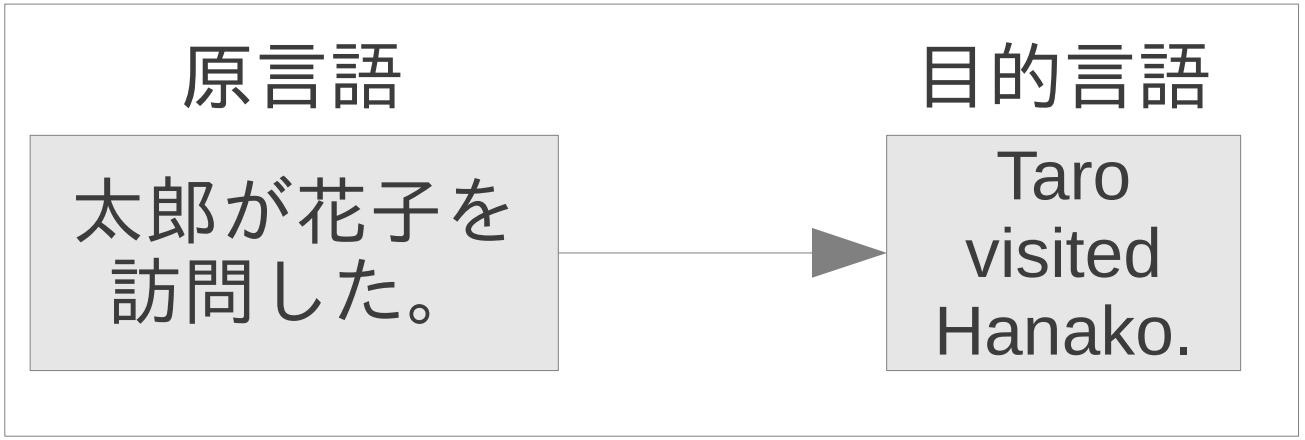
機械翻訳

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)
2013 年 1 月 17 日

機械翻訳

- 原言語から目的言語へと自動的に翻訳



- 近年に著しい発展と実用化

Google translate

excite

Nara Institute of Science and Technology NAIST

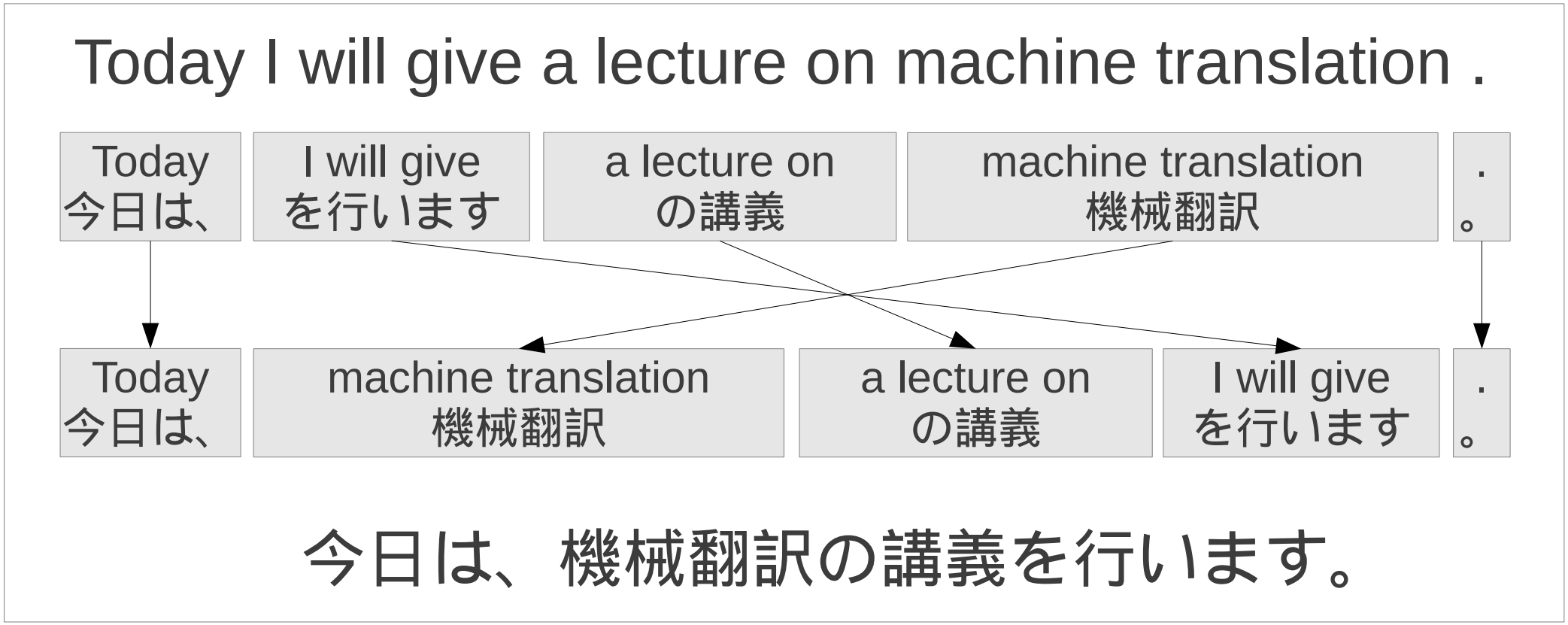
KYOTO UNIVERSITY KYOTO JAPAN FOUNDED 1897

機械翻訳の仕組み

Today I will give a lecture on machine translation .

機械翻訳の仕組み

- 文を翻訳可能なパターンに区切り、並べ替え



問題！

- 各文に対して膨大な可能性

花子 が 太郎 に 会った
Hanako met Taro
Hanako met to Taro
Hanako ran in to Taro
Taro met Hanako
The Hanako met the Taro

- どれが訳としてふさわしいか？
  - どの単語を選択するか→語彙選択の問題
  - どの並びが相応しいか→並べ替えの問題

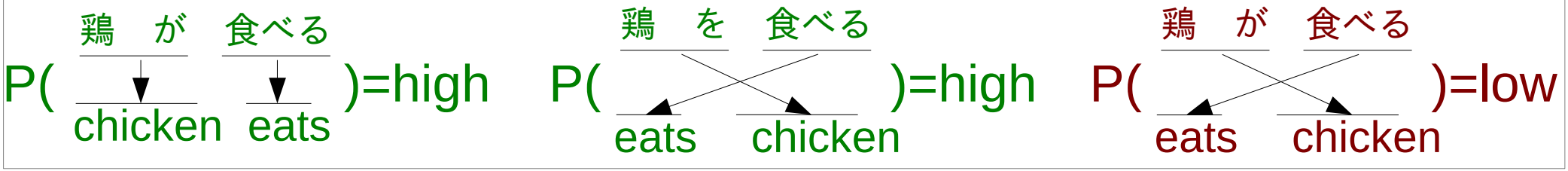
（フレーズベース）統計的機械翻訳

- 翻訳モデル (TM)

P(“ 今日” |“today”) = high　　P(“ 今日 は 、 ” |“today”) = medium

P(“ 昨日” |“today”) = low

- 並べ替えモデル (RM)



- 言語モデル (LM)

P(“Taro met Hanako”)=high　　P(“the Taro met the Hanako”)=high

機械翻訳システムの構築

- 各モデルを対訳文から学習

対訳文

太郎が花子を訪問した。
Taro visited Hanako.

花子にプレゼントを渡した。
He gave Hanako a present.

...

モデル

翻訳モデル

並べ替えモデル

言語モデル

パターン学習の例

- 日本語メニューのあるイタリア料理やに行ったら

チーズムース
Mousse di formaggi

タリアテッレ 4種のチーズソース
Tagliatelle al 4 formaggi

本日の鮮魚
Pesce del giorno

鮮魚のソテー お米とグリーンピース添え
Filetto di pesce su “Risi e Bisi”

ドルチェとチーズ
Dolce e Formaggi

- 「formaggi」の日本語訳は？「pesce」？「e」？

パターン学習の例

- 日本語メニューのあるイタリア料理やに行ったら

チーズムース
Mousse di formaggi

タリアテッレ 4種のチーズソース
Tagliatelle al 4 formaggi

本日の鮮魚
Pesce del giorno

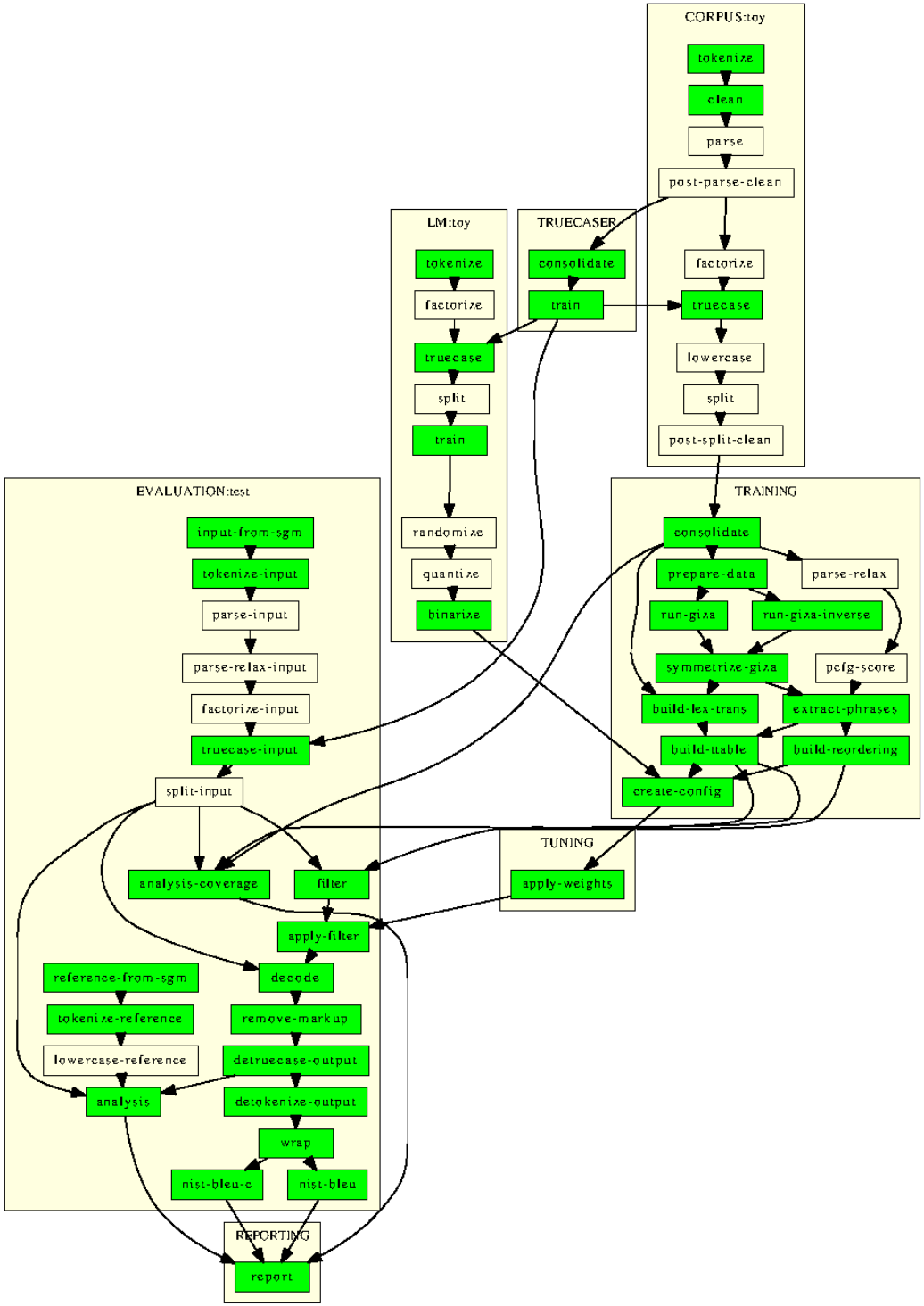
鮮魚のソテー お米とグリーンピース添え
Filetto di pesce su “Risi e Bisi”

ドルチェとチーズ
Dolce e Formaggi

- 「formaggi」の日本語訳は？「pesce」？「e」？

言うは易く
行うは難し
(easier said than done)

機械翻訳の学習過程
フロー図→



セミナーの話

- 言語モデル
- かな漢字変換
- フレーズベース統計的機械翻訳
- フレーズベースモデルの構築
- 翻訳文の評価
- 重み調整（チューニング）
- 構文解析
- 階層的フレーズベース翻訳
- 統語情報を用いた翻訳

演習の事前準備 (1)

- Linux (Ubuntu/Cent OS など)
  - 端末から以下のコマンドを実行
  - 必要なパッケージを apt-get でインストール

```
$ sudo apt-get install automake autoconf g++ libtool libboost-all-dev perl python
```

- Mac
  - Xcode をインストール
  - 端末から以下のコマンドを実行
  - 必要なパッケージを apt-get でインストール

```
$ sudo port install automake autoconf libtool boost
```

演習の事前準備 (2)

- Windows
  - http://www.cygwin.com/ から Cygwin をインストール
  - インストールする際、以下のパッケージを選択：

    Devel -> automake autoconf gcc-g++ libtool make
    Libs -> libboost-devel
    Perl -> perl
    Python -> python

演習の事前準備 (3)

- 全 OS
  - Java の SDK をダウンロードしてインストール

http://www.oracle.com/technetwork/java/javase/downloads/index.html

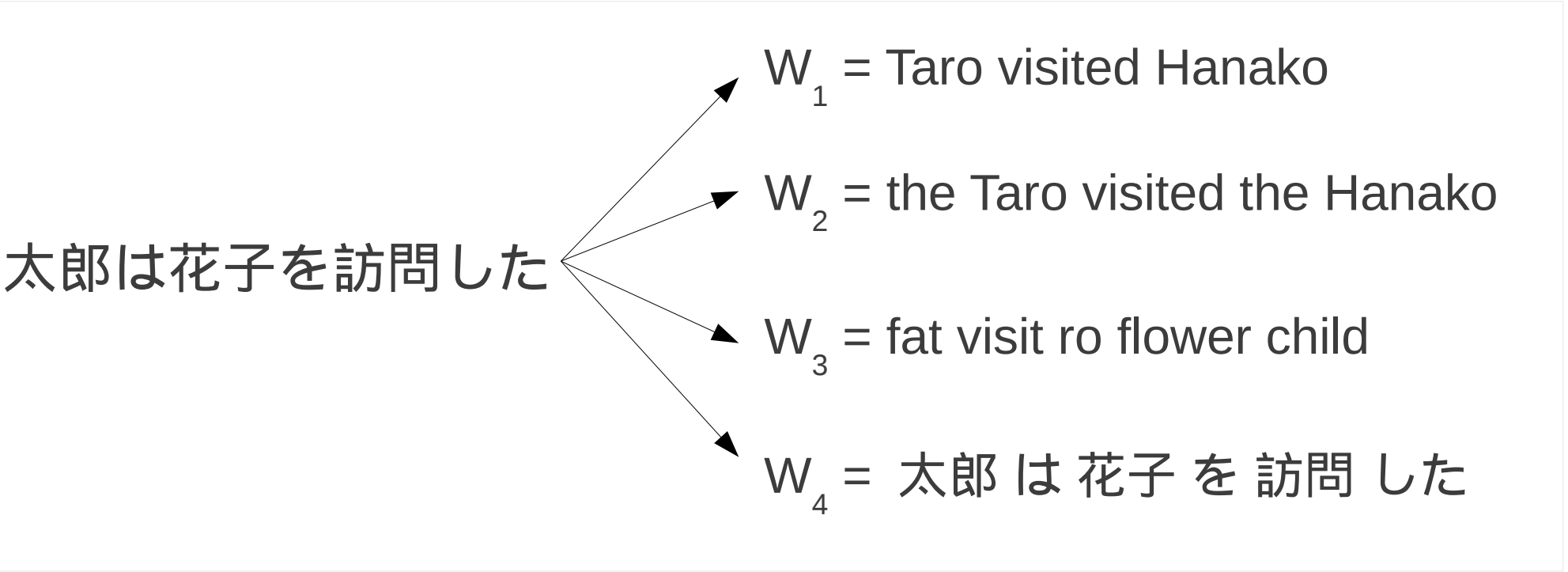
演習の事前準備 (4)

- 演習に必要なソフト・データは配布資料の alagin/download に存在
- プログラム： Moses, GIZA++, pialign, IRSTLM, KyTea, Stanford Parser, Eda, AlignmentTool, lader
- データ：京都フリー翻訳タスクで用いられる Wikipedia 京都関連記事
- これを全てインストールすることは手間がかかる
- ホームディレクトリに解凍し、インストールスクリプトを実行：
  - script/00-setup.sh

言語モデル

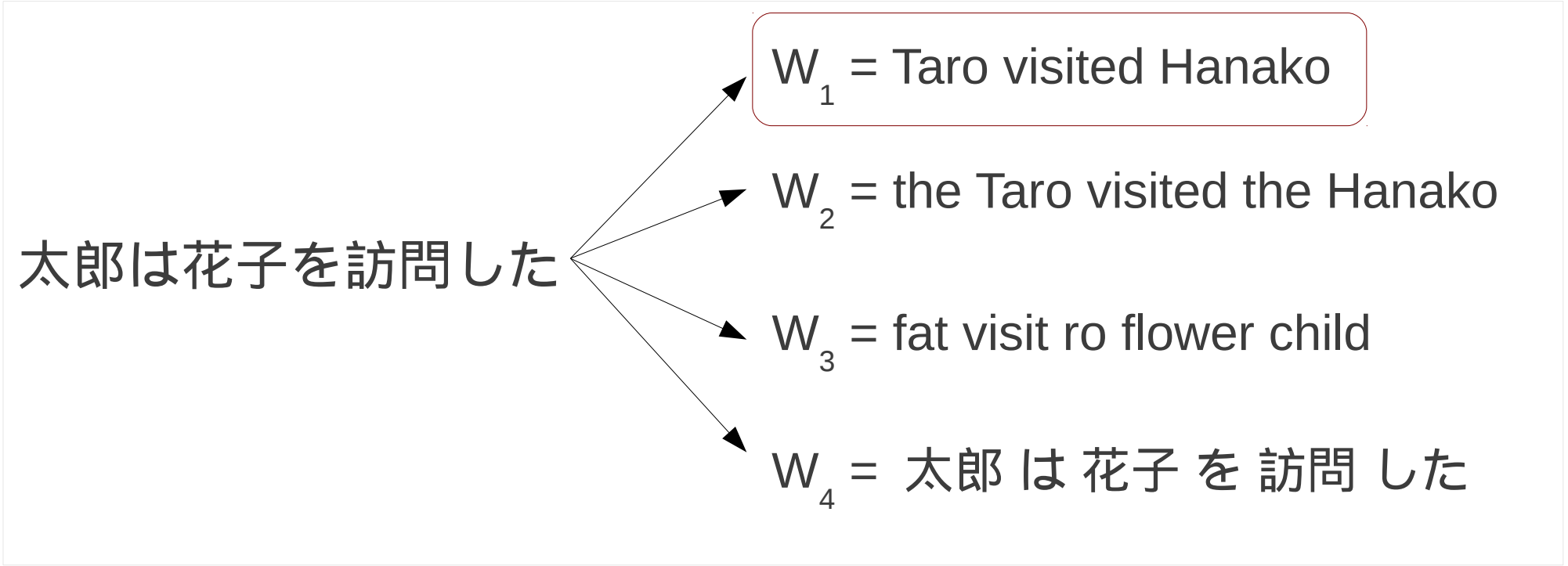
言語モデル？

- 日英翻訳を行いたい時に、どれが正解？



言語モデル？

- 日英翻訳を行いたい時に、どれが正解？



- 言語モデルは「もっともらしい」文を選んでくれる

確率的言語モデル

- 言語モデルが各文に確率を与える

W_1 = taro visited hanako	$P(W_1) = 4.021 * 10^{-3}$
W_2 = the taro visited the hanako	$P(W_2) = 8.932 * 10^{-4}$
W_3 = fat visit ro flower child	$P(W_3) = 2.432 * 10^{-7}$
W_4 = 太郎 は 花子 を 訪問 した	$P(W_4) = 9.124 * 10^{-23}$

- $P(W_1) > P(W_2) > P(W_3) > P(W_4)$ が望ましい
  - (日本語の場合は $P(W_4) > P(W_1), P(W_2), P(W_3)$ ？)

文の確率計算

- 文の確率が欲しい

$$W = \text{taro visited hanako}$$

- 変数で以下のように表す

$$P(|W| = 3, w_1 = \text{"taro"}, w_2 = \text{"visited"}, w_3 = \text{"hanako"})$$

文の確率計算

- 文の確率が欲しい

W = taro visited hanako

- 変数で以下のように表す (連鎖の法則を用いて):

P(|W| = 3, w_1="taro", w_2="visited", w_3="hanako") =

P(w_1="taro" | w_0 = "<s>")

* P(w_2="visited" | w_0 = "<s>", w_1="taro")

* P(w_3="hanako" | w_0 = "<s>", w_1="taro", w_2="visited")

* P(w_4="</s>" | w_0 = "<s>", w_1="taro", w_2="visited", w_3="hanako")

注:
文頭「 <s> 」と文末「 </s> 」記号

注:
P(w_0 = <s>) = 1

確率の漸次的な計算

- 前のスライドの積を以下のように一般化

$$P(W) = \prod_{i=1}^{|W|+1} P(w_i | w_0 \dots w_{i-1})$$

- 以下の条件付き確率の決め方は？

$$P(w_i | w_0 \dots w_{i-1})$$

最尤推定による確率計算

- コーパスの単語列を数え上げて割ることで計算

$$P(w_i|w_1 \ldots w_{i-1}) = \frac{c(w_1 \ldots w_i)}{c(w_1 \ldots w_{i-1})}$$

i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

P(live | <s> i) = c(<s> i live)/c(<s> i) = 1 / 2 = 0.5

P(am | <s> i) = c(<s> i am)/c(<s> i) = 1 / 2 = 0.5

最尤推定の問題

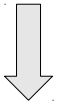
- 頻度の低い現象に弱い：

学習：

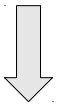
i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

確率計算：

<s> i live in nara . </s>



P(nara|<s> i live in) = 0/1 = 0



P(W=<s> i live in nara . </s>) = 0

1-gram モデル

- 履歴を用いないことで低頻度の現象を減らす

$$P(w_i|w_1 \ldots w_{i-1}) \approx P(w_i) = \frac{c(w_i)}{\sum_{\tilde{w}} c(\tilde{w})}$$

i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

P(nara) = 1/20 = 0.05
P(i) = 2/20 = 0.1
P(</s>) = 3/20 = 0.15

P(W=i live in nara . </s>) =
0.1 * 0.05 * 0.1 * 0.05 * 0.15 * 0.15 = $5.625 * 10^{-7}$

未知語の対応

- 未知語が含まれる場合は 1-gram でさえも問題あり

i live in osaka . </s>
i am a graduate student . </s> → P(i) = 2/20 = 0.1
my school is in nara . </s>

P(nara) = 1/20 = 0.05
P(kyoto) = 0/20 = 0

- 多くの場合（例：音声認識）、未知語が無視される
- 他の解決法
  - 少しの確率を未知語に割り当てる ($\lambda_{unk} = 1-\lambda_1$)
  - 未知語を含む語彙数を N とし、以下の式で確率計算

$$P(w_i) = \lambda_1 P_{ML}(w_i) + (1-\lambda_1)\frac{1}{N}$$

未知語の例

- 未知語を含む語彙数： $N=10^6$
- 未知語確率： $\lambda_{unk}=0.05$ ($\lambda_1 = 0.95$)

$$P(w_i) = \lambda_1 P_{ML}(w_i) + (1-\lambda_1)\frac{1}{N}$$

P(nara) = 0.95*0.05 + 0.05*(1/10^6) = 0.04750005

P(i) = 0.95*0.10 + 0.05*(1/10^6) = 0.09500005

P(kyoto) = 0.95*0.00 + 0.05*(1/10^6) = 0.00000005

1-gram モデルは語順を考慮しない

- 以下の確率は同等

P_{uni}(w=taro visited hanako) =
P(w=taro) * P(w=visited) * P(w=hanako) * P(w=</s>)

=

P_{uni}(w=hanako visited taro) =
P(w=taro) * P(w=visited) * P(w=hanako) * P(w=</s>)

1-gram モデルは単語の関係性を考慮しない

- 文法的な文：（名詞と活用が一致）

P_{uni}(w=i am) =
P(w=i) * P(w=am) * P(w=</s>)

P_{uni}(w=we are) =
P(w=we) * P(w=are) * P(w=</s>)

- 文法的でない文：（名詞と活用が矛盾）

P_{uni}(w=we am) =
P(w=we) * P(w=am) * P(w=</s>)

P_{uni}(w=i are) =
P(w=i) * P(w=are) * P(w=</s>)

しかし、確率は上記の文と同等

文脈を考慮することで解決！

- 1-gram モデルは文脈を考慮しない

$$P(w_i|w_0 \dots w_{i-1}) \approx P(w_i)$$

- 2-gram は 1 単語の文脈を考慮

$$P(w_i|w_0 \dots w_{i-1}) \approx P(w_i|w_{i-1})$$

- 3-gram は 2 単語の文脈を考慮

$$P(w_i|w_0 \dots w_{i-1}) \approx P(w_i|w_{i-2}w_{i-1})$$

- 4-gram 、 5-gram 、 6-gram などなど

n-gram 確率の最尤推定

- n 単語と n-1 単語からなる文字列の頻度を利用

$$P(w_i|w_{i-n+1}\cdots w_{i-1}) = \frac{c(w_{i-n+1}\cdots w_i)}{c(w_{i-n+1}\cdots w_{i-1})}$$

i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

n=2 →
P(osaka | in) = c(in osaka)/c(in) = 1 / 2 = 0.5
P(nara | in) = c(in nara)/c(in) = 1 / 2 = 0.5

低頻度 n-gram の問題

- n-gram 頻度が 0→n-gram 確率も 0

P(osaka | in) = c(in osaka)/c(in) = 1 / 2 = 0.5
P(nara | in) = c(in nara)/c(in) = 1 / 2 = 0.5
P(school | in) = c(in school)/c(in) = 0 / 2 = **0**!!

