)

シェルコマンド:

- 単純なコマンド (フィルタと呼ばれる) の連鎖でデータを加工
  (処理の流れは常に一直線)
- 処理対象のデータはプレーンテキスト

シェルコマンドを用いた手法が効果的な状況

- プログラムの効率性よりも開発の効率性が重視される場合
  (とりあえず動くものを作って試してみたい時など)
- 高度なプログラミングの知識なしである程度複雑なデータの
  加工などをしたい場合

ノ格主語と共起する述語の頻度表を作る

手順:

1. ノ格主語を含む節をコーパスから抽出 ⇒ 木構造の ‘grep’
2. 抽出した部分木から述語の部分を切り出す ⇒ 木構造の ‘sed’
3. 数を数えて頻度表を作る ⇒ 普通の UNIX ツール

ノ格主語と共起する述語の頻度表を作る

手順:

1. ノ格主語を含む節をコーパスから抽出 ⇒ 木構造の 'grep'
2. 抽出した部分木から述語の部分を切り出す ⇒ 木構造の 'sed'
3. 数を数えて頻度表を作る ⇒ 普通の UNIX ツール

```
(IP-REL (NP-LOC *T*)
        (PP (NP (N 自宅))
            (P の))
        (NP-SBJ *の*)
        (VB ある))
```

ノ格主語と共起する述語の頻度表を作る

手順:

1. ノ格主語を含む節をコーパスから抽出 ⇒ 木構造の ‘grep’
2. 抽出した部分木から述語の部分を切り出す ⇒ 木構造の ‘sed’
3. 数を数えて頻度表を作る ⇒ 普通の UNIX ツール

```
(IP

                (VB ある))
```

フィルタによるツリー検索・加工

```
$ tgrep2 -c Keyaki.tgrep '/IP/ < (NP-SBJ|NP-SBJ2 < *の*)' |   # (1)
      tsurgeon.sh ga-no-conv-pred-extract.tsurgeon |          # (2)
      sort | uniq -c | sort -nr                               # (3)
       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))
       ...
       ...
```

1. ノ格主語を含む節をコーパスから抽出　⇒ 木構造の 'grep'
2. 抽出した部分木から述語の部分を切り出す　⇒ 木構造の 'sed'
3. 数を数えて頻度表を作る　⇒ 普通の UNIX ツール

まとめ

ツリーバンクは複雑なデータ構造の言語資源だが、

- 単純なステップに区切って検索やデータの加工をすることで比較的簡単に必要な情報を抽出することができる
- コマンドラインから用いるパイプライン・ツールを組み合わせる UNIX 系の OS で伝統的に用いられている手法が特に便利

他の応用の可能性:

- アノテーションの情報量が密なデータ資源の処理
- 検索ツールやインターフェースの開発が発展途上であるデータ資源の処理

謝辞

ご清聴ありがとうございました。

本研究は以下の研究費の助成を受けました。

- 科研費 基盤研究 (B) 15H03210 「統語・意味解析情報タグ付きコーパス開発用アノテーション研究：複文を中心に」 (平成 27 年度～平成 31 年度)

NPCMJ コーパスの開発、また本研究は、以下の方々の貢献に支えられています (敬称略)。記して感謝いたします。

- NPCMJ コーパス開発チーム
  Alastair Butler、Stephen Wright Horn、長崎郁、Prashant Pardeshi、吉本啓、窪田愛 (昨年度)
- NPCMJ コーパス・アノテーター@国語研、東北大学
- NPCMJ コーパス・フィードバックチーム@神戸大学

付録 1: tgrep2 と tsurgeon

tgrep2 はその名の通り木構造に対する grep と考えればよい

基本的な使い方: ノード間の支配関係や前後関係を指定して検索

⇒ 指定した構造にマッチする部分木 (あるいはマッチした部分木を含む文全体の木) を返す

例: けやきコーパスでノ格主語を含む節を検索する

検索式: `/IP/ < (NP-SBJ|NP-SBJ2 < *の*)`

- 「*の* 」という文字列を直接支配する NP-SBJ または NP-SBJ2 を含む節 (IP) を検索

付録 1: tgrep2 と tsurgeon

tsurgeon は木構造に対する sed のようなものである

例: IP ノードの子ノードの中で、述語でない (つまり VB や ADJ でない) ノードを消す

- 「I, A, V の文字で始まらない非終端ノードを消す」というふうに書けば、大体欲しい結果を得ることができる。

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

delete x
```

付録 1: tgrep2 と tsurgeon

ソースコード 1: 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
```