- 1-gram モデルと同じく、線形補間を用いる

2-gram: $$P(w_i|w_{i-1}) = \lambda_2 P_{ML}(w_i|w_{i-1}) + (1-\lambda_2) P(w_i)$$

1-gram: $$P(w_i) = \lambda_1 P_{ML}(w_i) + (1-\lambda_1) \frac{1}{N}$$

補間係数の選択法：グリッド探索

- λ_2 と λ_1 の様々な値を試し、尤度が最も高くなるように選択

$$\lambda_2 = 0.95, \lambda_1 = 0.95$$
$$\lambda_2 = 0.95, \lambda_1 = 0.90$$
$$\lambda_2 = 0.95, \lambda_1 = 0.85$$
$$\dots$$
$$\lambda_2 = 0.95, \lambda_1 = 0.05$$
$$\lambda_2 = 0.90, \lambda_1 = 0.95$$
$$\lambda_2 = 0.90, \lambda_1 = 0.90$$
$$\dots$$
$$\lambda_2 = 0.05, \lambda_1 = 0.10$$
$$\lambda_2 = 0.05, \lambda_1 = 0.05$$

問題：

選択肢が多すぎる
→ 選択に時間がかかる！

全ての n-gram に対して同じ λ
→ 尤度が最適とは限らない！

文脈を考慮した補間係数の選択

頻度の高い単語： Tokyo	頻度の低い単語： Tottori
c(Tokyo city) = 40 c(Tokyo is) = 35 c(Tokyo was) = 24 c(Tokyo tower) = 15 c(Tokyo port) = 10 …	c(Tottori is) = 2 c(Tottori city) = 1 c(Tottori was) = 0
ほとんどの 2-gram が既観測 → 大きなλが最適	未観測の 2-gram が多い → 小さなλが最適

- 補間係数の選択にも文脈を考慮：

$$P(w_i|w_{i-1}) = \lambda_{w_{i-1}} P_{ML}(w_i|w_{i-1}) + (1-\lambda_{w_{i-1}}) P(w_i)$$

Witten-Bell 平滑化

- $\lambda_{w_{i-1}}$ を選ぶ方法の１つ

$$\lambda_{w_{i-1}} = 1 - \frac{u(w_{i-1})}{u(w_{i-1}) + c(w_{i-1})}$$

$u(w_{i-1})$ = w_{i-1} の後に続く単語の異なり数

- 例えば、

c(Tottori is) = 2　c(Tottori city) = 1
c(Tottori) = 3　u(Tottori) = 2

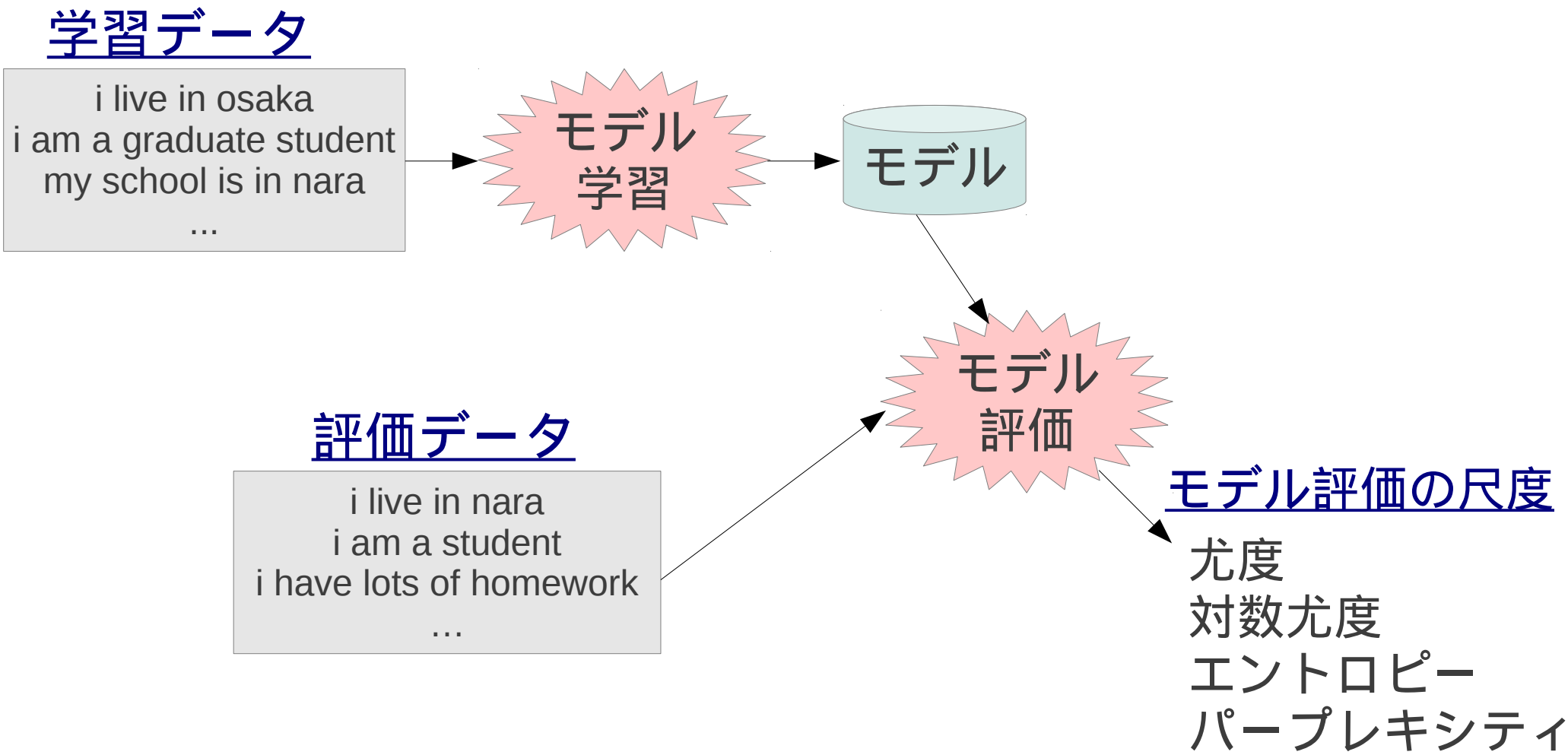
$$\lambda_{Tottori} = 1 - \frac{2}{2+3} = 0.6$$

c(Tokyo city) = 40 c(Tokyo is) = 35 ...
c(Tokyo) = 270　u(Tokyo) = 30

$$\lambda_{Tokyo} = 1 - \frac{30}{30+270} = 0.9$$

言語モデルの評価の実験設定

- 学習と評価のための別のデータを用意



尤度

- 尤度はモデル M が与えられた時の観測されたデータ (評価データ W_{test}) の確率

$$P(W_{test}|M) = \prod_{w \in W_{test}} P(w|M)$$

i live in nara	P(w="i live in nara"|M) =	$2.52*10^{-21}$
		x
i am a student	P(w="i am a student"|M) =	$3.48*10^{-19}$
		x
my classes are hard	P(w="my classes are hard"|M) =	$2.15*10^{-34}$
		=
		$1.89*10^{-73}$

対数尤度

- 尤度の値が非常に小さく、桁あふれがしばしば起こる
- 尤度を対数に変更することで問題解決

$$\log P(W_{test}|M) = \sum_{w \in W_{test}} \log P(w|M)$$

i live in nara i am a student my classes are hard	log P(w="i live in nara"|M) = -20.58 + log P(w="i am a student"|M) = -18.45 + log P(w="my classes are hard"|M) = -33.67 = -72.60

エントロピー

- エントロピー H は負の底２の対数尤度を単語数で割った値

$$H(W_{test}|M) = \frac{1}{|W_{test}|} \sum_{w \in W_{test}} -\log_2 P(w|M)$$

i live in nara
i am a student
my classes are hard

$\log_2$ P(w="i live in nara"|M)= 68.43

$\log_2$ P(w="i am a student"|M)= + 61.32

$\log_2$ P(w="my classes are hard"|M)= + 111.84

単語数: / 12

= 20.13

* </s> を単語として数えることもあるが、ここでは入れていない

エントロピーと情報圧縮

- エントロピーHは与えられたデータを圧縮するのに必要なビット数でもある（シャノンの情報理論により）

$$H = \frac{1}{|W_{test}|} \sum_{w \in W_{wtest}} -\log_2 P(w|M)$$

a bird a cat a dog a </s>

Encoding

a	→ 1
bird	→ 000
cat	→ 001
dog	→ 010
</s>	→ 011

P(w= “a”) = 0.5 $-\log_2 0.5 = 1$

P(w= “bird”) = 0.125 $-\log_2 0.125 = 3$

P(w= “cat”) = 0.125 $-\log_2 0.125 = 3$

P(w= “dog”) = 0.125 $-\log_2 0.125 = 3$

P(w= “</s>”) = 0.125 $-\log_2 0.125 = 3$

10001001101011

パープレキシティ

- ２のエントロピー乗

$$PPL = 2^{H}$$

- 一様分布の場合は、選択肢の数に当たる

$$V = 5 \quad H = -\log_2 \frac{1}{5} \quad PPL = 2^{H} = 2^{-\log_2 \frac{1}{5}} = 2^{\log_2 5} = 5$$

カバレージ

- 評価データに現れた単語（n-gram）の中で、モデルに含まれている割合

a bird a cat a dog a </s>

“dog” は未知語

カバレージ : 7/8 *

* 文末記号を除いた場合は → 6/7

研究

- n-gram に勝てるものはあるのか？ [Goodman 01]
  - 計算がシンプルで高速
  - 探索アルゴリズムと相性が良い
  - シンプルな割に強力
- その他の手法
  - 統語情報を利用した言語モデル [Charniak 03]
  - ニューラルネット言語モデル [Bengio 06]
  - モデル M [Chen 09]
  - などなど…

(機械翻訳で広く使われる)
言語モデルツールキット

- SRILM:
  - 最も広く使われる言語モデル学習・格納ツールキット
  - オープンソース、研究利用は無料、商用利用は有料
- IRSTLM:
  - 他の言語モデル学習ツールキット
  - オープンソース、研究、商用はともに無料
- KenLM:
  - 学習はできないが、効率の良い言語モデル格納と高速なアクセスが売り
  - オープンソース、研究、商用はともに無料

IRSTLM で言語モデル学習 (1)

- 配布資料の download/irstlm-5.80.01.tgz
- 事前準備で usr/bin にインストール済み
- 言語モデル構築の 4 ステップ
  - 1: 文頭・文末記号を追加
  - 2: 言語モデル学習（確率・平滑化計算）
  - 3: パープレキシティの計算
  - 4: 翻訳用のバイナリ化

IRSTLM で言語モデル学習 (2)

- 利用するデータは data/tok/kyoto-train.ja
- まず lm ディレクトリを作成

```
$ mkdir lm
```

- IRSTLM ディレクトリを指定

```
$ export IRSTLM=$HOME/alagin/usr
```

IRSTLM で言語モデル学習 (3)

- IRSTLM の add-start-end.sh で文頭・文末記号の付加

```
$ usr/bin/add-start-end.sh < data/tok/kyoto-train.ja > lm/kyoto-train.sb.ja
```

- 付与されたデータを閲覧：

```
$ head lm/kyoto-train.sb.ja
</s>
<s> 雪舟 （ せっしゅう 、 1420 年 （ 応永 27 年 ） - 1506 年 （ 永正 3 年 ）
） は 号 で 、 15 世紀 後半 室町 時代 に 活躍 し た 水墨 画 家 ・ 禅僧 で 、 画聖
と も 称え られ る 。 </s>
<s> 日本 の 水墨 画 を 一変 さ せ た 。 </s>
```

IRSTLM で言語モデル学習 (3)

- n-gram 確率の計算と ARPA 形式への変換

```
$ usr/bin/build-lm.sh -i lm/kyoto-train.sb.ja \
    -t ./tmp -p -s improved-kneser-ney -o lm/kyoto-train.ja.lm
$ usr/bin/compile-lm --text yes lm/kyoto-train.ja.lm.gz lm/kyoto-train.ja.arpa
```

- 言語モデルファイルの閲覧

```
$ head lm/kyoto-train.ja.arpa
\data\
ngram 1= 146902
ngram 2= 1517994
ngram 3= 1040240

\1-grams:
-6.80151    <s> -1.03883
-1.4588 </s>
-5.22748    雪舟 -0.483976
```

log(P(雪舟))

log(P(< バックオフ >| 雪舟))

IRSTLM で言語モデル学習 (4)

- 開発データ data/tok/kyoto-dev.ja で言語モデルの評価：

```
$ usr/bin/compile-lm lm/kyoto-train.ja.lm.gz --eval data/tok/kyoto-dev.ja
…
%% Nw=26856 PP=130.57 PPwp=10.87 Nbo=11694 Noov=145 OOV=0.54%
…
```

単語数

パープレキシティ

未知語による
パープレキシティ

低次元 n-gram への
バックオフ回数

未知語数

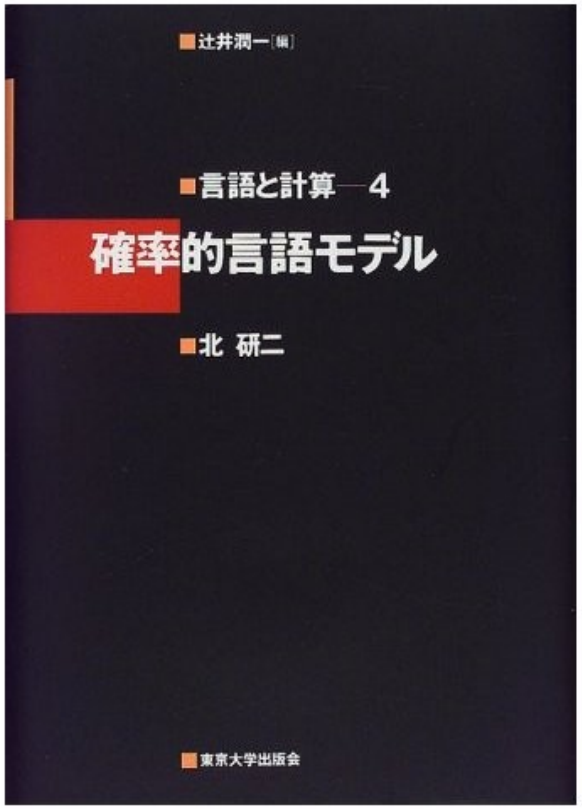
未知語率

IRSTLM で言語モデル学習 (5)

- 効率化のため、モデルをバイナリ化

```
$ mosesdecoder/bin/build_binary lm/kyoto-train.ja.arpa lm/kyoto-train.ja.blm
```

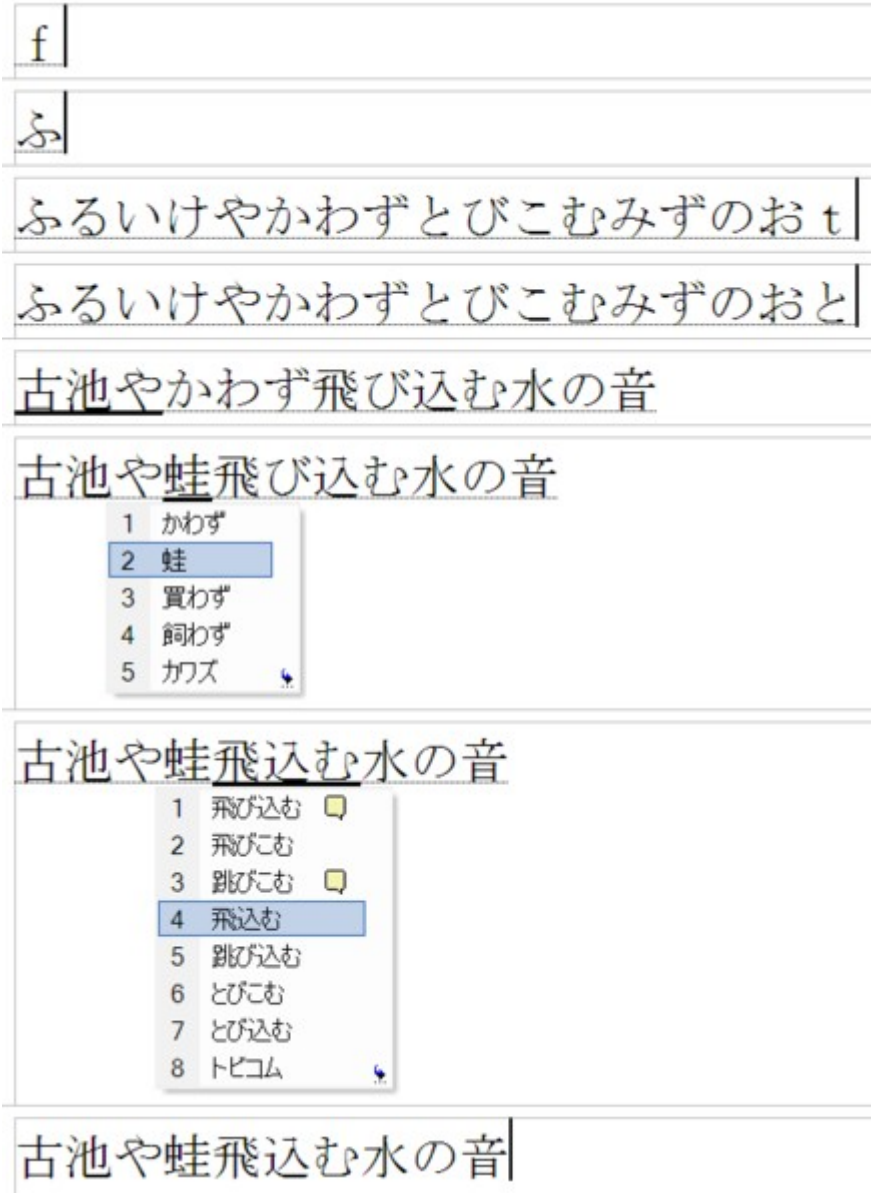
更に勉強するには



- 自然言語処理プログラミングチュートリアル (1,2 回目)
  http://www.phontron.com/teaching.php
- “A bit of progress in language modeling” [Goodman 01]

かな漢字変換

この画面はご存知ですか？



[img: Wikipedia]

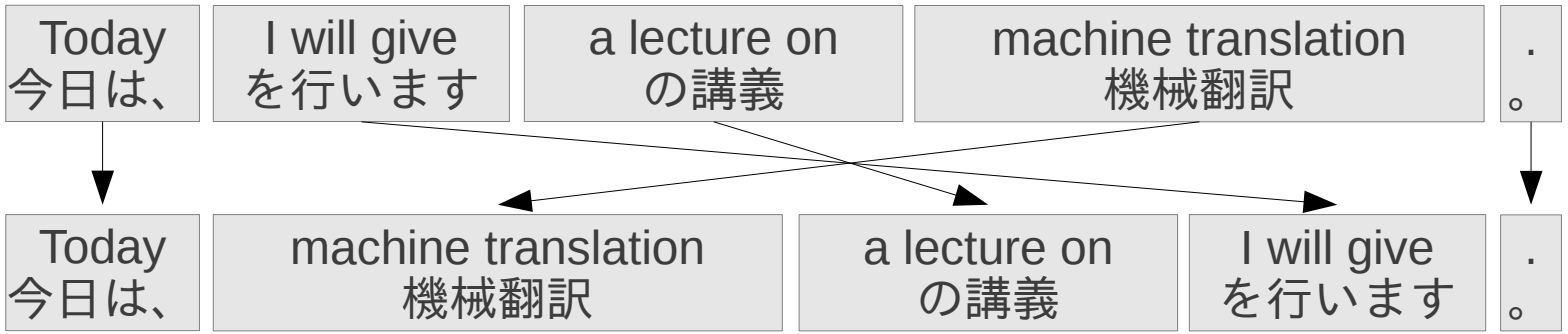
日本語入力

- インタフェース
- 予測提示
- かな漢字変換
- (辞書追加などなど)

機械翻訳とかな漢字変換

フレーズベース機械翻訳

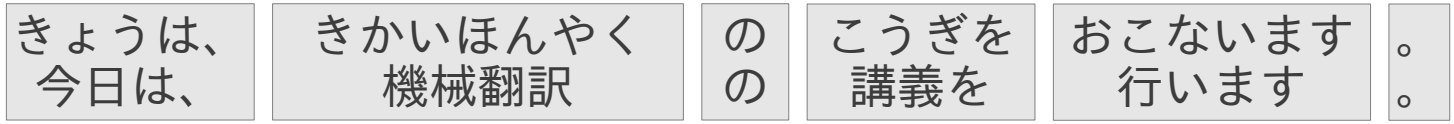
Today I will give a lecture on machine translation .



今日は、機械翻訳の講義を行います。

かな漢字変換

きょうは、きかいほんやくのこうぎをおこないます。



今日は、機械翻訳の講義を行います。

選択肢が膨大！

かなかんじへんかんはにほんごにゅうりょくのいちぶ

かな漢字変換は日本語入力の一部 良い！
仮名漢字変換は日本語入力の一部 良い？
かな漢字変換は二本後入力の一部 悪い
家中ん事変感歯にﾎﾝ御乳力の胃治舞 ?!?!

…

- 良い候補と悪い候補を区別するには？

かな漢字変換の確率モデル [森 +98]

- ひらがな文が与えられた場合、最も確率の高い かな漢字混じり文を計算

かなかんじへんかんはにほんごにゅうりょくのいちぶ

↓

かな漢字変換は日本語入力の一部

$$\underset{E}{\operatorname{argmax}}\, P\left(E \middle| F\right)$$

- これをどうやってモデル化？

系列に対する生成モデル

- ベイズ則で確率を分解

$$\begin{aligned}\underset{E}{\operatorname{argmax}}\, P(E|F) &= \underset{E}{\operatorname{argmax}}\, \frac{P(F|E)P(E)}{P(F)} \\ &= \underset{E}{\operatorname{argmax}}\, P(F|E)P(E)\end{aligned}$$

変換モデル (TM)
漢字とかなの関係を考慮
「にほん」はたぶん「日本」

言語モデル (LM)
前の漢字と次の漢字の関係を考慮
「日本」の後、「語」が現れやすい

モデルの例

- 言語モデル (LM)

P(“ 今日と明日”)= 高　　P(“ 大きな鯨の卵”)= 中

P(“ 明あえ羽か水戸”)= 低

- 変換モデル (TM)

P(“ きょう” |“ 京”) = 1　　P(“ きょう” |“ 今日”) = 0.7

P(“ きょう” |“ ラーメン”) = 0

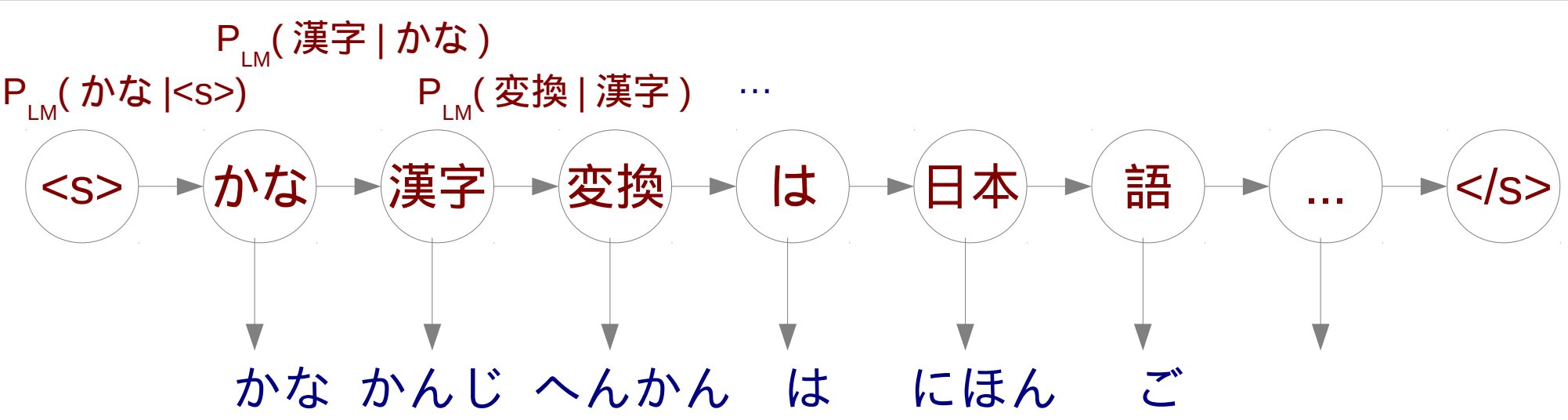
(並べ替えモデルは不要)

かな漢字変換の系列モデル

- 漢字→漢字の言語モデル確率
  - 2-gram モデル
- 漢字→かなの変換モデル確率

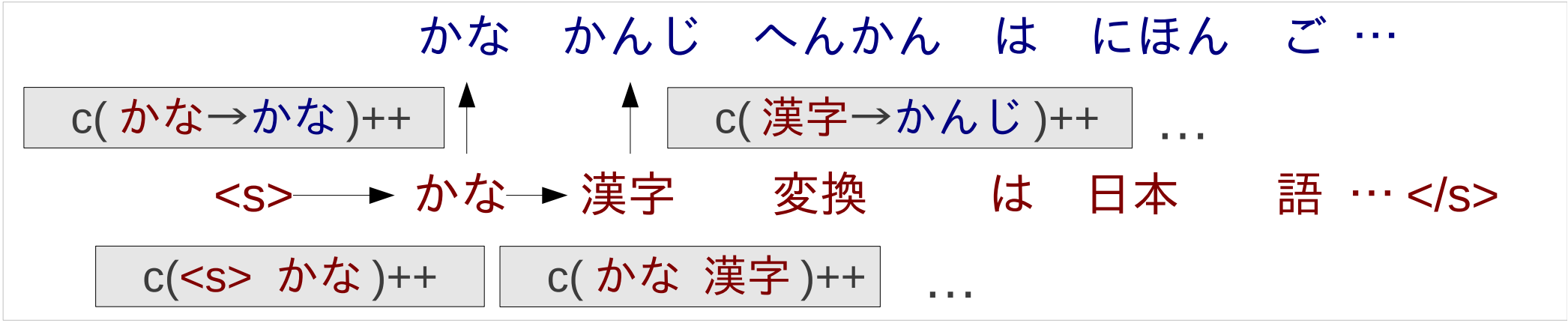
$$P(E) \approx \prod_{i=1}^{I+1} P_{LM}(e_i|e_{i-1})$$

$$P(F|E) \approx \prod_{1}^{I} P_{TM}(f_i|e_i)$$



変換モデル・言語モデルの学習

- 単語分割・発音付与されたコーパスから



- 文脈の頻度で割ることで確率を求める
  （言語モデルの場合は平滑化も利用）

$$P_T(\text{漢字} \mid \text{かな}) = c(\text{かな 漢字})/c(\text{かな}) = 1/3$$

$$P_E(\text{にっぽん} \mid \text{日本}) = c(\text{日本} \rightarrow \text{にっぽん})/c(\text{日本}) = 1/3$$

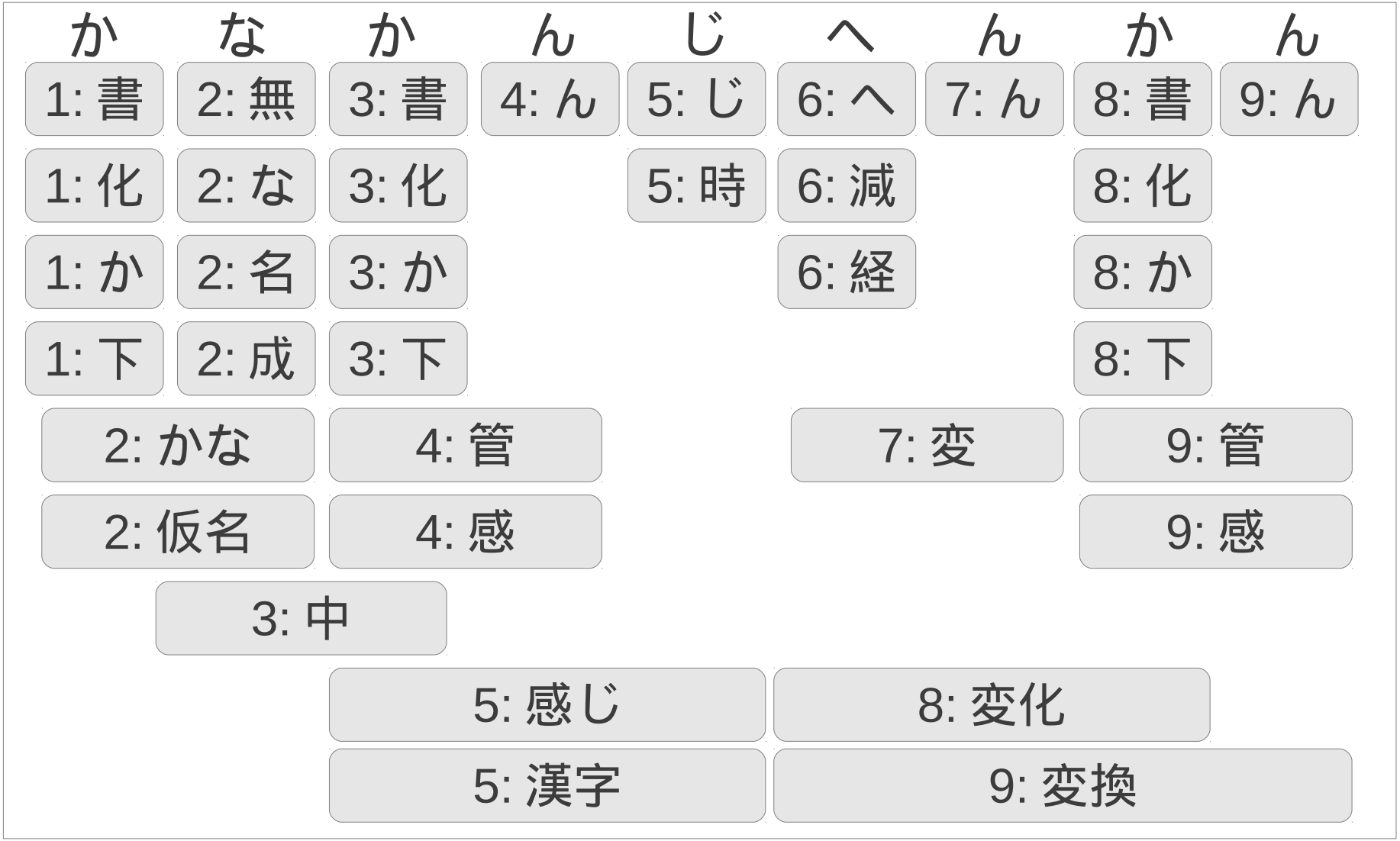
かな漢字変換候補の選択

- 良い候補と悪い候補をモデルで区別する

かなかんじへんかんはにほんごにゅうりょくのいちぶ

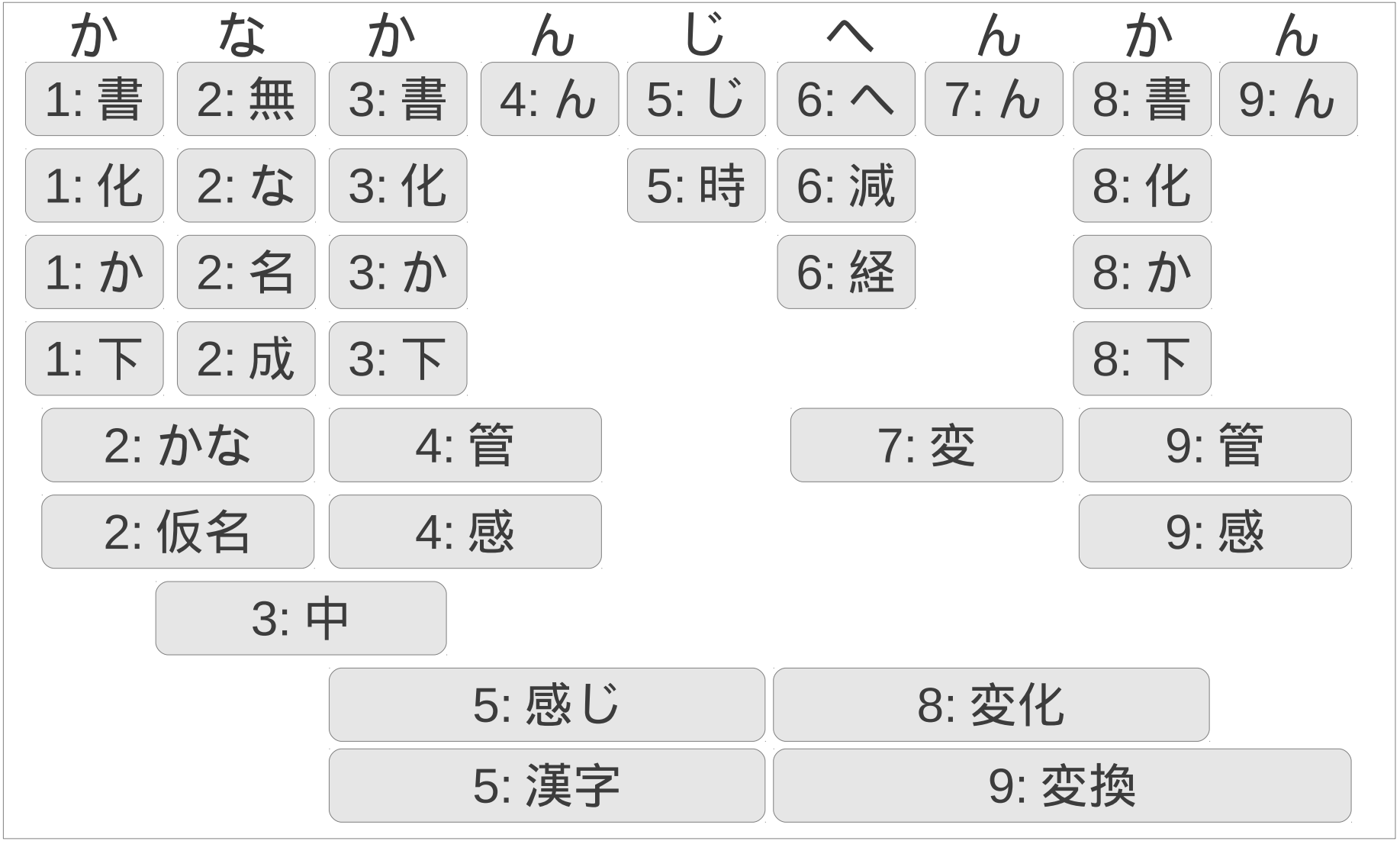
	P(E)	P(F|E)	P(E)*P(F|E)
かな漢字変換は日本語入力の一部	$4.02*10^{-3}$	$1.25*10^{-1}$	最大！
仮名漢字変換は日本語入力の一部	$2.04*10^{-3}$	$1.25*10^{-1}$	
かな漢字変換は二本後入力の一部	$8.34*10^{-5}$	$4.12*10^{-2}$	
家中ん事変感歯にﾎﾝ御乳力の胃治舞	$1.18*10^{-10}$	$2.18*10^{-7}$	
…			

組み合わせ爆発で全候補の列挙は不可能



仮説の数は？

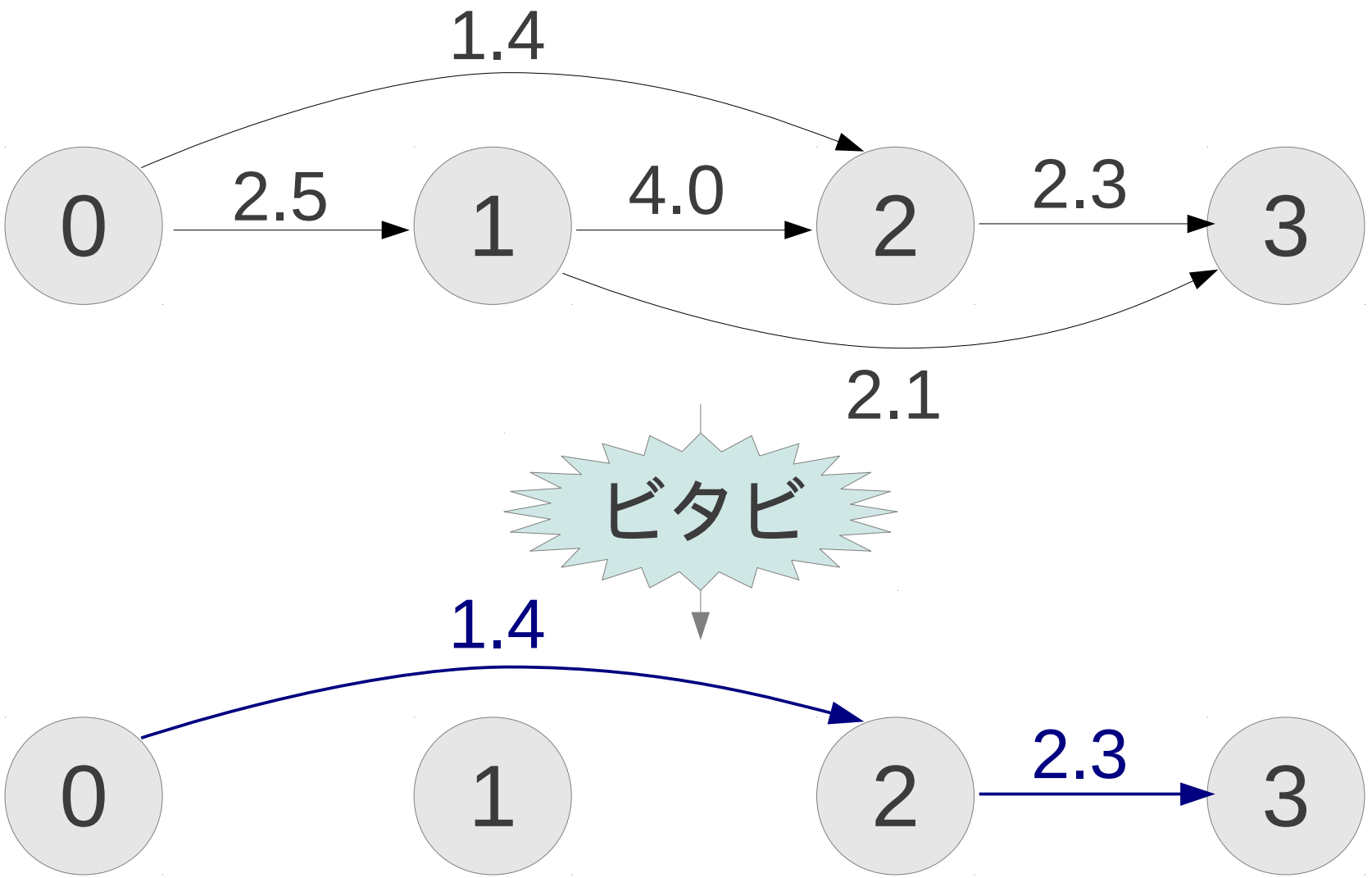
組み合わせ爆発で全候補の列挙は不可能



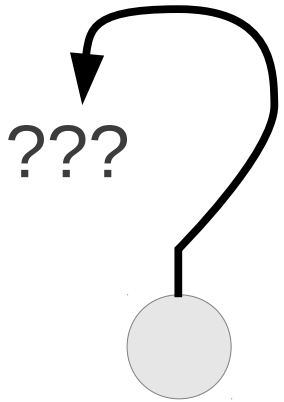
仮説の数は？ か :4 かな :18 かなか :76 かなかん :112 …64
かなかんじへんかん : 5720

効率的な探索法：ビタビアルゴリズム

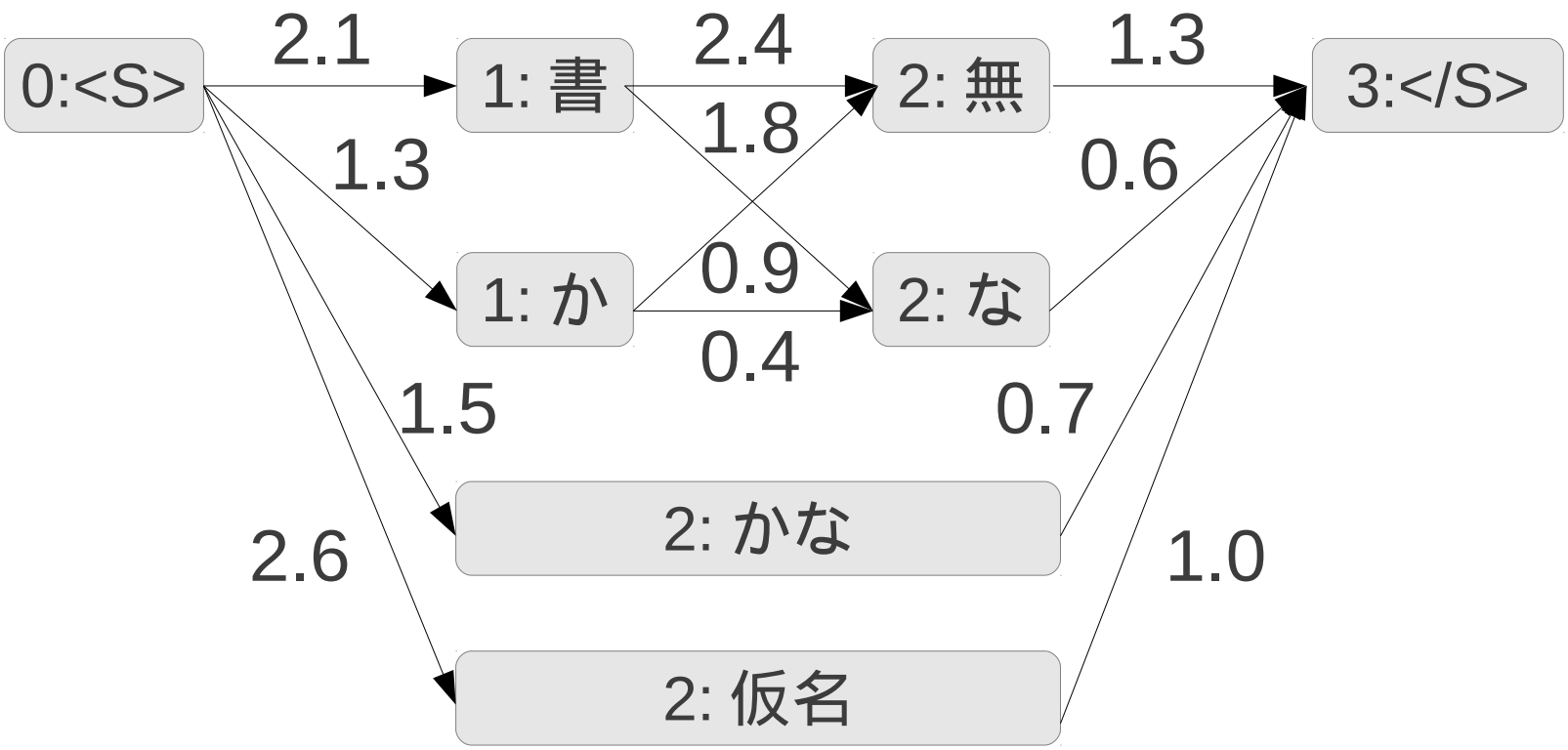
- グラフの最短経路を見つけるアルゴリズム



グラフ？
単語分割の話じゃなかったっけ？



グラフとしてのかな漢字変換

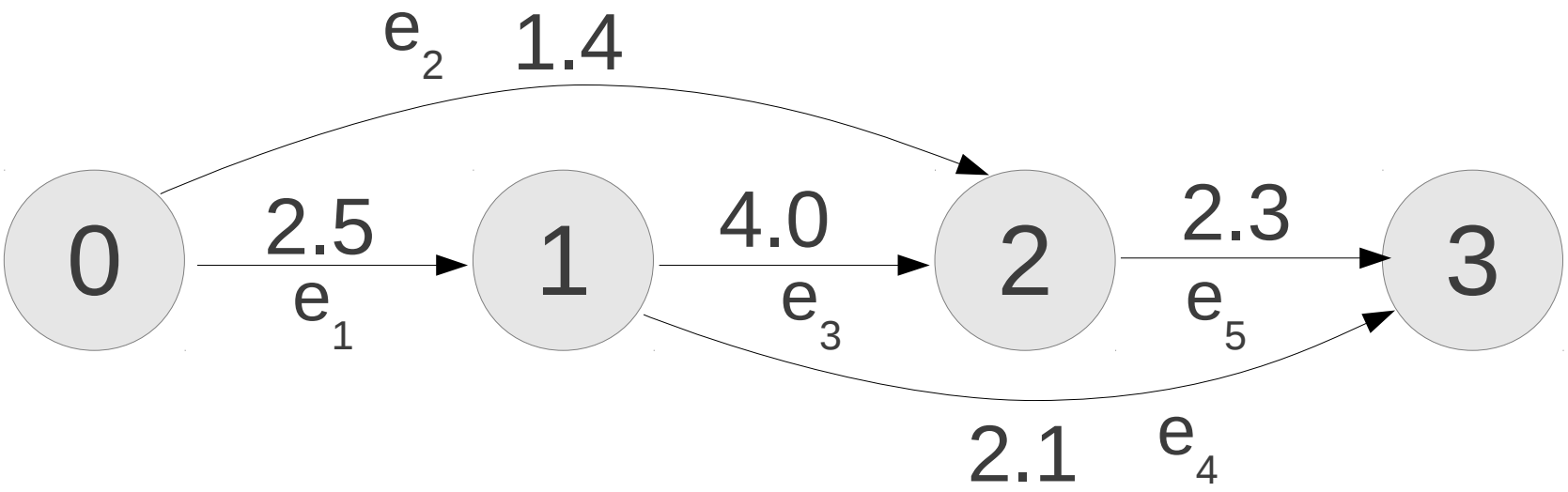


- 各エッジは単語を表す
- エッジの重みは全てのモデルを考慮した負の対数確率
  - なぜ負？（ヒント：最短経路）

ビタビアルゴリズムの探索過程

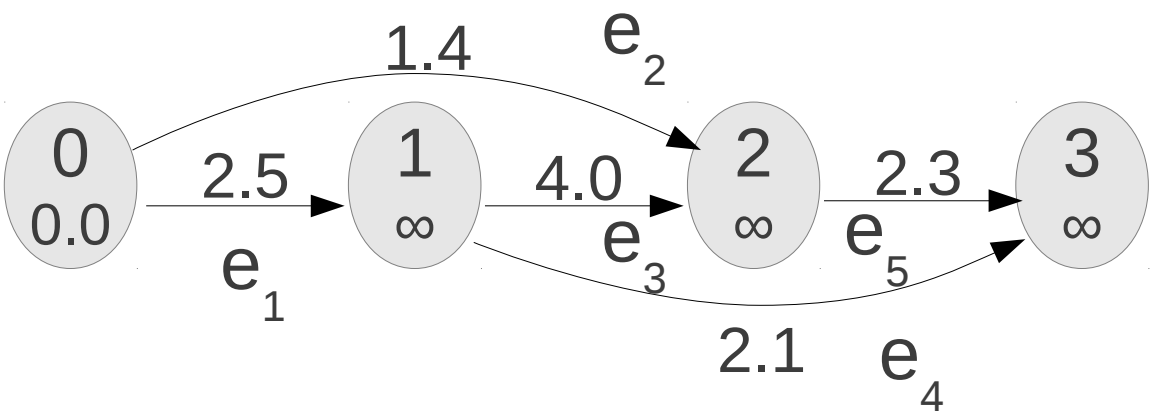
- 前向きステップで、各ノードへたどる最短経路の長さを計算
- 後ろ向きステップで、最短経路自体を構築

前向きステップ



best_score[0] = 0
for each *node* in the *graph* (昇順)
    best_score[*node*] = ∞
    for each incoming *edge* of *node*
        score = *best_score*[*edge.prev_node*] + *edge.score*
        if *score* < *best_score*[*node*]
            best_score[*node*] = *score*
            best_edge[*node*] = *edge*

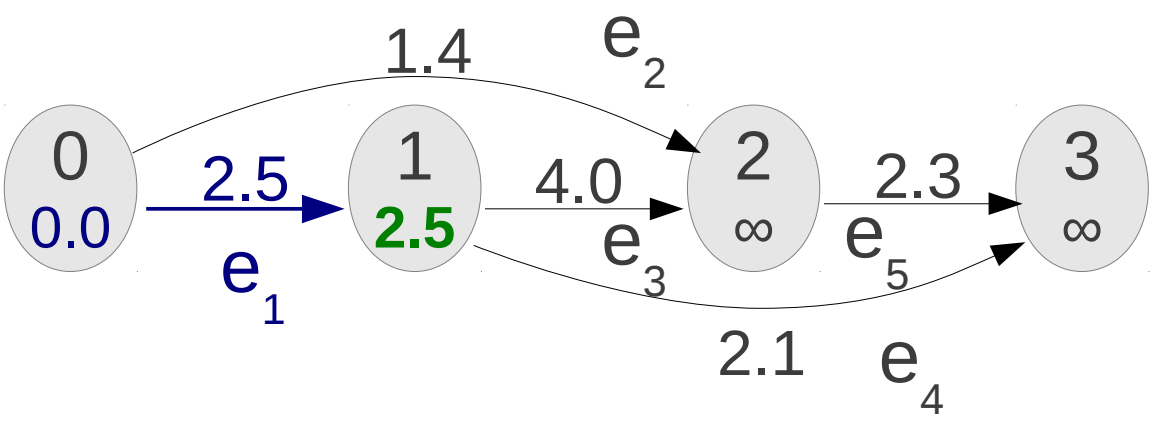
例：



初期化：

best_score[0] = 0

例：



初期化：

best_score[0] = 0

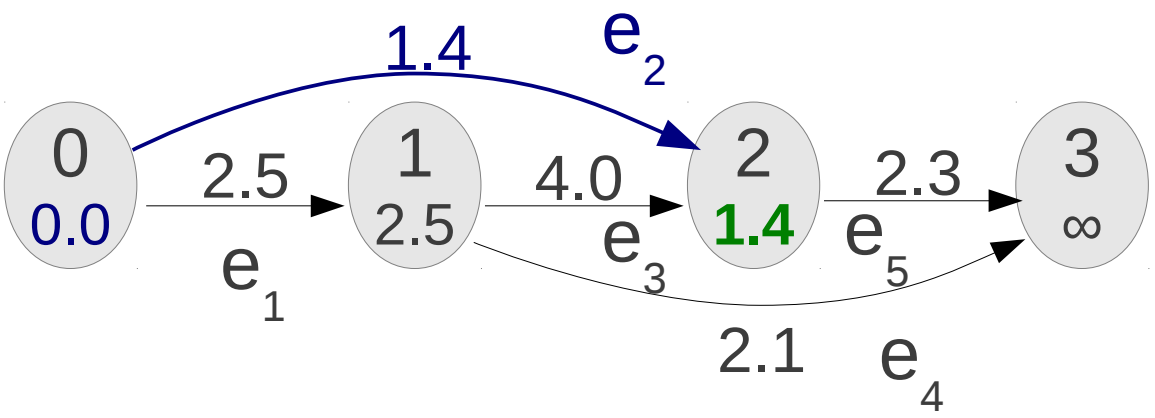
e_1 を計算：

score = 0 + 2.5 = 2.5 (< ∞)

best_score[1] = 2.5

best_edge[1] = e_1

例：



初期化：

best_score[0] = 0

e_1 を計算：

score = 0 + 2.5 = 2.5 (< ∞)

best_score[1] = 2.5

best_edge[1] = e_1

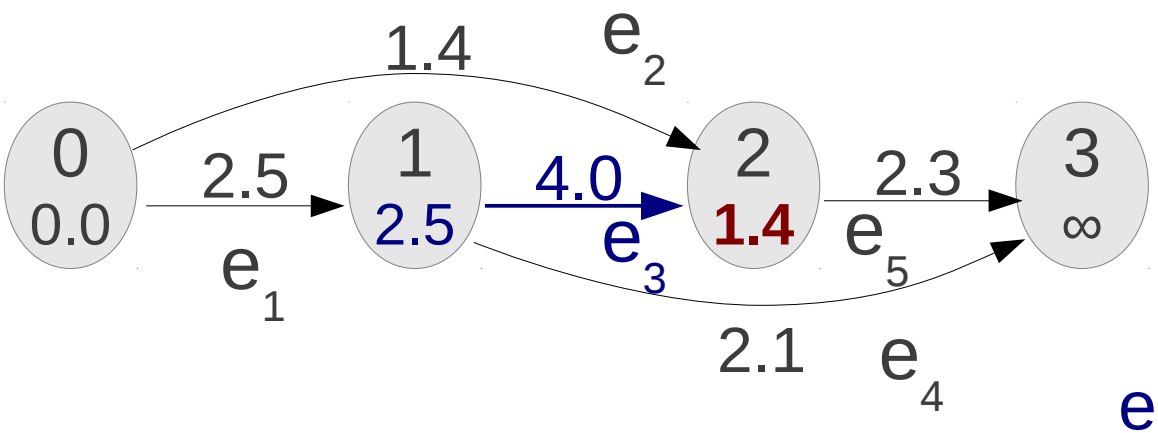
e_2 を計算：

score = 0 + 1.4 = 1.4 (< ∞)

best_score[2] = 1.4

best_edge[2] = e_2

例 :



初期化 :

best_score[0] = 0

e_1 を計算 :

score = 0 + 2.5 = 2.5 (< ∞)

best_score[1] = 2.5

best_edge[1] = e_1

e_2 を計算 :

score = 0 + 1.4 = 1.4 (< ∞)

best_score[2] = 1.4

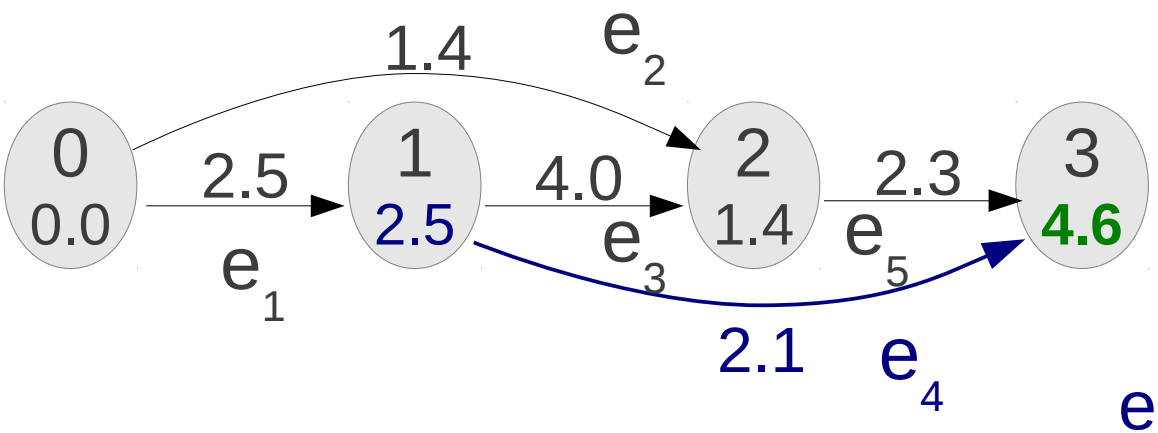
best_edge[2] = e_2

e_3 を計算 :

score = 2.5 + 4.0 = 6.5 (> 1.4)

変更なし !