付録 2: 既存の言語資源と NPCMJ の違い

BCCWJ vs. NPCMJ/けやき

	BCCWJ	NPCMJ/けやき
サイズ	大 (1 億語)	小 (64 万語 *)
粒度	形態論情報のみ	統語構造
主な用途	量的分析	質的分析

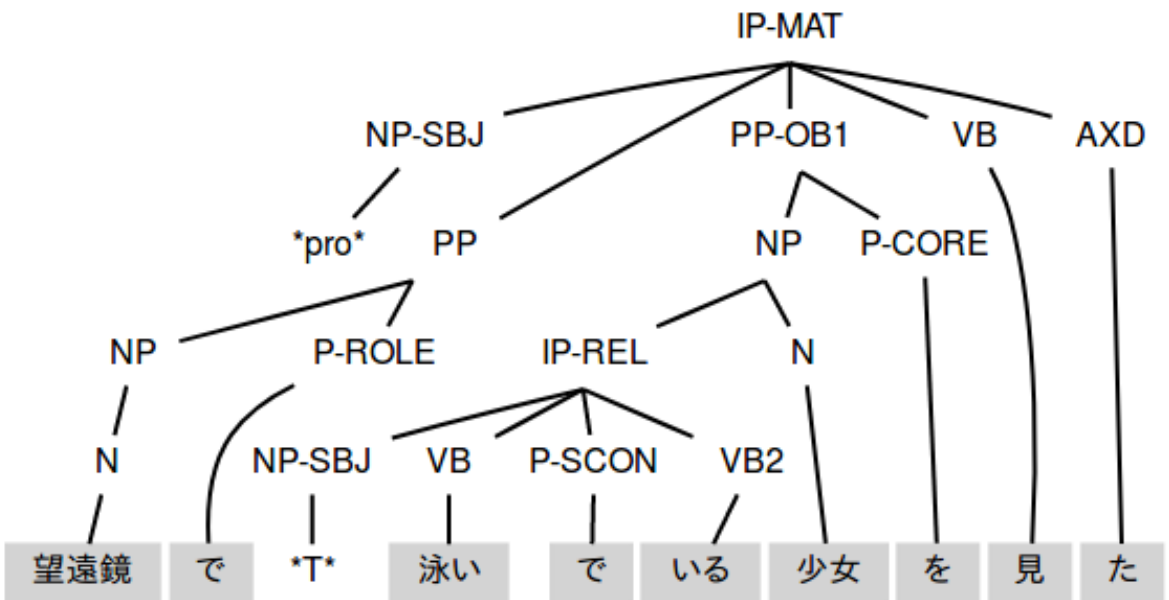
* けやき コーパスの語数

付録 2: 既存の言語資源と NPCMJ の違い

京大コーパス

望遠——┐
　　鏡で——┐
泳いでいる——┐　　│
　　少女を——┤
　　見た

NPCMJ パーザーでの解析結果



付録 2: 既存の言語資源と NPCMJ の違い

係り受けコーパスと句構造コーパス

係り受けコーパスと句構造コーパスとのいくつかの違い:

- 句レベルでのカテゴリの情報の有無
- 文法関係の情報の有無

言語学者にとっての使いやすさを決める要因:

- 見慣れた形式でアノテーションがされているか
- 検索用ツールが整備されているか

付録 3: NPCMJ/けやきコーパスを用いた言語研究の例

日本語の「テ形」と CSC

(3) a. *太郎が [[雑誌を買って]、 [__ 読んだ]] 本

b. 太郎が [[__ 買って]、[__ 読んだ]] 本

例外はあるか?

(4) 太郎が [[紀伊国屋に行って]、[__ 買った]] 本

→ こういう例外を Google から拾ってくるのは大変!

探したい構造:

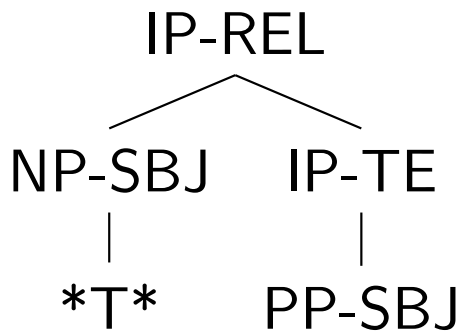
$[_{\mathrm{RC}}$ . . . [. . . て] [. . . た]] 名詞

↑

どちらかの節からのみ抜き出しが起こっている

付録 3: NPCMJ/けやきコーパスを用いた言語研究の例

検索式: IP-REL < (NP-SBJ < *T*) < (IP-TE < PP-SBJ)



- 最初の節には主語の空所があり
- 二つ目の節には空所でない主語がある
- 関係節

Butler, Alastair, 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.

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

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, 1–28 頁.

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.