例 :



初期化 :

best_score[0] = 0

e_1 を計算 :

score = 0 + 2.5 = 2.5 (< ∞)

best_score[1] = 2.5

best_edge[1] = e_1

e_2 を計算 :

score = 0 + 1.4 = 1.4 (< ∞)

best_score[2] = 1.4

best_edge[2] = e_2

e_3 を計算 :

score = 2.5 + 4.0 = 6.5 (> 1.4)

変更なし !

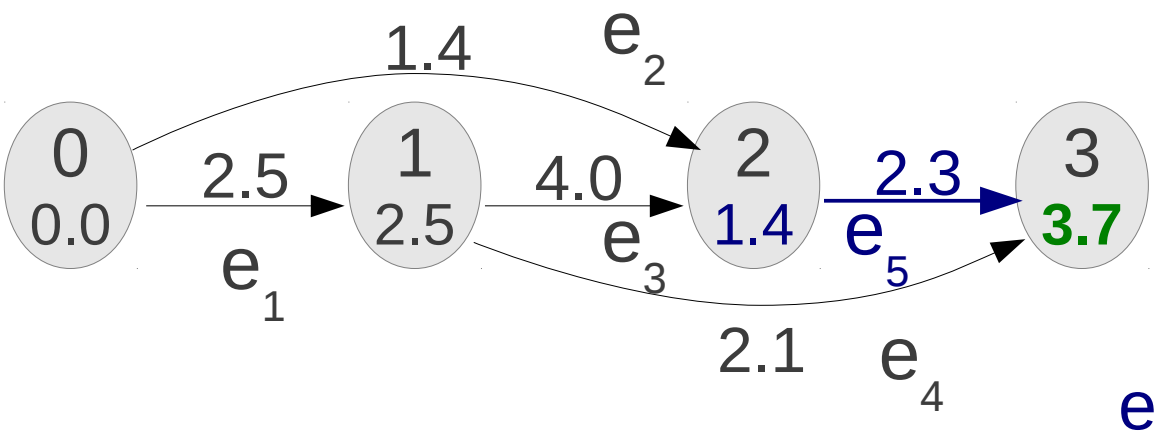
e_4 を計算 :

score = 2.5 + 2.1 = 4.6 (< ∞)

best_score[3] = 4.6

best_edge[3] = e_4

例 :



初期化 :
best_score[0] = 0

e_1 を計算 :
score = 0 + 2.5 = 2.5 (< ∞)
best_score[1] = 2.5
best_edge[1] = e_1

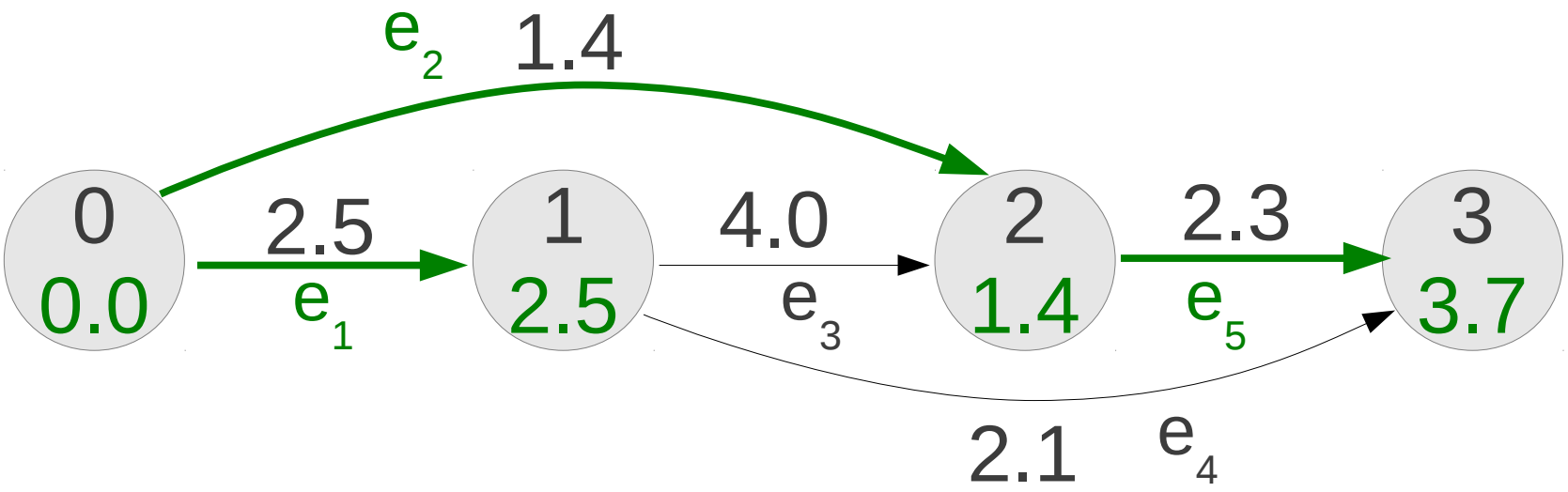
e_2 を計算 :
score = 0 + 1.4 = 1.4 (< ∞)
best_score[2] = 1.4
best_edge[2] = e_2

e_3 を計算 :
score = 2.5 + 4.0 = 6.5 (> 1.4)
変更なし !

e_4 を計算 :
score = 2.5 + 2.1 = 4.6 (< ∞)
~~best_score[3] = 4.6~~
~~best_edge[3] = e_4~~

e_5 を計算 :
score = 1.4 + 2.3 = 3.7 (< 4.6)
best_score[3] = 3.7
best_edge[3] = e_5

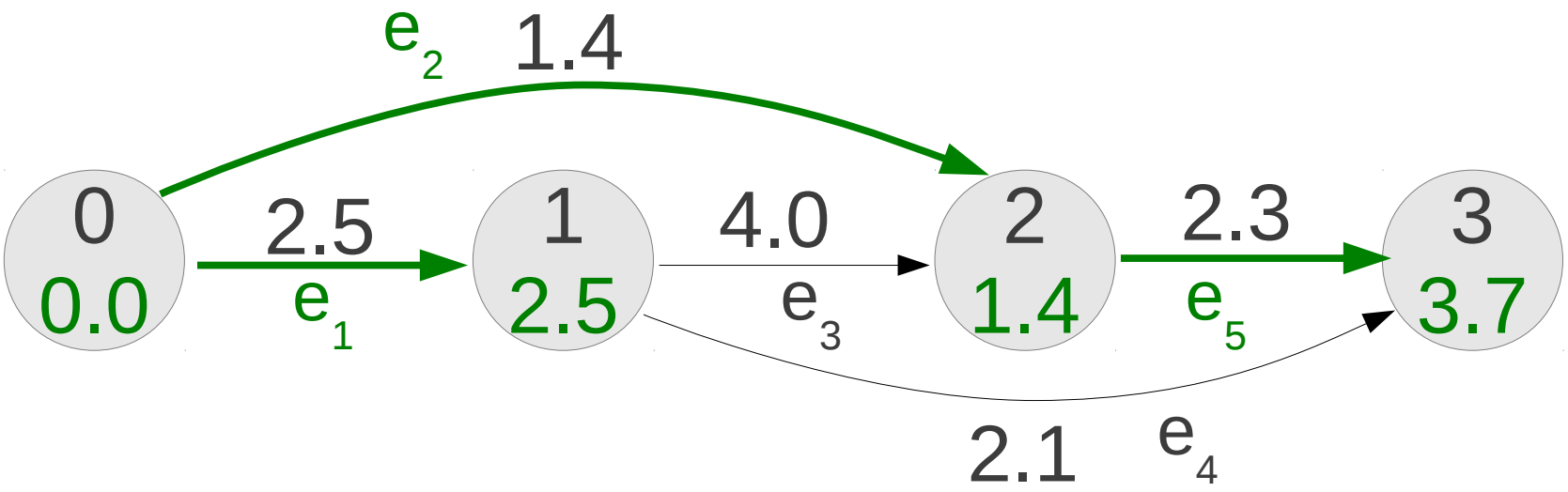
前向きステップの結果：



best_score = (0.0, 2.5, 1.4, 3.7)

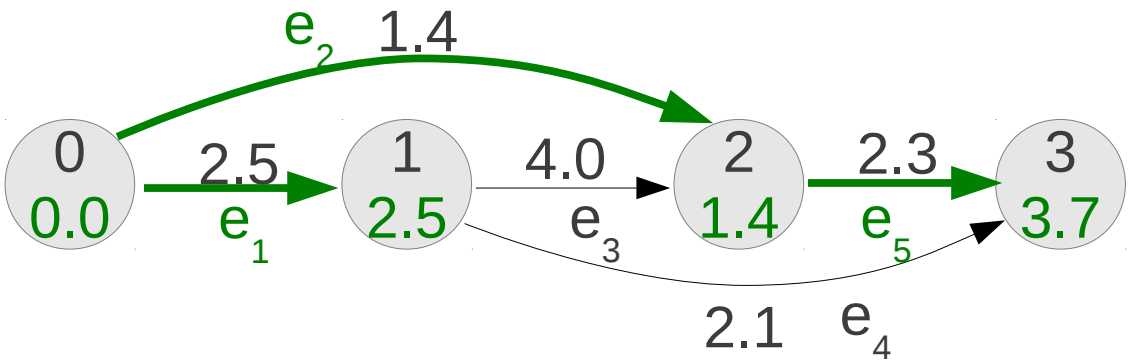
best_edge = (NULL, e_1, e_2, e_5)

後ろ向きステップのアルゴリズム



best_path = [ ]
next_edge = *best_edge*[*best_edge*.length – 1]
while *next_edge* != NULL
    add *next_edge* to *best_path*
    next_edge = *best_edge*[*next_edge.prev_node*]
reverse best_path

後ろ向きステップの例

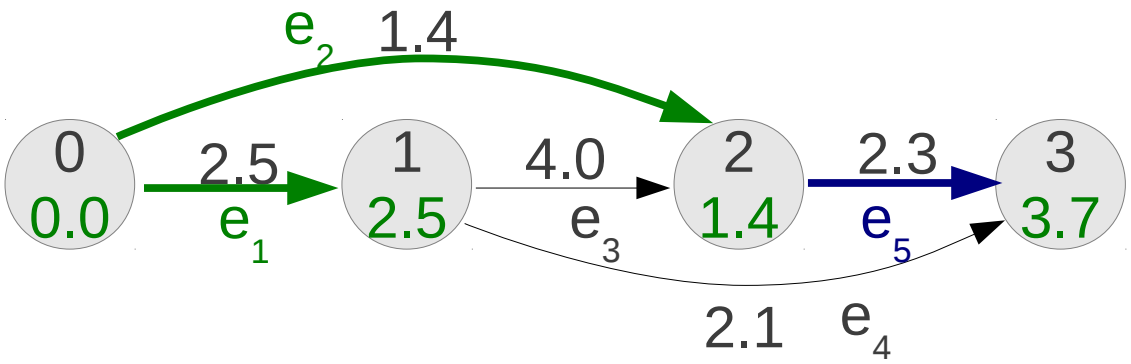


初期化 :

best_path = []

next_edge = best_edge[3] = e_5

後ろ向きステップの例



初期化 :

best_path = []

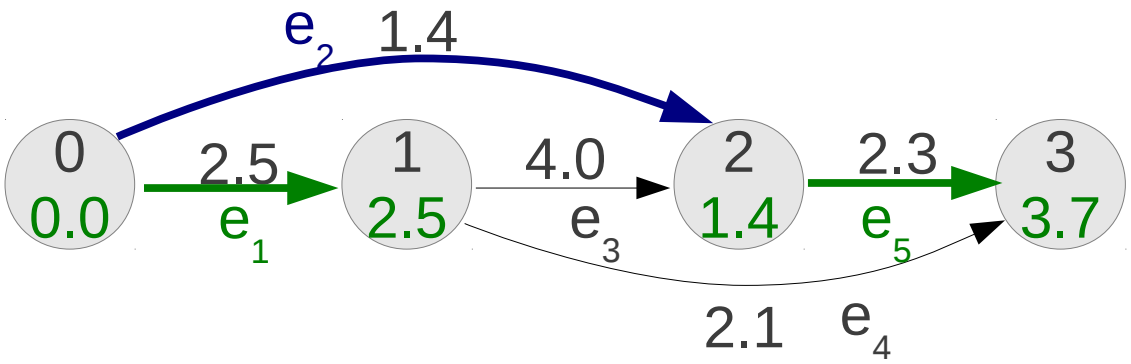
next_edge = best_edge[3] = e_5

e_5 を計算 :

best_path = [e_5]

next_edge = best_edge[2] = e_2

後ろ向きステップの例



初期化 :

best_path = []

next_edge = best_edge[3] = e_5

e_5 を計算 :

best_path = [e_5]

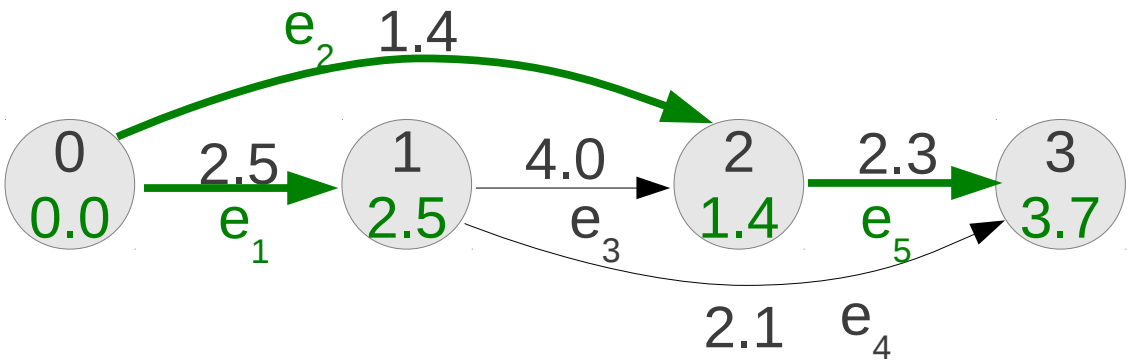
next_edge = best_edge[2] = e_2

e_2 を計算 :

best_path = [e_5, e_2]

next_edge = best_edge[0] = NULL

後ろ向きステップの例



初期化 :

best_path = []

next_edge = best_edge[3] = e_5

e_5 を計算 :

best_path = [e_5]

next_edge = best_edge[2] = e_2

e_5 を計算 :

best_path = [e_5, e_2]

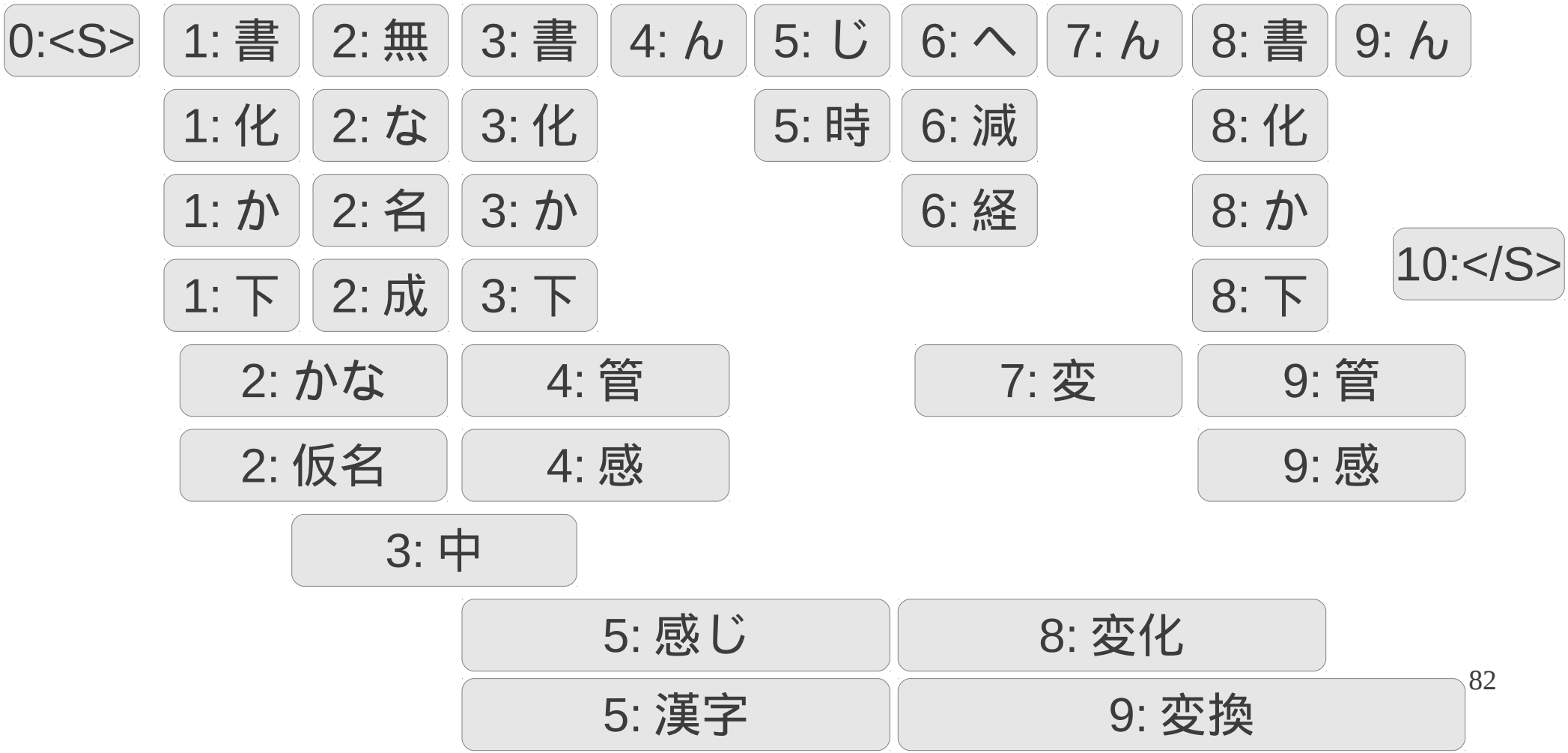
next_edge = best_edge[0] = NULL

逆順に並べ替え :

best_path = [e_2, e_5]

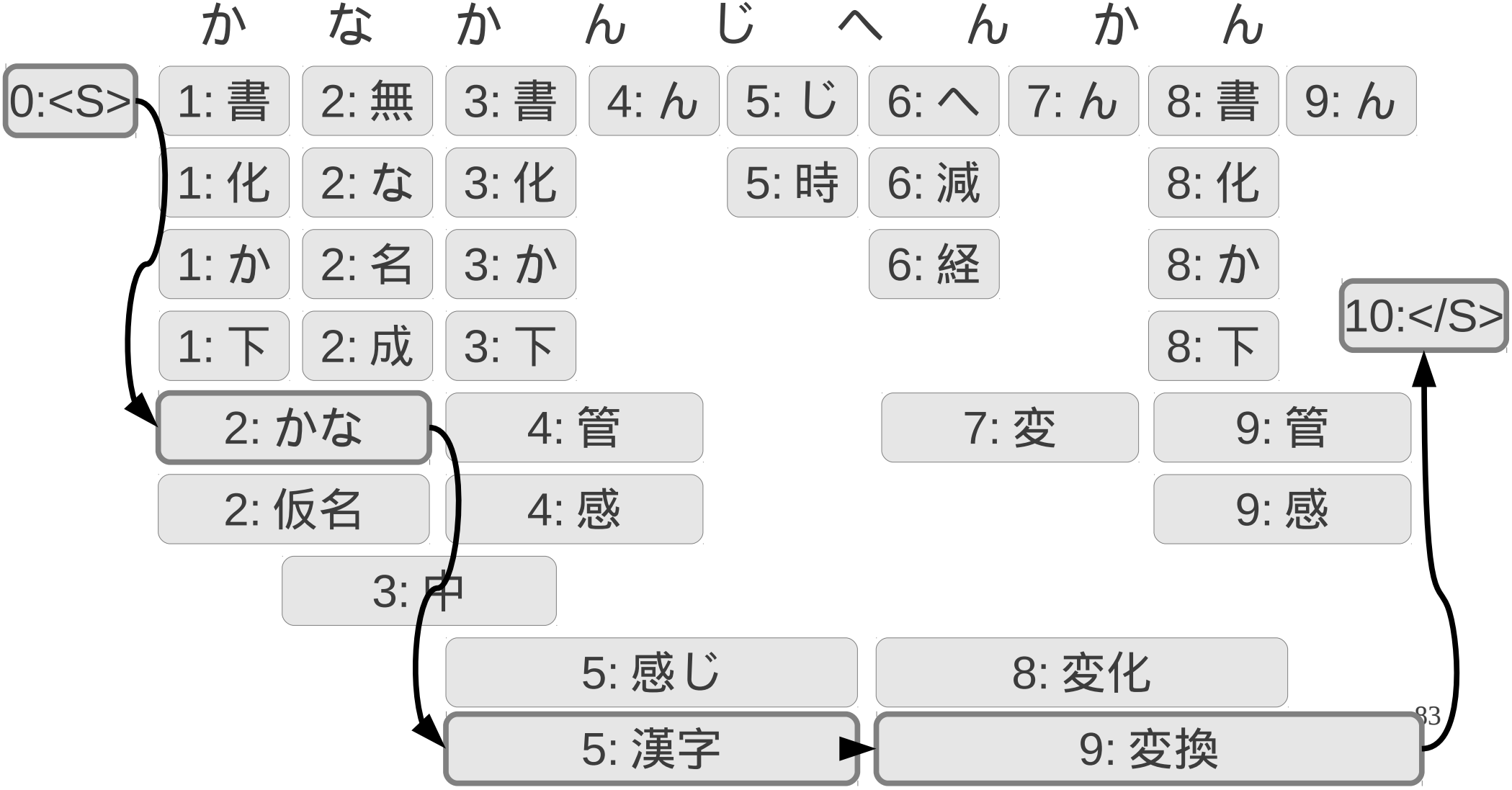
かな漢字変換の探索

- ビタビアルゴリズムを利用



かな漢字変換の探索

- ビタビアルゴリズムを利用



かな漢字変換の探索

- 0:<S> で探索開始

か　な　か　ん　じ　へ　ん　か　ん

0:<S>　S[“0:<S>”] = 0

かな漢字変換の探索

- 0→1 のスパンをまたがる単語を全て展開

か　な　か　ん　じ　へ　ん　か　ん

0:<S>

1: 書　S[“1: 書”] = -log (P_{TM}(か | 書) * P_{LM}(書 |<S>)) + S[“0:<S>”]

1: 化　S[“1: 化”] = -log (P_{TM}(か | 化) * P_{LM}(化 |<S>)) + S[“0:<S>”]

1: か　S[“1: か”] = -log (P_{TM}(か | か) * P_{LM}(か |<S>)) + S[“0:<S>”]

1: 下　S[“1: 下”] = -log (P_{TM}(か | 下) * P_{LM}(下 |<S>)) + S[“0:<S>”]

かな漢字変換の探索

- 0→2 のスパンをまたがる単語を全て展開

か な か ん じ へ ん か ん

0:<S>

1: 書

1: 化

1: か

1: 下

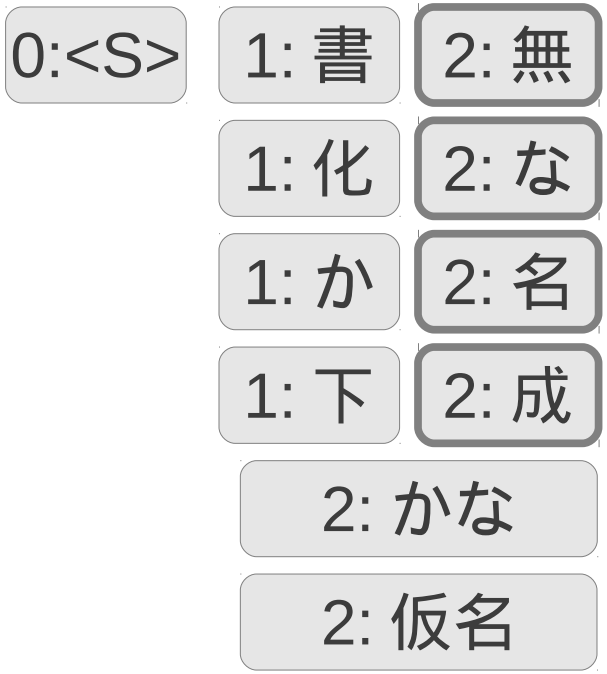
2: かな　S[1: かな] = -log (P_{TM}(かな | かな) * P_{LM}(かな |<S>)) + S[“0:<S>”]

2: 仮名　S[1: 仮名] = -log (P_{TM}(かな | 仮名) * P_{LM}(仮名 |<S>)) + S[“0:<S>”]

かな漢字変換の探索

- 1→2 のスパンをまたがる単語を全て展開

か　な　か　ん　じ　へ　ん　か　ん



S[“2: 無”] = min(
-log (P_{TM}(な | 無) * P_{LM}(無 | 書)) + S[“1: 書”],
-log (P_{TM}(な | 無) * P_{LM}(無 | 化)) + S[“1: 化”],
-log (P_{TM}(な | 無) * P_{LM}(無 | か)) + S[“1: か”],
-log (P_{TM} な | 無) * P_{LM}(無 | 下)) + S[“1: 下”])

S[“2: な”] = min(
-log (P_{TM}(な | な) * P_{LM}(な | 書)) + S[“1: 書”],
-log (P_{TM}(な | な) * P_{LM}(な | 化)) + S[“1: 化”],
-log (P_{TM}(な | な) * P_{LM}(な | か)) + S[“1: か”],
-log (P_{TM}(な | な) * P_{LM}(な | 下)) + S[“1: 下”])

■ ■ ■

かな漢字変換のまとめ

- 仮説の良し悪しを確率モデルで判断
- 変換モデルと言語モデル
- 各モデルをコーパスから学習
- ビタビアルゴリズムで確率の高い解を探索

演習：かな漢字変換の学習 (1)

- 1: 読み付与
- 2: 言語モデル学習
- 3: 変換モデル学習
- 4: 変換の探索

演習：かな漢字変換の学習 (2)

- まずかな漢字変換用のディレクトリを作成

```
$ mkdir kkc
```

- KyTea という形態素解析器で単語分割・読み付与
  (手軽に学習できるよう、 20000 文のみを利用)

```
$ head -20000 data/orig/kyoto-train.ja |
    usr/bin/kytea -notag 1 -tagbound "|" > kkc/kyoto-train.wordpron
```

品詞不要　　単語と読みを「|」で分割

- もう一度単語分割のみを行う

```
$ head -20000 data/orig/kyoto-train.ja |
    usr/bin/kytea -notags > kkc/kyoto-train.word
```

(sed で読みを消すこともできる）

演習：かな漢字変換の学習 (3)

- 言語モデルと変換モデルを構築

```
$ usr/local/kkc/train-bigram.py kkc/kyoto-train.word > kkc/kyoto-train.lm
$ usr/local/kkc/train-tm.py kkc/kyoto-train.wordpron > kkc/kyoto-train.tm
```

- 変換モデルで「人」の発音を閲覧

```
$ grep "^人 " kkc/kyoto-train.tm
人 ひと 0.493261
人 にん 0.444744
人 じん 0.061995
```

演習：かな漢字変換の学習 (4)

- テストに使うひらがなファイルを作成

```
$ echo " きよみずでらはきょうとにあります。 " > kkc/test.txt
```

- かな漢字変換を実行

```
$ usr/local/kkc/kkc.py kkc/kyoto-train.lm kkc/kyoto-train.tm kkc/test.txt
清水 寺 は 京都 に あ り ま す 。
```

- kkc/kyoto-train.tm の確率を変更し、結果を変えてみる

更に勉強するには

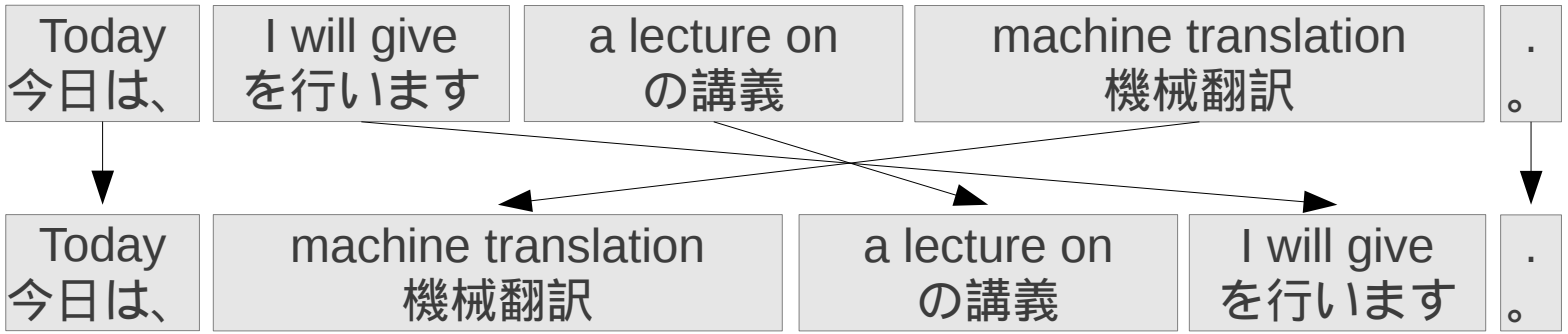
- 自然言語処理プログラミングチュートリアル (6 回目)
  http://www.phontron.com/teaching.php

かな漢字変換
↓
フレーズベース機械翻訳

フレーズベース機械翻訳と かな漢字変換

フレーズベース機械翻訳

Today I will give a lecture on machine translation .



今日は、機械翻訳の講義を行います。

かな漢字変換

きょうは、きかいほんやくのこうぎをおこないます。

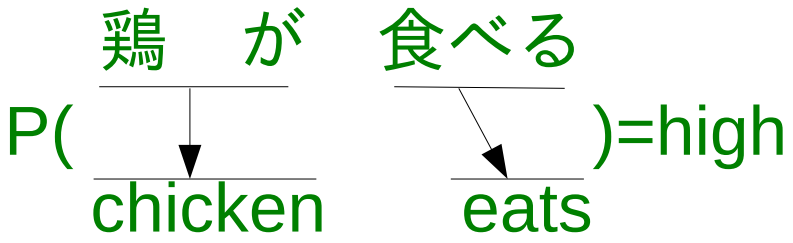
今日は、機械翻訳の講義を行います。

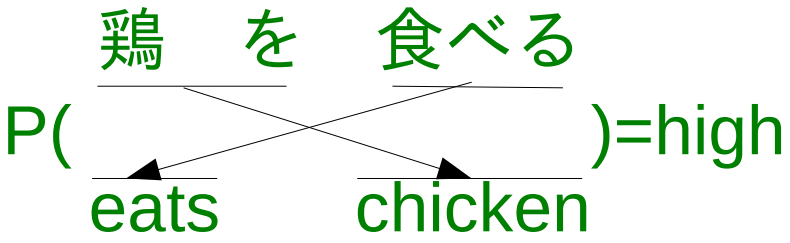
機械翻訳 (MT) と かな漢字変換 (KKC) の差

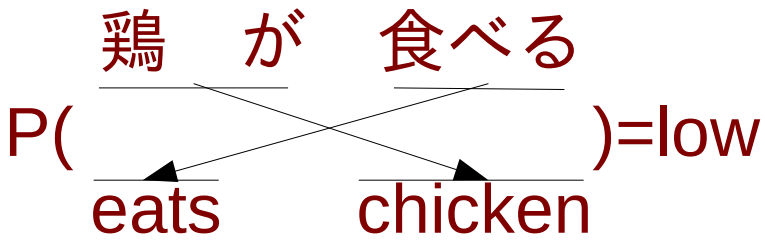
- 並べ替えモデル
  KKC: 必要なし、 MT: 重要
- 探索アルゴリズム
  KKC: 単純なビタビ、 MT: 並べ替えを考慮し、 複雑
- 学習データ
  KKC: 読み付きデータ、 MT: 対訳データ
- 学習アルゴリズム
  KKC: 最尤推定、 MT: いろいろな工夫が必要

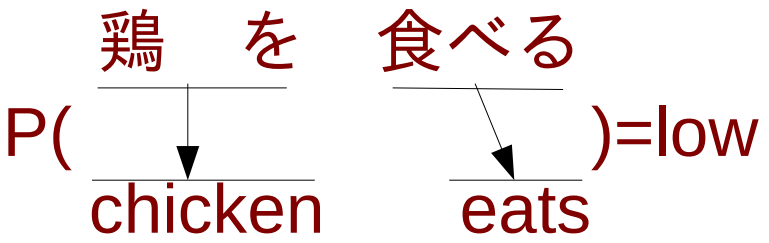
並べ替えモデル

- 目的言語文が正しい語順になるように確率を付与



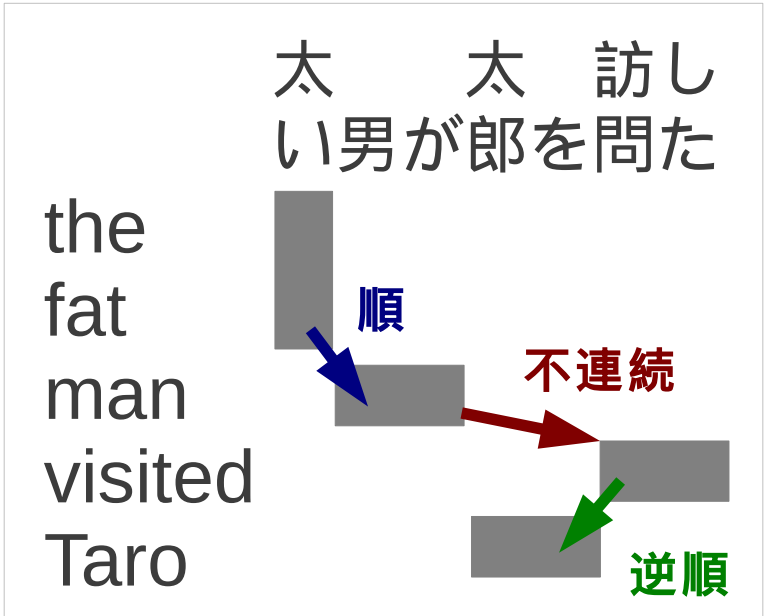






語彙化並べ替えモデル [Koehn+ 05]

- 順・逆順・不連続



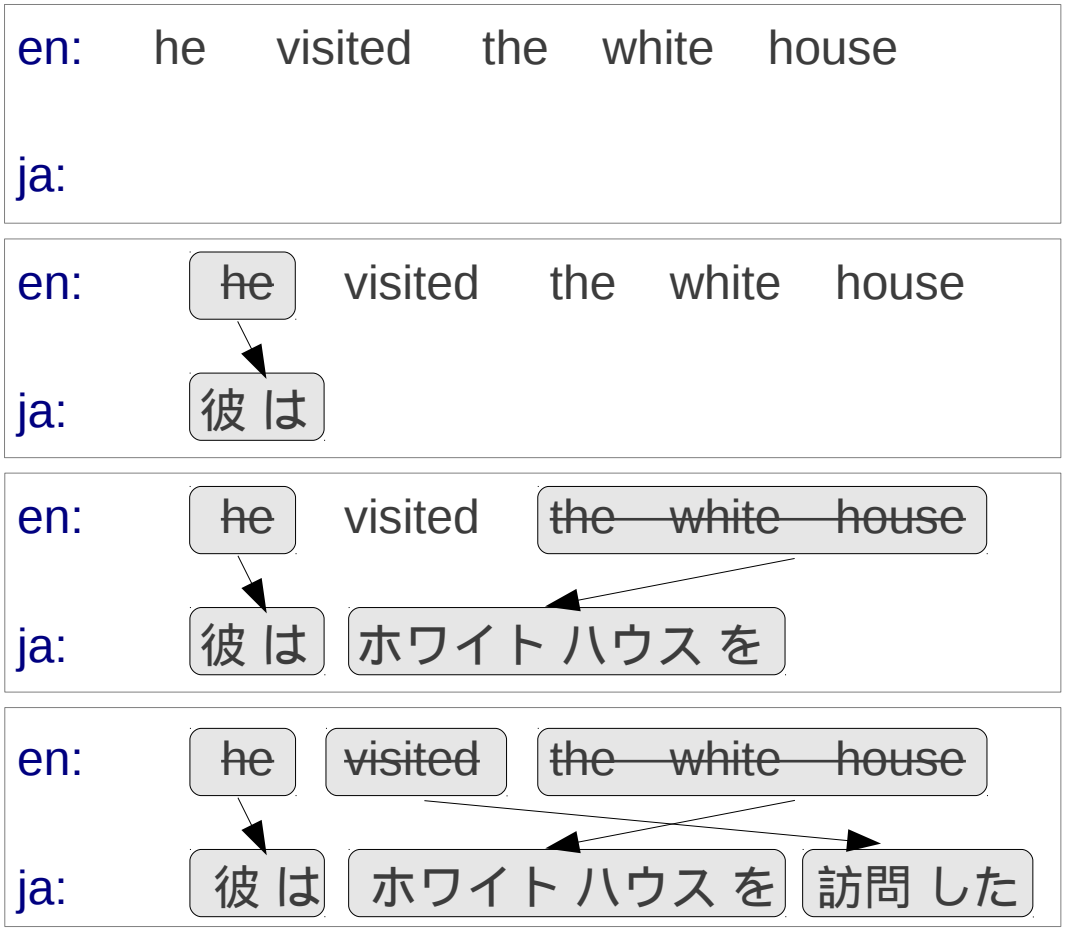
太い → the fat
順の確率が高い

太郎 を → Taro
逆順の確率が高い

- 入力・出力、右・左などで条件付けた確率

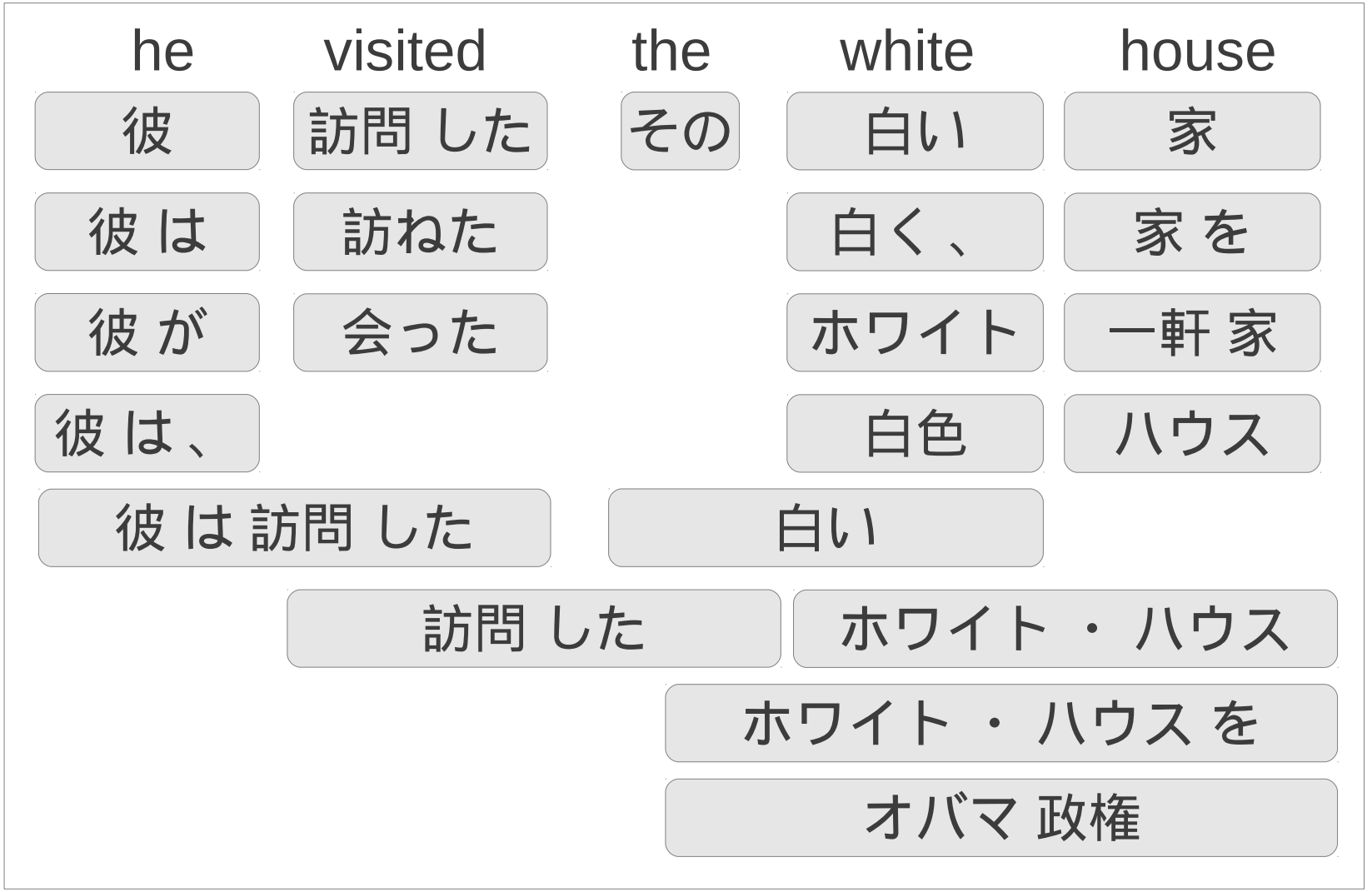
フレーズベース機械翻訳の探索（デコーディング, [Koehn+ 03]）

- 目的言語文を文頭から構築
- 翻訳された原言語単語を記憶
- スコアの最も高い候補を出力
- ビタビアルゴリズムの拡張であるビーム探索を利用



デコーディングのアルゴリズム概要

- まずかな漢字変換と同じように、翻訳候補を列挙



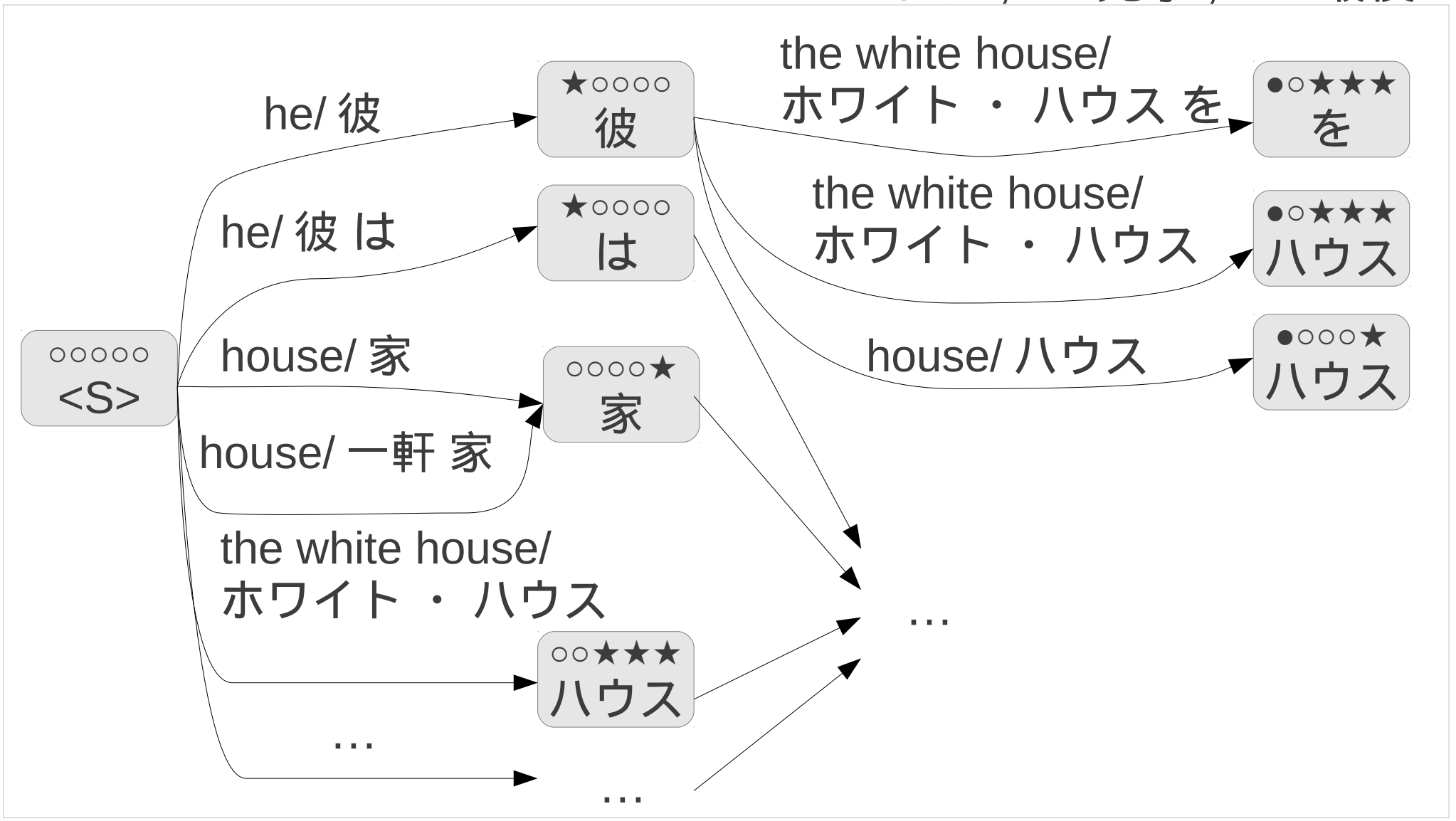
ビタビアルゴリズムの展開

- かな漢字変換と同じように、候補をグラフとして表現
- 目的言語文の文頭から展開
  - かな漢字変換と異なり、原言語は文頭からと制約せず
- 各ノードのラベル：
  - 翻訳し終わった原言語単語
  - 最後に翻訳した単語（並べ替えモデルのため）
  - 目的言語側の最後お n-1 単語（言語モデルのため）

ビタビグラフの例

E=he visited the white house

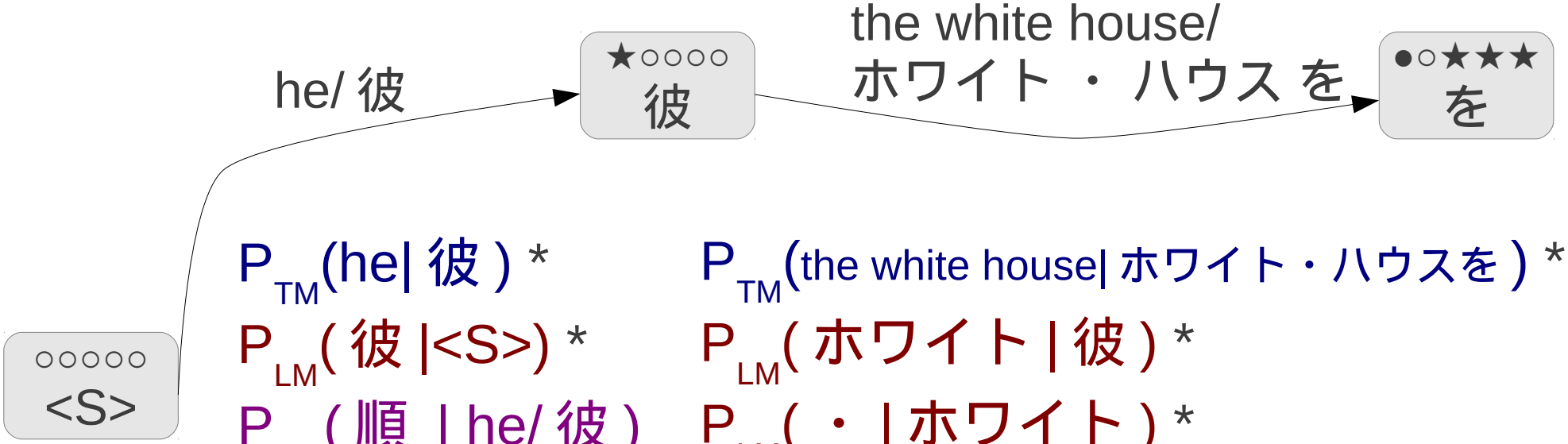
言語モデル n=2
○= まだ , ●= 完了 , ★= 最後



グラフに対するスコア付け

E=he visited the white house

言語モデル n=2
○= まだ , ●= 完了 , ★= 最後



P_{TM}(he| 彼) *
P_{LM}(彼 |<S>) *
P_{RM}(順 | he/ 彼)

P_{TM}(the white house| ホワイト・ハウスを) *
P_{LM}(ホワイト | 彼) *
P_{LM}(・ | ホワイト) *
P_{LM}(ハウス | ・) *
P_{LM}(を | ハウス) *
P_{RM}(不連続 |
the white house| ホワイト・ハウスを)

問題：グラフが膨大

- 単語の並べ替えだけを考えても O(n!)

問題：グラフが膨大

- 単語の並べ替えだけを考えても O(n!)
- 解決策：
  - 並べ替えの制限を設ける
  - ビーム探索を導入

並べ替え制限

- 探索中 n 単語以上を翻訳せずに飛ばすことを許さない
- 例えば n=4 の場合

he visited the white house

彼 は ... → 0 単語 → OK

訪問 した ... → 1 単語 → OK

白い ... → 3 単語 → OK

家 ... → 4 単語 → NG

ビーム探索

- n 単語が翻訳された仮説の中で、スコアの高い b 子だけを更に展開 (n=1, b=3 の場合)

he visited the white house

彼 は …	1.5	展開
訪問 した …	1	展開
彼 が …	0.8	展開
白い …	0.5	破棄
彼 …	0.3	破棄

…

研究

- ラティス入力の探索 [Dyer 08]
- 統語ベース翻訳の探索 [Mi 08]
- 最小ベイズリスク [Kumar 04]
- 厳密な解の求め方 [Germann 01]

演習： Moses デコーダで翻訳 (1)

- 学習済みのモデルは model ディレクトリに
  - moses.ini (設定ファイル)
  - phrase-table.gz (翻訳モデル)
  - reordering-table.wbe-msd-bidirectional-fe.gz (並べ替えモデル)
- それぞれを確認
- 翻訳モデルは：

kyoto city , kyoto prefecture ||| 京都 府 、 京都 市 ||| … (様々な統計)

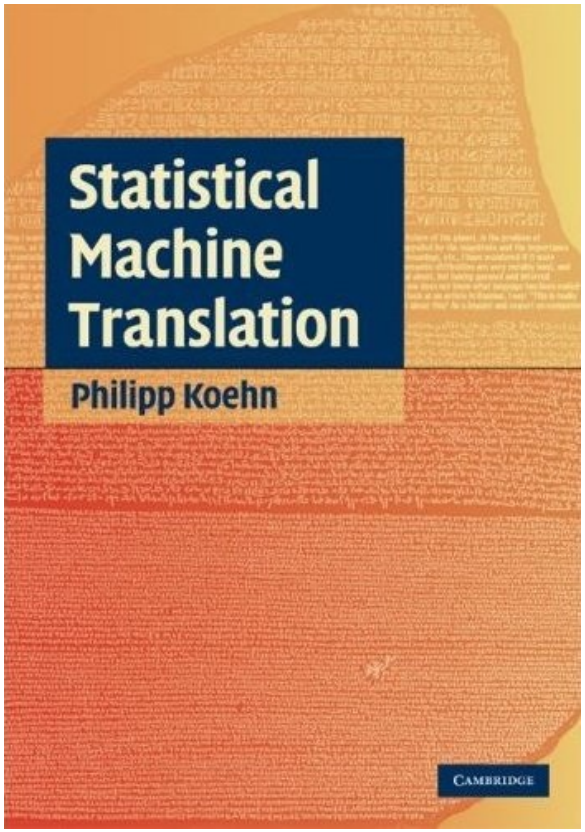
演習： Moses デコーダで翻訳 (2)

- デコーダを実行

```
$ mosesdecoder/bin/moses -config model/moses.ini
…
Created input-output object : [26.041] seconds
```

- 好きな文を英語で入力
  - 大文字を使わない
  - 句読点を単語と別で書く
  - 例えば：「 he is a student in kyoto . 」

更に勉強するには



[Chapter 6]

- 日本語の資料はまだ少ない…

フレーズベース統計的機械翻訳の学習

機械翻訳の (基本的な) 学習過程

- データの収集・作成
- 単語分割・トークン化
- 言語モデル
- 単語の対応付け
- パターン抽出・スコア付け
- 訳出（デコーディング）
- 翻訳の評価
- 重み調整（チューニング）

必要なデータ

- 対訳文データ
  - 翻訳・並べ替えモデルで利用

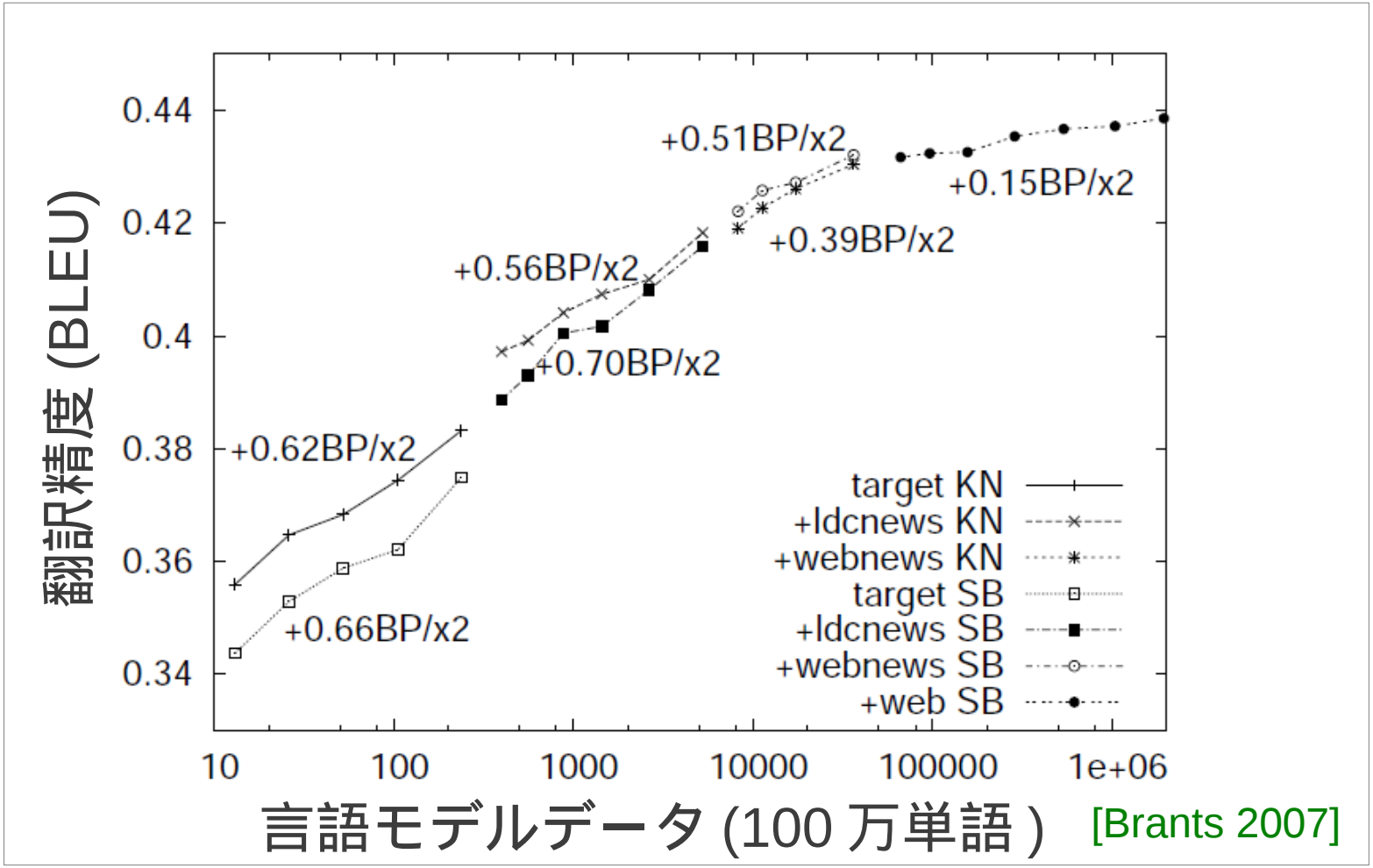
これはペンです。	This is a pen.
昨日は友達と食べた。	I ate with my friend yesterday.
象は花が長い。	Elephants' trunks are long.

- 単言語データ (目的言語側)
  - 言語モデルに利用

This is a pen.
I ate with my friend yesterday.
Elephants' trunks are long.

良いデータは

- 大きい →



[Brants 2007]

- ノイズを含まない
- 訳したいテキストと同じ分野

対訳データの収集方法

- 質の高い対訳データ：
  - 政府・自治体
  - ニュース
  - 特許
- Web から収集
- 複数のデータ源を組み合わせる

日英の対訳データ

- アイデアは？

例：日英の対訳データ

	ジャンル	入手
TED 講演	講演	無料
英辞郎	辞書	数千円
英辞郎例文	雑誌・例文	数千円
京都関連 Wikipedia	百科事典	無料
日英中基本文	例文	無料
読売新聞	新聞	契約で無料
田中コーパス	様々	無料
NTCIR PatentMT	特許	契約で無料
Wikipedia 言語リンク	辞書	無料
WWWJDIC	辞書	無料

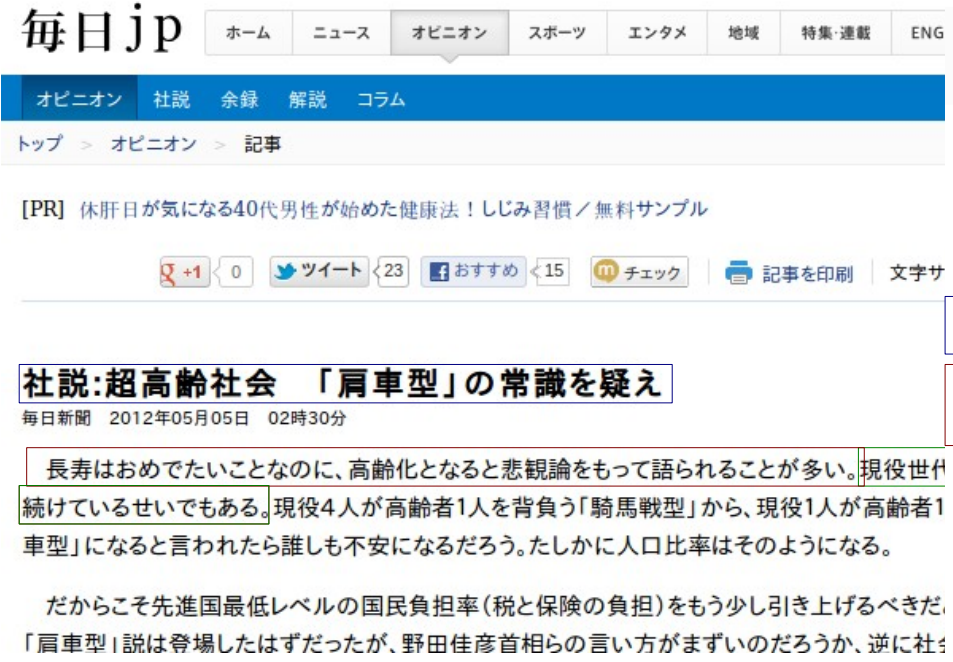
Web からのデータ収集

- 並列ページの発見 [Resnik 03]

[画像：毎日新聞]

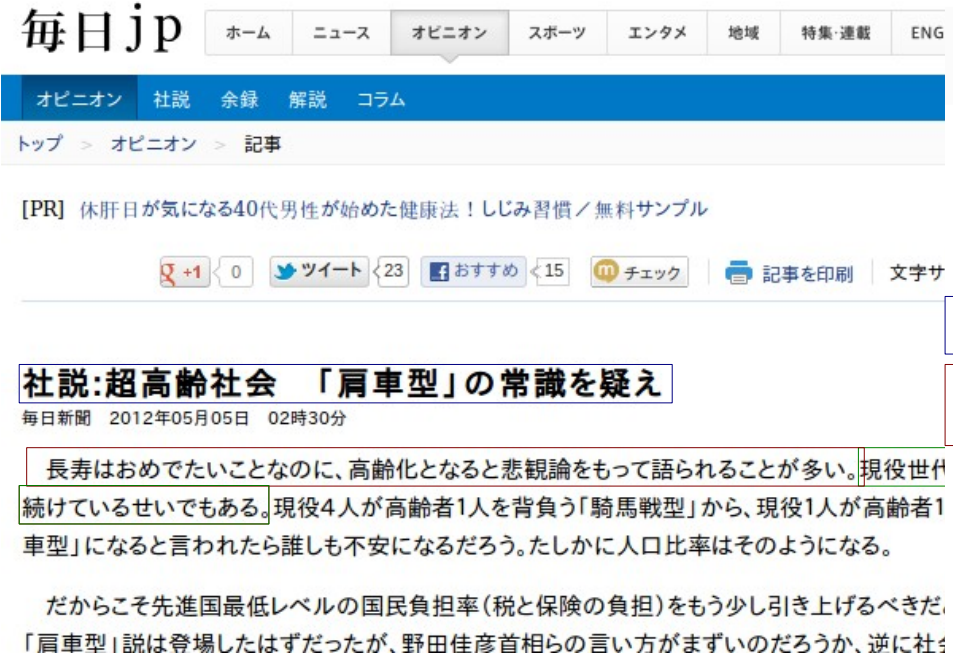
Web からのデータ収集

- 並列ページの発見 [Resnik 03]
- 文アライメント [Moore 02]



Web からのデータ収集

- 並列ページの発見 [Resnik 03]
- 文アライメント [Moore 02]



- データ作成のクラウドソーシング [Ambati 10]
  - Mechanical Turk 、 duolingo 等

トークン化

- 例：日本語の単語分割

太郎が花子を訪問した。

↓

太郎 が 花子 を 訪問 した 。

- 例：英語の小文字化、句読点の分割

Taro visited Hanako.

↓

taro visited hanako .

トークン化がとても重要

- ちょうど良い長さ：正しい翻訳

太郎 が	→	taro ○
太郎 を	→	taro ○

- 長すぎると学習データに含まれなければ翻訳不可

太郎が	→	taro ○	学習データ内
太郎を	→	太郎を ✗	学習データ外

- 短すぎると誤訳の可能性

太 郎 が	→	fat ro ✗
太 郎 を	→	fat ro ✗

トークン化ツール

- ヨーロッパの言語

```
tokenize.perl en < input.en > output.en
tokenize.perl fr < input.fr > output.fr
```

- 日本語

MeCab: `mecab -O wakati < input.ja > output.ja`
KyTea: `kytea -notags < input.ja > output.ja`
JUMAN, etc.

- 中国語

Stanford Segmenter, LDC, KyTea, etc...

研究

- 機械翻訳の精度向上につながるトークン化
  - 精度が重要か、一貫性が重要か [Chang 08]
  - 他の言語に合わせた単語挿入 [Sudoh 11]

太郎 が 花子 を 訪問 した 。

Taro <ARG1> visited <ARG2> Hanako .

- 活用の処理（韓国語、アラビア語等） [Niessen 01]

단어란 도대체 무엇일까요？

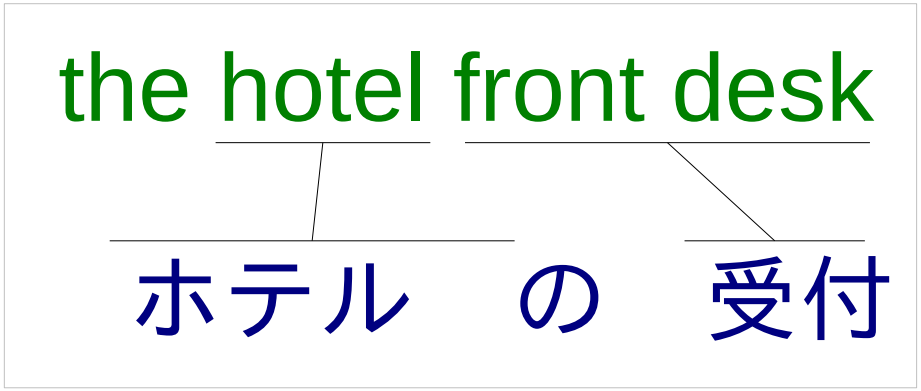
↓

단어 란 도대체 무엇 일 까요 ？

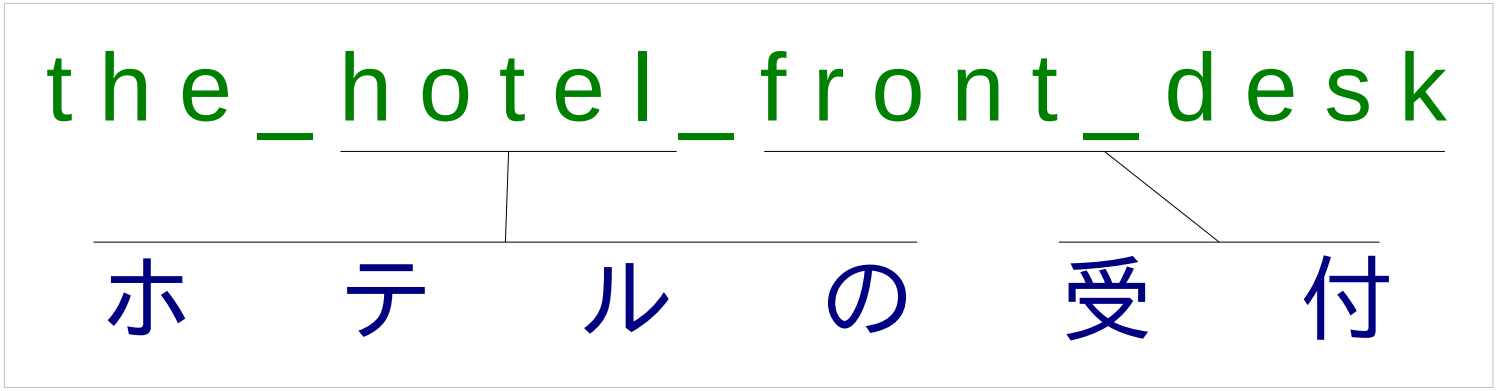
- 教師なし学習 [Chung 09, Neubig 12]

部分文字列に基づく機械翻訳 [Neubig+ 12]

- 単語に基づくモデルではなく



- 文字列に基づく機械翻訳！



- 言語資源なしで未知語に対応、精度が単語翻訳と匹敵

演習：データ準備 (1)

- トークン化と小文字化されたデータのディレクトリ作成

```
$ mkdir mydata mydata/tok mydata/low
```

- kyoto-test の日本語を分割し、小文字化する

```
$ usr/bin/kytea -notags < data/orig/kyoto-test.ja > mydata/tok/kyoto-test.ja
$ mosesdecoder/scripts/tokenizer/lowercase.perl <
      mydata/tok/kyoto-test.ja > mydata/low/kyoto-test.ja
```

- kyoto-test の英語をトークン化し、小文字化する

```
$ mosesdecoder/scripts/tokenizer/tokenizer.perl <
      data/orig/kyoto-test.en > mydata/tok/kyoto-test.en
$ mosesdecoder/scripts/tokenizer/lowercase.perl <
      mydata/tok/kyoto-test.en > mydata/low/kyoto-test.en
```

- kyoto-dev, kyoto-tune, kyoto-train で同じ処理

演習：データ準備 (2)

- データから 1 単語未満、 40 単語を超える文を削除

```
$ mosesdecoder/scripts/training/clean-corpus-n.perl
      mydata/low/kyoto-train ja en mydata/low/kyoto-train.cln 1 40
```

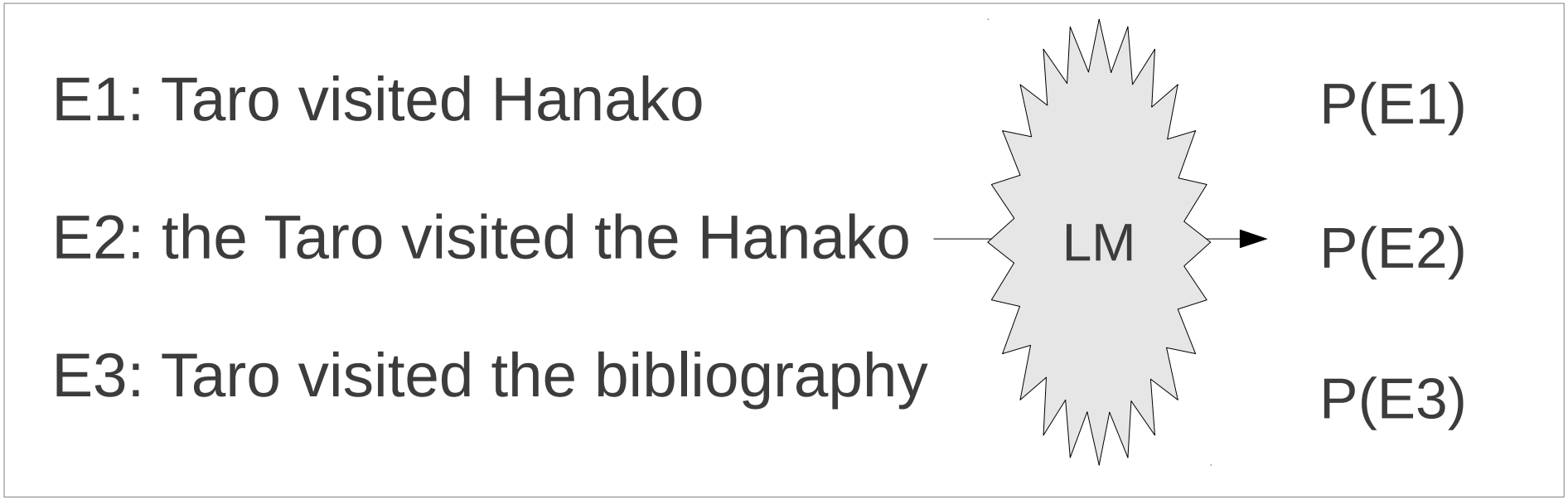
- 演習時間削減のため、 2 万文だけの学習データを作成（通常は全てを利用）

```
$ head -20000 < mydata/low/kyoto-train.cln.ja > mydata/low/kyoto-train.small.ja
$ head -20000 < mydata/low/kyoto-train.cln.en > mydata/low/kyoto-train.small.en
```

モデル学習

言語モデル

- 目的言語側の各文に確率を与える



- 良い言語モデル：流暢性の高い文に高い確率を

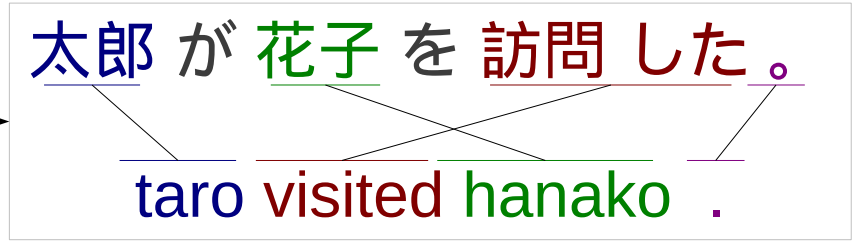
P(E1) > P(E2)　　P(E1) > P(E3)

アライメント

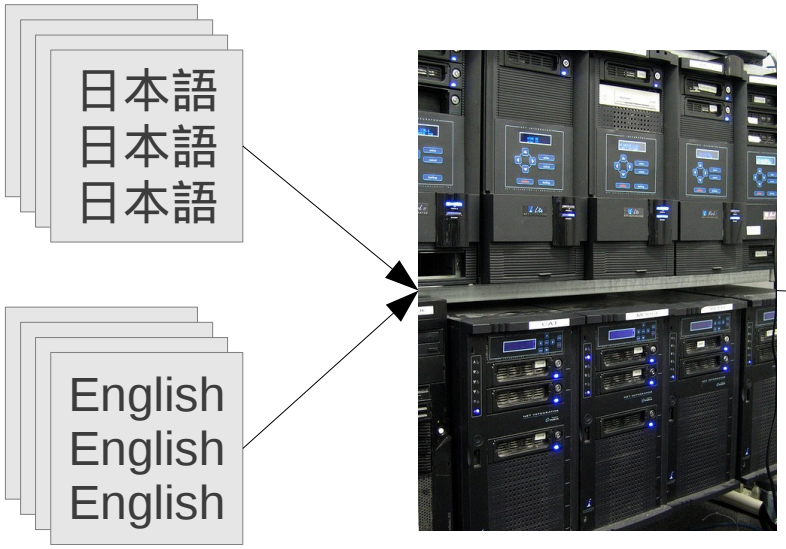
- 文内の単語対応を発見

太郎 が 花子 を 訪問 した 。

taro visited hanako .



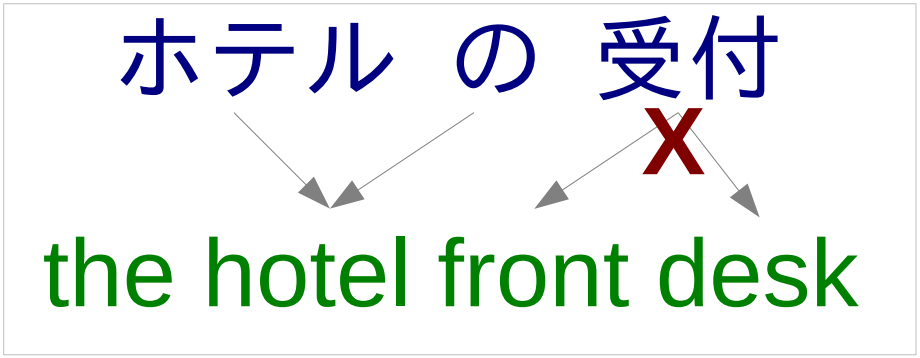
- 確率モデルによる自動学習（教師なし）が主流

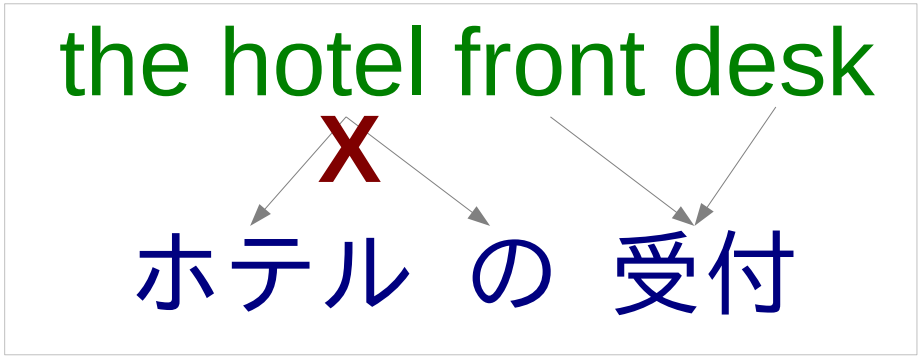


P(花子 |hanako) = 0.99
P(太郎 |taro) = 0.97
P(visited| 訪問) = 0.46
P(visited| した) = 0.04
P(花子 |taro) = 0.0001

IBM/HMM モデル [Brown+ 93, Vogel+ 96]

- １対多アライメントモデル





- IBM Model 1: 語順を考慮しない
- IBM Models 2-5, HMM: 徐々に考慮する情報を導入（精度・計算コスト ++ ）

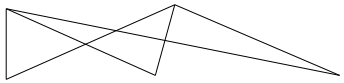
EM アルゴリズム

- IBM モデルは EM(Expectation-Maximization) アルゴリズムで学習
- アイデア：
  - E ステップ：確率モデルに基づいて単語の対応を推定
  - M ステップ：単語の対応に基づいてモデル更新
- 反復を繰り返すことでモデルの推定精度を向上

EM アルゴリズムの例

- 初期状態：単語の対応は未知、確率も一様

チーズ ムース

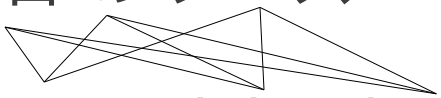


Mousse di formaggi

本日 の 鮮魚

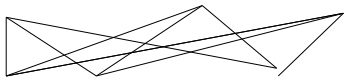
Pesce del giorno

本日 の チーズ



Formaggi del giorno

ドルチェ と チーズ



Dolce e Formaggi

P(チーズ |formaggi) = 0.16
P(ムース |formaggi) = 0.16
P(本日 |formaggi) = 0.16
P(の |formaggi) = 0.16
P(ドルチェ |formaggi) = 0.16
P(と |formaggi) = 0.16

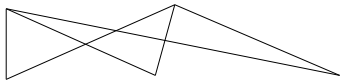
P(本日 |giorno) = 0.25
P(の |giorno) = 0.25
P(鮮魚 |giorno) = 0.25
P(チーズ |giorno) = 0.25

…

EM アルゴリズムの例

- M ステップ：確率を更新

チーズ ムース

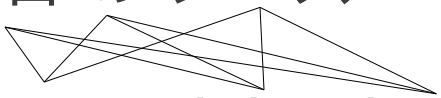


Mousse di formaggi

本日 の 鮮魚

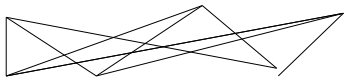
Pesce del giorno

本日 の チーズ



Formaggi del giorno

ドルチェ と チーズ



Dolce e Formaggi

P(チーズ |formaggi) = 0.375
P(ムース |formaggi) = 0.125
P(本日 |formaggi) = 0.125
P(の |formaggi) = 0.125
P(ドルチェ |formaggi) = 0.125
P(と |formaggi) = 0.125

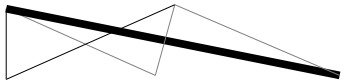
P(本日 |giorno) = 0.33
P(の |giorno) = 0.33
P(鮮魚 |giorno) = 0.16
P(チーズ |giorno) = 0.16

...

EM アルゴリズムの例

- E ステップ：単語対応を更新

チーズ ムース



Mousse di formaggi

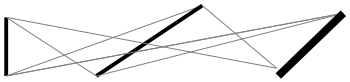
本日 の 鮮魚

Pesce del giorno

本日 の チーズ

Formaggi del giorno

ドルチェ と チーズ



Dolce e Formaggi

P(チーズ |formaggi) = 0.375
P(ムース |formaggi) = 0.125
P(本日 |formaggi) = 0.125
P(の |formaggi) = 0.125
P(ドルチェ |formaggi) = 0.125
P(と |formaggi) = 0.125

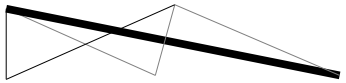
P(本日 |giorno) = 0.33
P(の |giorno) = 0.33
P(鮮魚 |giorno) = 0.16
P(チーズ |giorno) = 0.16

...

EM アルゴリズムの例

- M ステップ：確率を更新

チーズ ムース



Mousse di formaggi

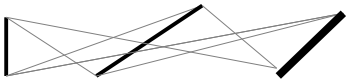
本日 の 鮮魚

Pesce del giorno

本日 の チーズ

Formaggi del giorno

ドルチェ と チーズ



Dolce e Formaggi

P(チーズ |formaggi) = 0.9
P(ムース |formaggi) = 0.02
P(本日 |formaggi) = 0.02
P(の |formaggi) = 0.02
P(ドルチェ |formaggi) = 0.02
P(と |formaggi) = 0.02

P(本日 |giorno) = 0.48
P(の |giorno) = 0.48
P(鮮魚 |giorno) = 0.02
P(チーズ |giorno) = 0.02

...

EM アルゴリズムの例

- E ステップ：単語対応を更新

チーズ ムース

Mousse di formaggi

本日 の 鮮魚

Pesce del giorno

本日 の チーズ

Formaggi del giorno

ドルチェ と チーズ

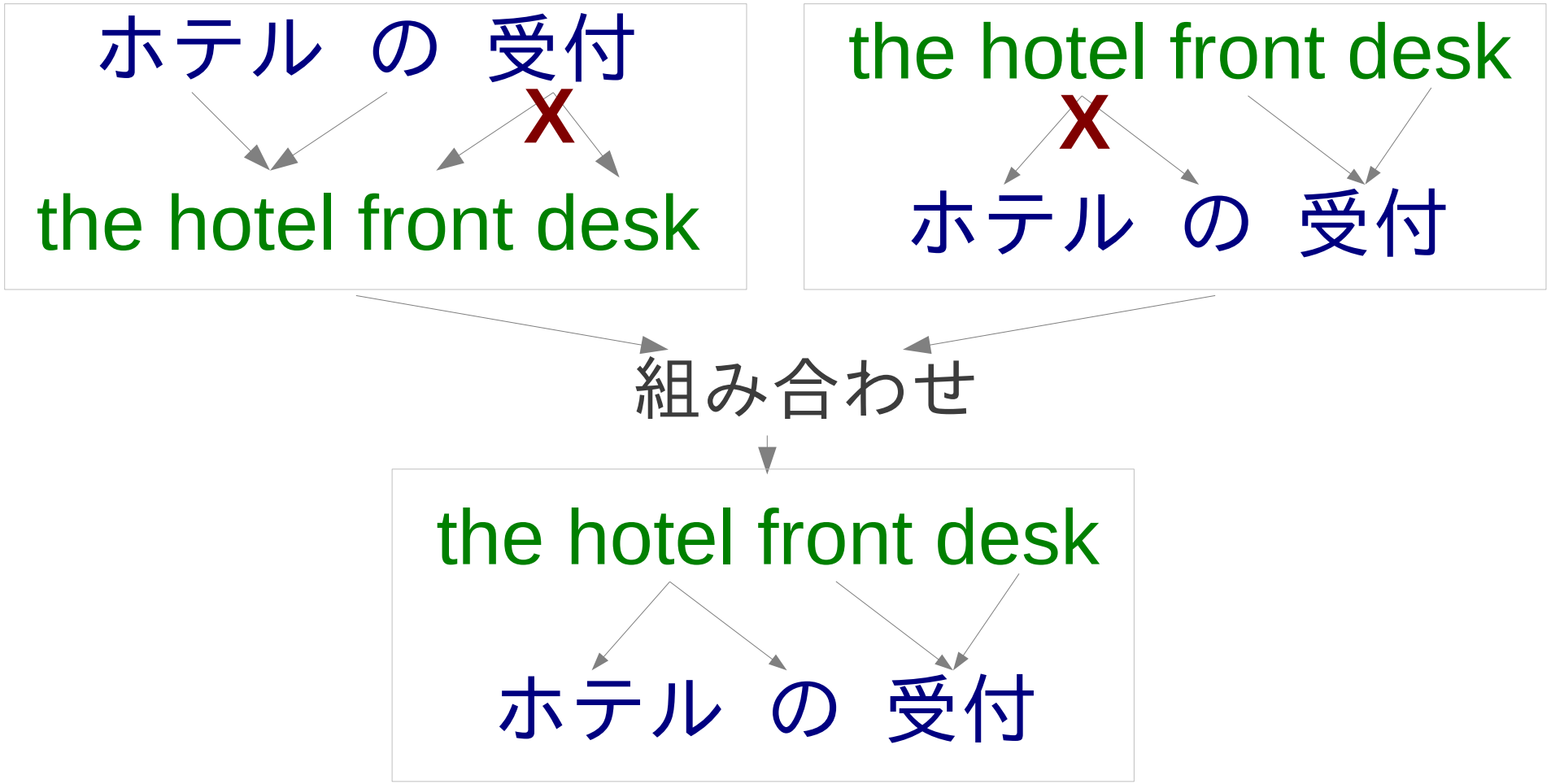
Dolce e Formaggi

P(チーズ |formaggi) = 0.9
P(ムース |formaggi) = 0.02
P(本日 |formaggi) = 0.02
P(の |formaggi) = 0.02
P(ドルチェ |formaggi) = 0.02
P(と |formaggi) = 0.02

P(本日 |giorno) = 0.48
P(の |giorno) = 0.48
P(鮮魚 |giorno) = 0.02
P(チーズ |giorno) = 0.02

...

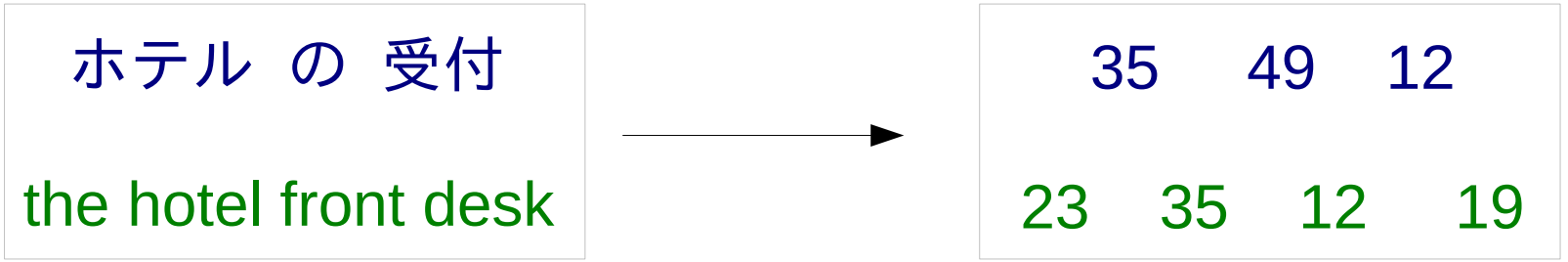
１対多アライメントの組み合わせ



- 様々なヒューリスティック手法（ grow-diag-final ）

ツール

- mkcls: 2 言語で単語クラスを自動発見



- GIZA++: IBM モデルによるアライメント（クラスを用いて確率を平滑化）

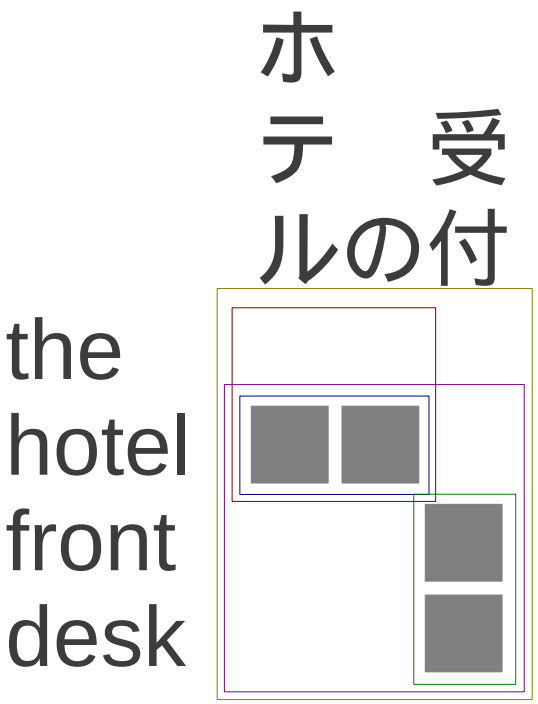
- symal: 両方向のアライメントを組み合わせる
- (Moses の train-model.perl の一部として実行される)

研究

- アライメントは本当に重要なのか？ [Ganchev+ 08]
- 教師ありアライメント [Fraser 06, Haghighi 09]
- 統語情報を使ったアライメント [DeNero 07]
- フレーズベースアライメント [Marcu 02, DeNero 08]

フレーズ抽出

- アライメントに基づいてフレーズを列挙



ホテル の → hotel
ホテル の → the hotel
受付 → front desk
ホテルの受付 → hotel front desk
ホテルの受付 → the hotel front desk

フレーズのスコア計算

- ５つの標準的なスコアでフレーズの信頼性・使用頻度
  - フレーズ翻訳確率
    P(**f**|**e**) = c(**f**,**e**)/c(**e**)　　P(**e**|**f**) = c(**f**,**e**)/c(**f**)

    例：　c(ホテル　の , the hotel) / c(the hotel)
  - 語彙 (lexical) 翻訳確率
    - フレーズ内の単語の翻訳確率を利用 (IBM Model 1)
    - 低頻度のフレーズ対の信頼度判定に役立つ

    $P(f|e) = \Pi_f 1/|\mathbf{e}| \sum_e P(f|e)$

    例：
    (P(ホテル |the)+P(ホテル |hotel))/2 * (P(の |the)+P(の |hotel))/2
  - フレーズペナルティ：すべてのフレーズで 1

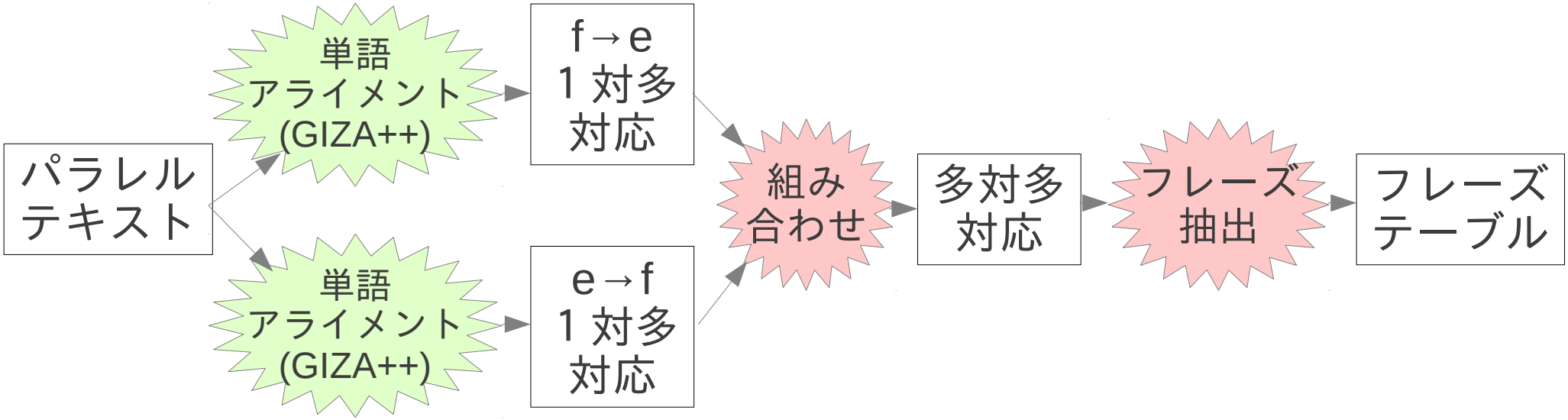
ツール

- extract: フレーズ抽出
- phrase-extract/score: フレーズのスコア付け
- (Moses の train-model.perl の一部として実行される)

研究

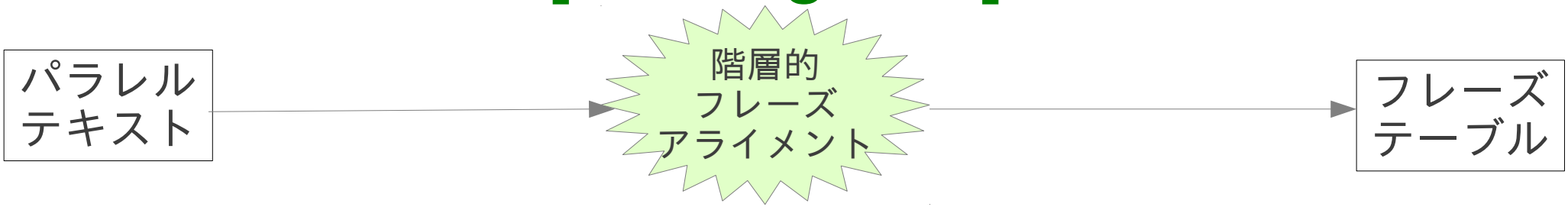
- 翻訳モデルの分野適用 [Koehn 07, Matsoukas 09]
- 不要・信頼度の低いフレーズの削除 [Johnson 07]
- フレーズ曖昧性解消 [Carpuat 07]

従来の流れの長所短所



+ 意外と強力、 Moses などで広く使われる
- 複雑でヒューリスティックスがたくさん
- 段階に分かれるため、最終目的であるフレーズテーブルの構築に合わないアライメントを獲得
- 網羅的に抽出されたフレーズテーブルは膨大

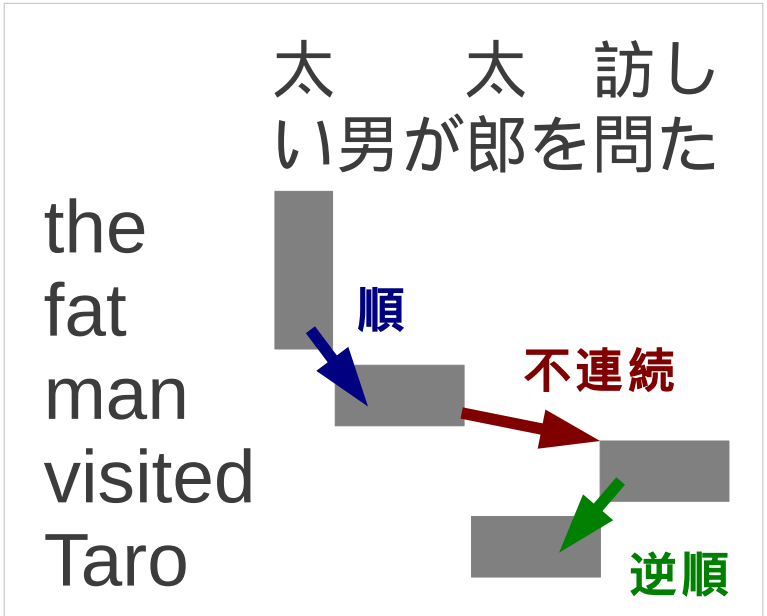
同時アライメント・抽出 [Neubig+ 11]



- \+ 直接フレーズテーブルをモデル化できる
  + ヒューリスティックスの必要がなくなる
  + 精度を保ちながらフレーズテーブルのサイズが減らせる
- Inversion Transduction Grammar (ITG) を用いた階層的モデルで実現
- pialign ツールキットで実装
  http://www.phontron.com/pialign

語彙化並べ替えモデル

- 順・逆順・不連続



太い → the fat
順の確率が高い

太郎 を → Taro
逆順の確率が高い

- 入力・出力、右・左などで条件付けた確率

ツール

- extract: フレーズ抽出と同一
- lexical-reordering/score: 並べ替えモデルを学習
- (Moses の train-model.perl の一部として実行される)

演習：翻訳モデルの学習 (1)

- 学習を一気に行うスクリプト：

```
$ mosesdecoder/scripts/training/train-model.perl
    -parallel
    -external-bin-dir $HOME/alagin/usr/local/giza-pp
    -root-dir $HOME/alagin/train
    -corpus $HOME/alagin/mydata/low/kyoto-train.small
    -f en -e ja
    -alignment grow-diag-final-and
    -reordering msd-bidirectional-fe
    -lm 0:3:$HOME/alagin/lm/kyoto-train.ja.blm:8 &> training.log
```

演習：翻訳モデルの学習 (2)

- 学習後、アライメントの結果を閲覧
- アライメント： train/model/aligned.grow-diag-final-and
  データ： mydata/low/kyoto-train.small.{ja,en}
- 閲覧の前に、データを head で削減

```
$ head -100 train/model/aligned.grow-diag-final-and > mydata/align.txt
$ head -100 mydata/low/kyoto-train.small.ja > mydata/ja.txt
$ head -100 mydata/low/kyoto-train.small.en > mydata/en.txt
```

- 閲覧 UI を起動し、これらを選択

```
$ java -jar download/AlignmentTool.jar
```

機械翻訳の評価

翻訳の曖昧性

- 他の機械学習・言語処理タスクと違い、翻訳には大きな曖昧性

花子 が 太郎 に 会った
Hanako met Taro
Hanako met to Taro
Hanako ran in to Taro
Taro met Hanako
The Hanako met the Taro

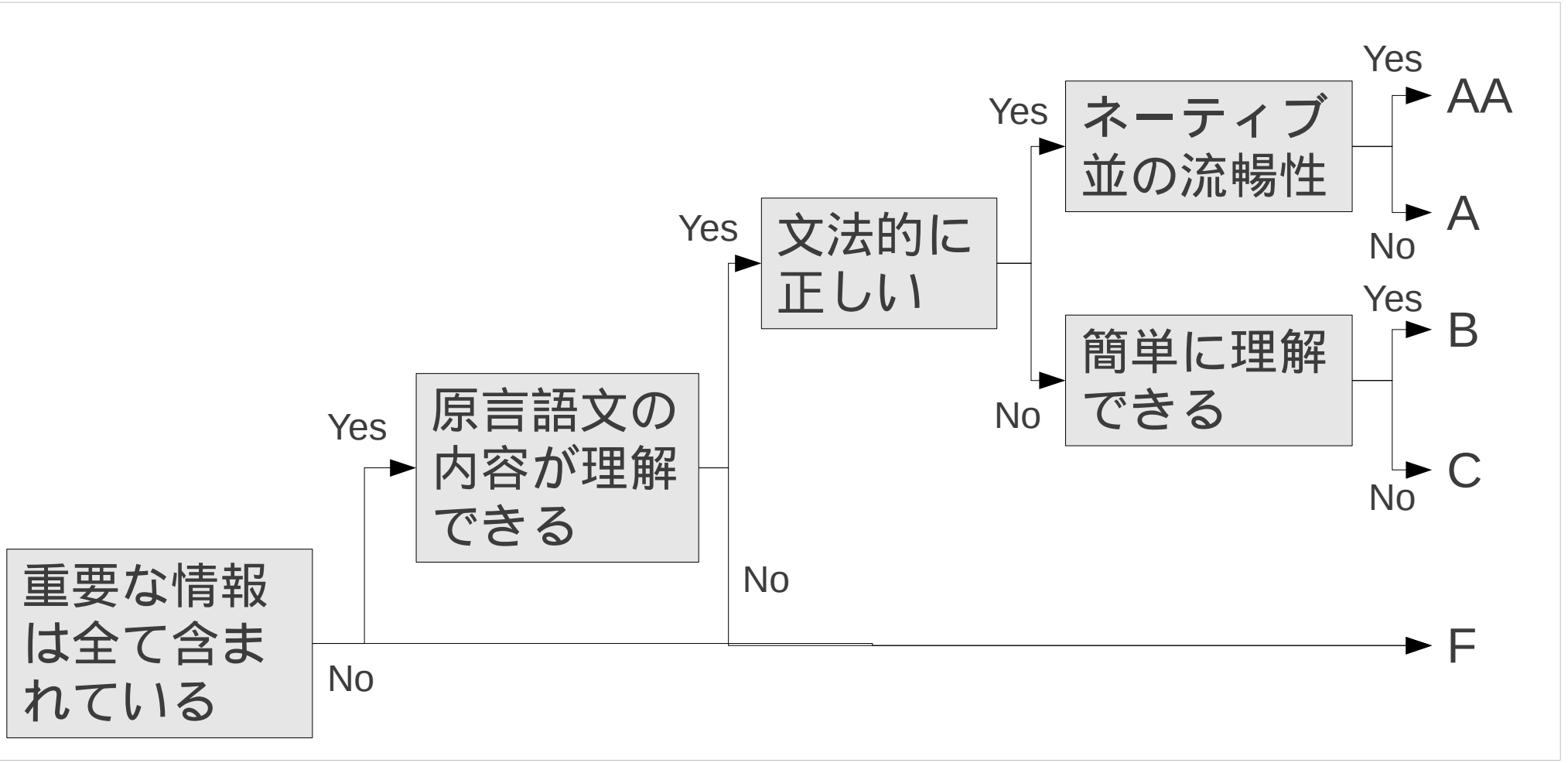
- → 翻訳の評価がとても難しい

人手評価

- 意味的妥当性 (Adequacy): 原言語文の意味が伝わるか
- 流暢性 (Fluency): 目的言語文が自然か
- 比較評価 (Pairwise): X と Y どっちの方が良いか

許容性 (Acceptability) [Goto+ 11]

- 妥当性と流暢性を組み合わせた評価基準



自動評価

- システム出力は正解文に一致するか

正解文 :	Taro visited Hanako
システム出力 :	the Taro visited the Hanako

- 翻訳の正解は単一ではないため、複数の正解を利用することも

正解文 :	Taro visited Hanako Taro ran in to Hanako
システム出力 :	the Taro visited the Hanako

BLEU [Papineni+ 03]

- 最初に提案された自動評価尺度
- 未だに最も標準的
- n-gram 適合率 + 短さペナルティ

Reference: Taro visited Hanako

System: the Taro visited the Hanako

1-gram: 3/5
2-gram: 1/4
brevity penalty = 1.0

Brevity: min(1, |System|/|Reference|) = min(1, 5/3)

BLEU-2 = $(3/5*1/4)^{1/2} * 1.0$
= 0.387

RIBES [Isozaki+ 10]

- 語順に着目した評価尺度
- 単語の順位相関を利用

正解: Taro visited Hanako

	Taro visits Hanako	Hanako Taro visited
	1 X 3	3 1 2
RIBES:	0.9036	0.3333
BLEU(+1):	0.5773	0.7598

- 日英・英日翻訳で異なる手法（RBMT・SMT）でも人間評価と高い相関

研究

- 言語資源を用いた評価尺度
  - METEOR: 類義語 [Banerjee 05]
  - MEANT: 意味解析 [Lo 11]
- チューニングに良い評価尺度 [Cer 10]
- 複数の評価尺度の利用 [Albrecht 07]
- 評価のクラウドソーシング [Callison-Burch 11]

演習：人手評価

- 翻訳結果を格納したファイル：
  download/en-ja-result.tsv
- 好きな表計算ソフトで開き、数文に Acceptability 評価を付ける

演習：自動評価

- download/kyoto-test-{pb,hiero,preorder}.ja に３つの翻訳器の出力
- BLEU による評価

```
$ mosesdecoder/scripts/generic/multi-bleu.perl
          data/tok/kyoto-test.ja < download/kyoto-test-pb.ja
$ mosesdecoder/scripts/generic/multi-bleu.perl
          data/tok/kyoto-test.ja < download/kyoto-test-hiero.ja
$ mosesdecoder/scripts/generic/multi-bleu.perl
          data/tok/kyoto-test.ja < download/kyoto-test-preorder.ja
```

- RIBES による評価

```
$ python3 usr/local/RIBES-1.02.3/RIBES.py
    -r data/tok/kyoto-test.ja download/kyoto-test-{pb,hiero,preorder}.ja
```

チューニング

チューニング [Och+ 02]

- 各モデルのスコアを組み合わせた解のスコア

	LM	TM	RM	
○ Taro visited Hanako	-4	-3	-1	-8
X the Taro visited the Hanako	-5	-4	-1	-10
X Hanako visited Taro	-2	-3	-2	-7 ← 最大 X

- スコアを重み付けると良い結果が得られる

	LM	TM	RM	
○ Taro visited Hanako	0.2*-4	0.3*-3	0.5*-1	-2.2 ← 最大 ○
X the Taro visited the Hanako	0.2*-5	0.3*-4	0.5*-1	-2.7
X Hanako visited Taro	0.2*-2	0.3*-3	0.5*-2	-2.3

- チューニングは重みを発見 : $w_{LM}=0.2$ $w_{TM}=0.3$ $w_{RM}=0.5$

対数線形モデル

- 訳文の対数確率を素性の重み付き和で組み合わせる

$$P(E|F) = \frac{1}{Z} \sum_i e^{\lambda_i \phi_i(E,F)}$$

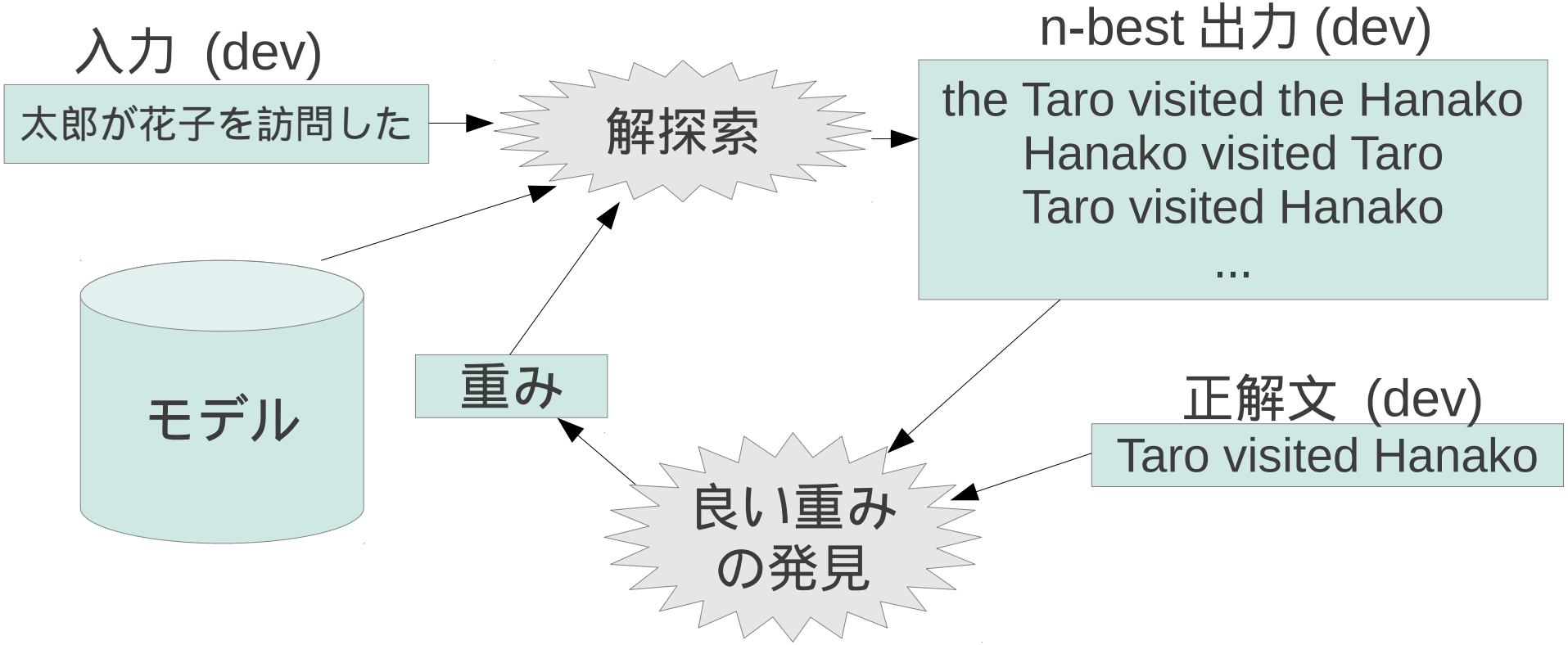
- φ_i は各モデルの対数確率

$$\phi_1(E,F) = \log P_{LM}(E)$$
$$\phi_2(E,F) = \log P_{TM}(E|F)$$
$$\phi_3(E,F) = \log P_{TM}(F|E)$$

…

誤り率最小化 (MERT) [Och 03]

- BLEU などの評価尺度を候補生成・重み調整の反復で最適化



MERT の重み更新

- 重みを１個ずつ更新

	重み			BLEU
	w_{LM}	w_{TM}	w_{RM}	
初期値:	0.1	0.1	0.1	0.20
w_{LM} を最適化	0.4	0.1	0.1	0.32
w_{TM} を最適化	0.4	0.1	0.1	0.32
w_{RM} を最適化	0.4	0.1	0.3	0.4
w_{LM} を最適化	0.35	0.1	0.3	0.41
w_{TM} を最適化				

重み１つの最適化

- 入力：
  n-best

f_1	ϕ_{LM}	ϕ_{TM}	ϕ_{RM}	BLEU
$e_{1,1}$	1	0	-1	0
$e_{1,2}$	0	1	0	1
$e_{1,3}$	1	0	1	0

f_2	ϕ_{LM}	ϕ_{TM}	ϕ_{RM}	BLEU
$e_{2,1}$	1	0	-2	0
$e_{2,2}$	3	0	1	0
$e_{2,3}$	2	1	2	1

固定された重み

$$w_{LM}=-1,\ w_{TM}=1$$

調整する重み

$$w_{RM}=???$$

重み１つの最適化

- 各仮説を線へ変換

$$y = a\,x + b$$

- 変数：
  - a は調整する素性の値
  - b は固定された素性の重み付き和
  - x は調整する重み（未知）

重み１つの最適化

- 例：

w_{LM}=-1, w_{TM}=1, w_{RM}=???

$$y = a\,x + b$$

$$a = \phi_{RM} \qquad b = w_{LM}\,\phi_{LM} + w_{TM}\,\phi_{TM}$$

f_1	ϕ_{LM}	ϕ_{TM}	ϕ_{RM}
$e_{1,1}$	1	0	-1
$e_{1,2}$	0	1	0
$e_{1,3}$	1	0	1

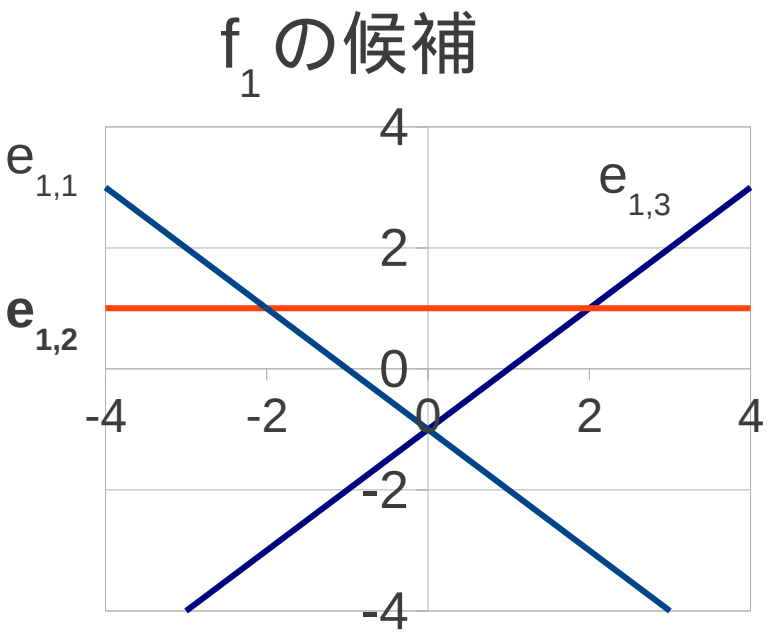
$a_{1,1}$=-1 $b_{1,1}$=-1

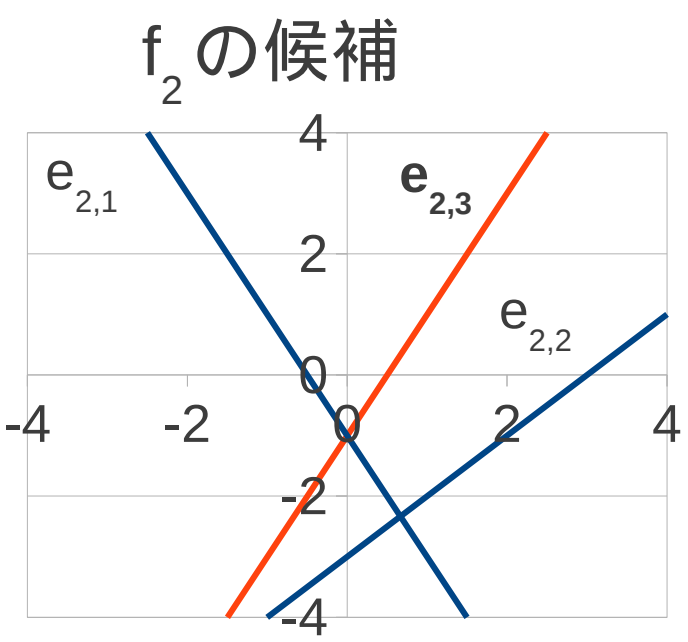
$a_{1,2}$=0 $b_{1,2}$=1

$a_{1,3}$=1 $b_{1,3}$=-1

重み１つの最適化

- 線をフラフに描写： $y = a x + b$



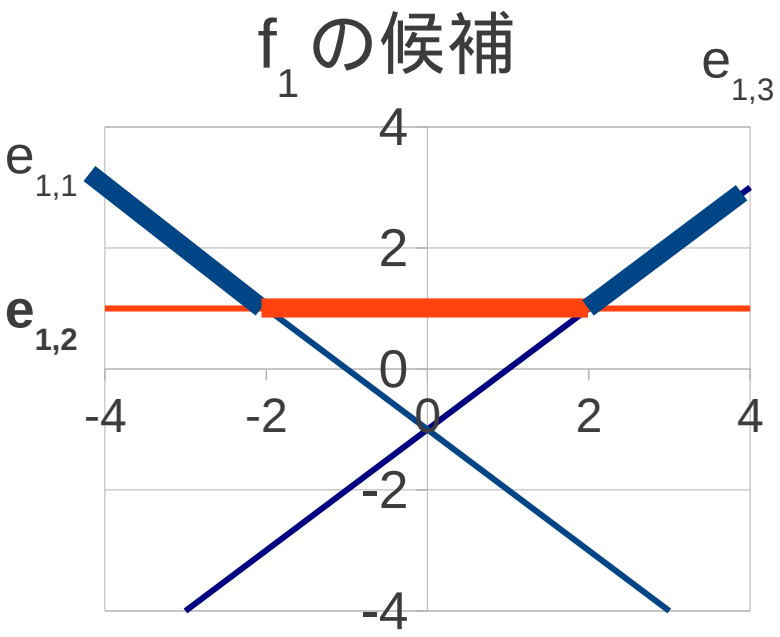


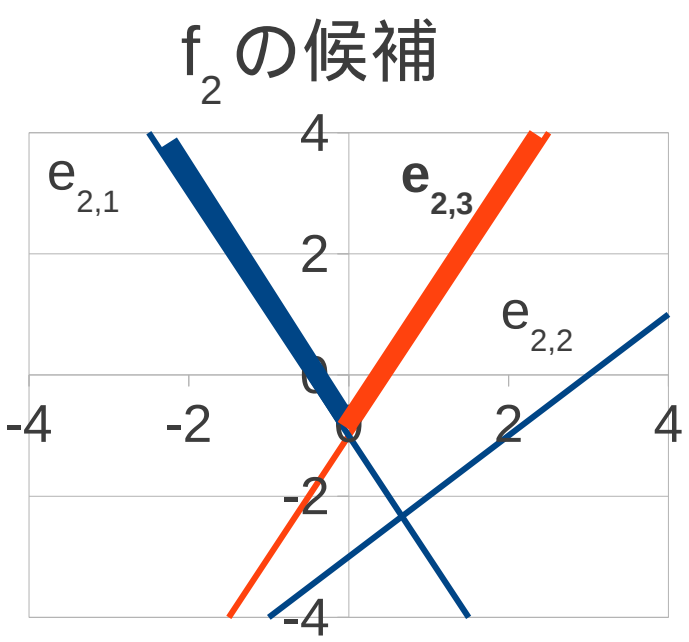
$a_{1,1}$=-1 $b_{1,1}$=-1
$a_{1,2}$=0 $b_{1,2}$=1
$a_{1,3}$=1 $b_{1,3}$=-1

$a_{2,1}$=-2 $b_{2,1}$=-1
$a_{2,2}$=1 $b_{2,2}$=-3
$a_{2,3}$=-2 $b_{2,3}$=1

重み１つの最適化

- *x* の各域における最も値の高い線を見つける

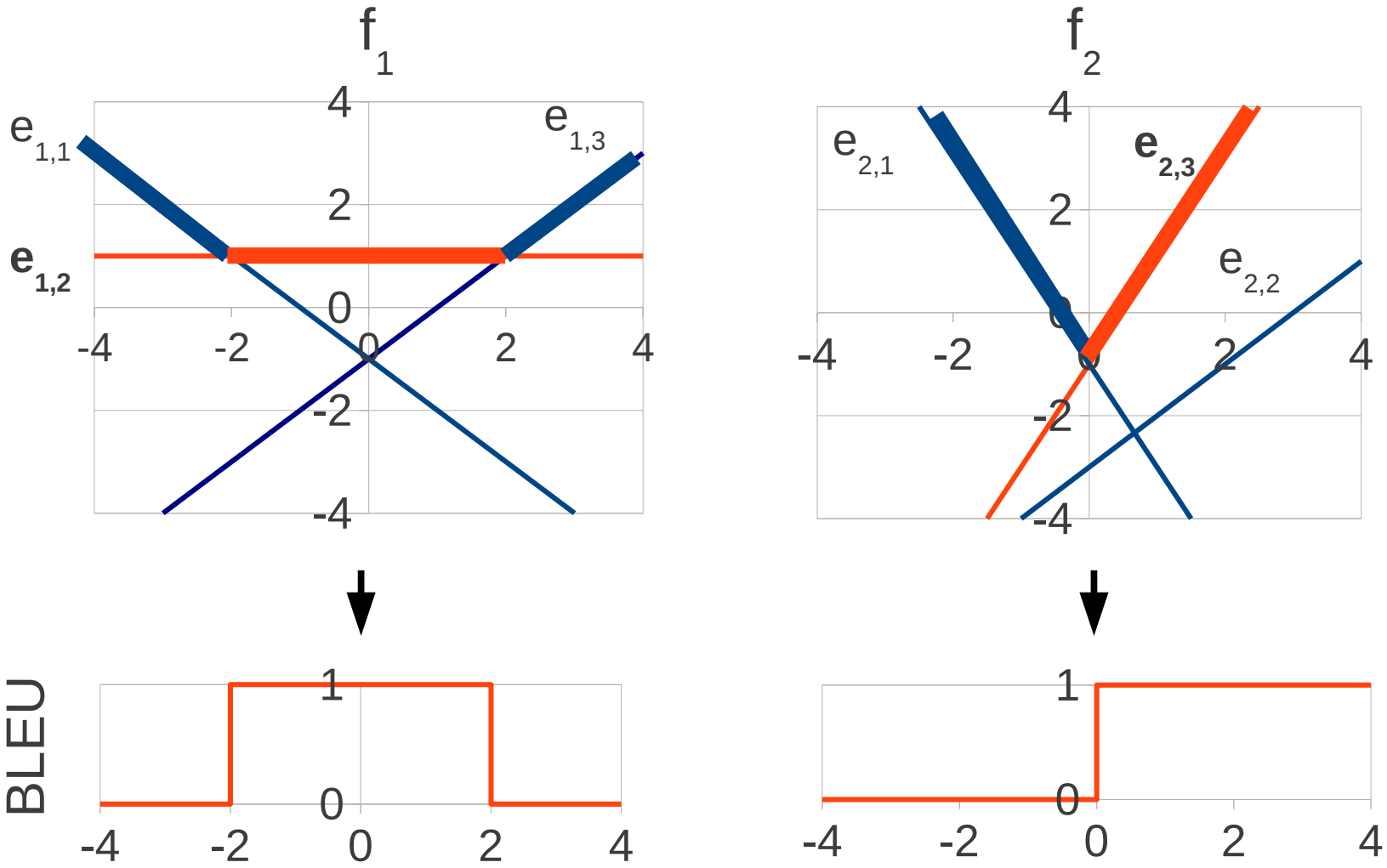




- これは凸包 (convex hull) と呼ぶ

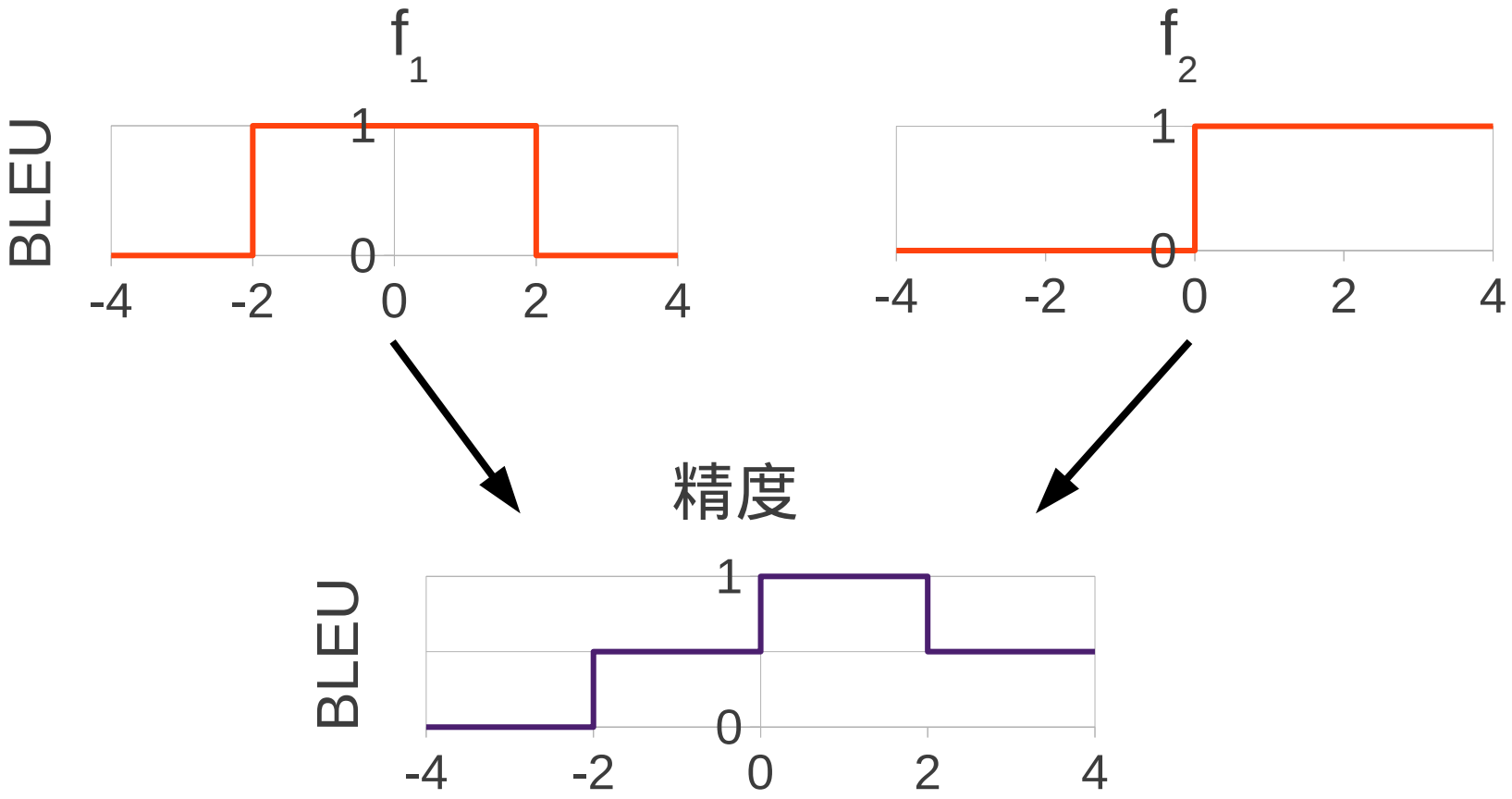
重み１つの最適化

- 凸包を用いて各域におけるスコアを計算



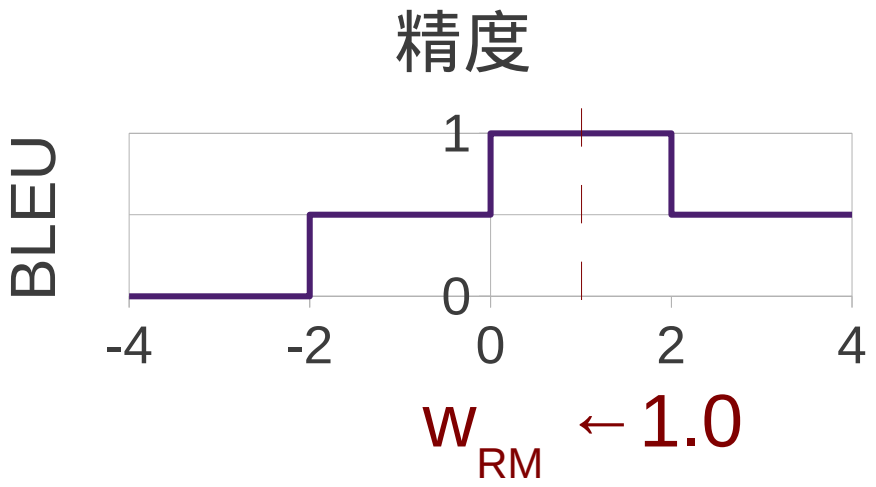
重み１つの最適化

- 複数の文を１つの精度グラフに変換



重み１つの最適化

- 最も精度の高い域の真ん中を新しい重みの値として選択



まとめ

- 各文に対して：
  - n-best 候補を作成
  - 線に変換し、凸包を作成
  - 凸包を精度グラフに変換
- 全文の精度グラフを組み合わせる
- 最もスコアの高い域を見つける
- 重みの値を精度が高くなるように更新

その他の学習法

- オンライン学習 [Liang 06, Watanabe 07]
  - 1文ずつスコアが高くなるように重みを更新
  - + 多くの素性が利用可能
  - + 収束が早い
  - - コーパス全体を同時に考慮できない
- ランク学習 [Hopkins 11]
  - n-best 中の文で BLEU が高い順にスコアが並ぶように学習
  - + 多くの素性が利用可能
  - + 実装が簡単
  - - コーパス全体を同時に考慮できない

研究

- ラティス出力のチューニング [Macherey 08]
- チューニングの高速化 [Suzuki 11]
- 複数の評価尺度の同時チューニング [Duh 12]

演習：チューニング

- MERT を行う mert-moses.pl を利用

```
$ mosesdecoder/scripts/training/mert-moses.pl
      $HOME/alagin/mydata/low/kyoto-tune.en
      $HOME/alagin/mydata/low/kyoto-tune.ja
      $HOME/alagin/mosesdecoder/bin/moses
      $HOME/alagin/train/model/moses.ini
      -mertdir $HOME/alagin/mosesdecoder/bin/
      -working-dir $HOME/alagin/tune
      -decoder-flags "-threads 4" &> tuning.log
```

(decoder-flags で並列スレッド数を指定)

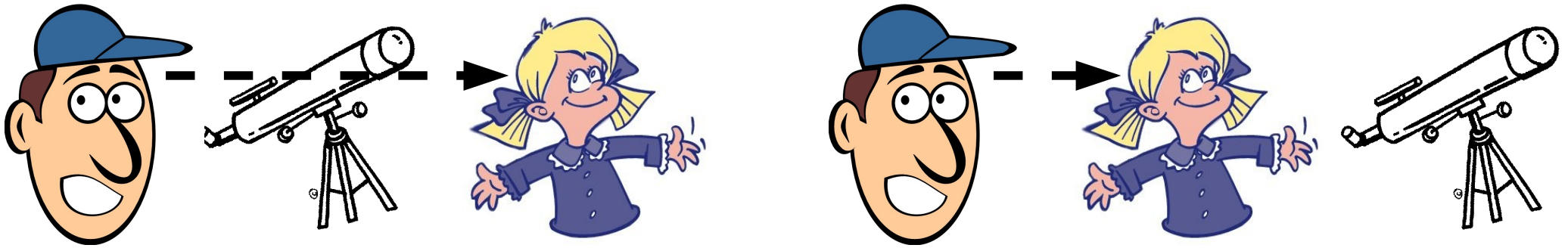
- チューニングが終われば BLEU の推移を表示

```
$ grep BLEU tune/*.moses.ini
```

構文解析

自然言語は曖昧性だらけ！

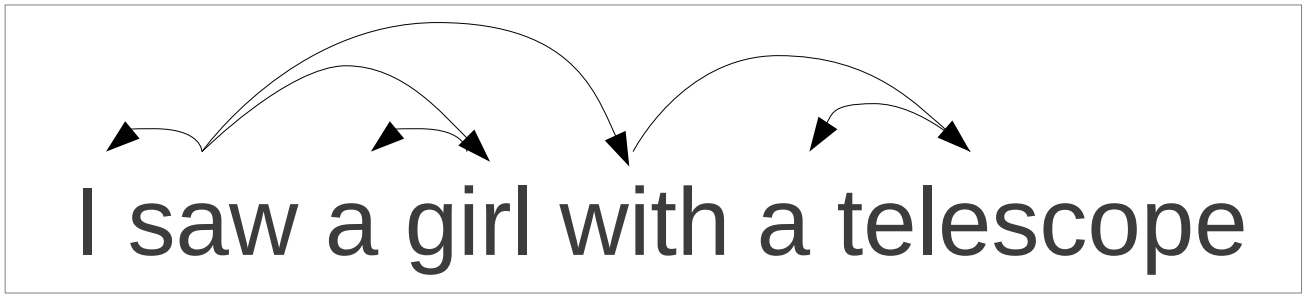
I saw a girl with a telescope



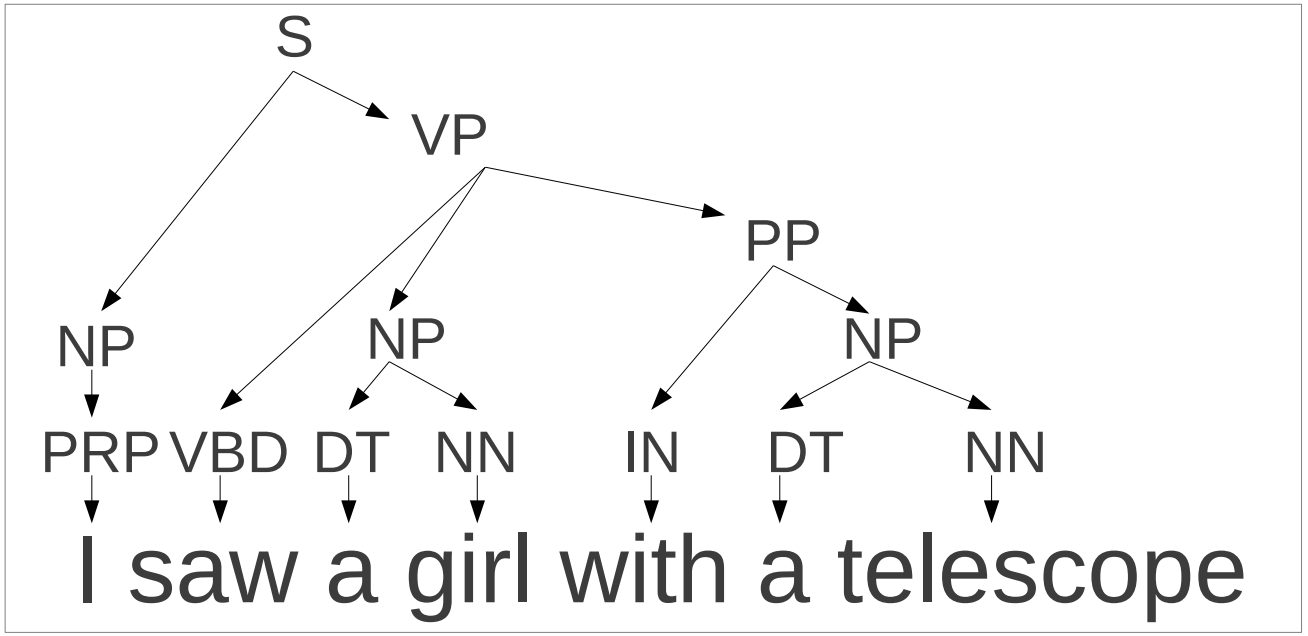
- 構文解析（パージング）は構造的な曖昧性を解消

構文解析の種類

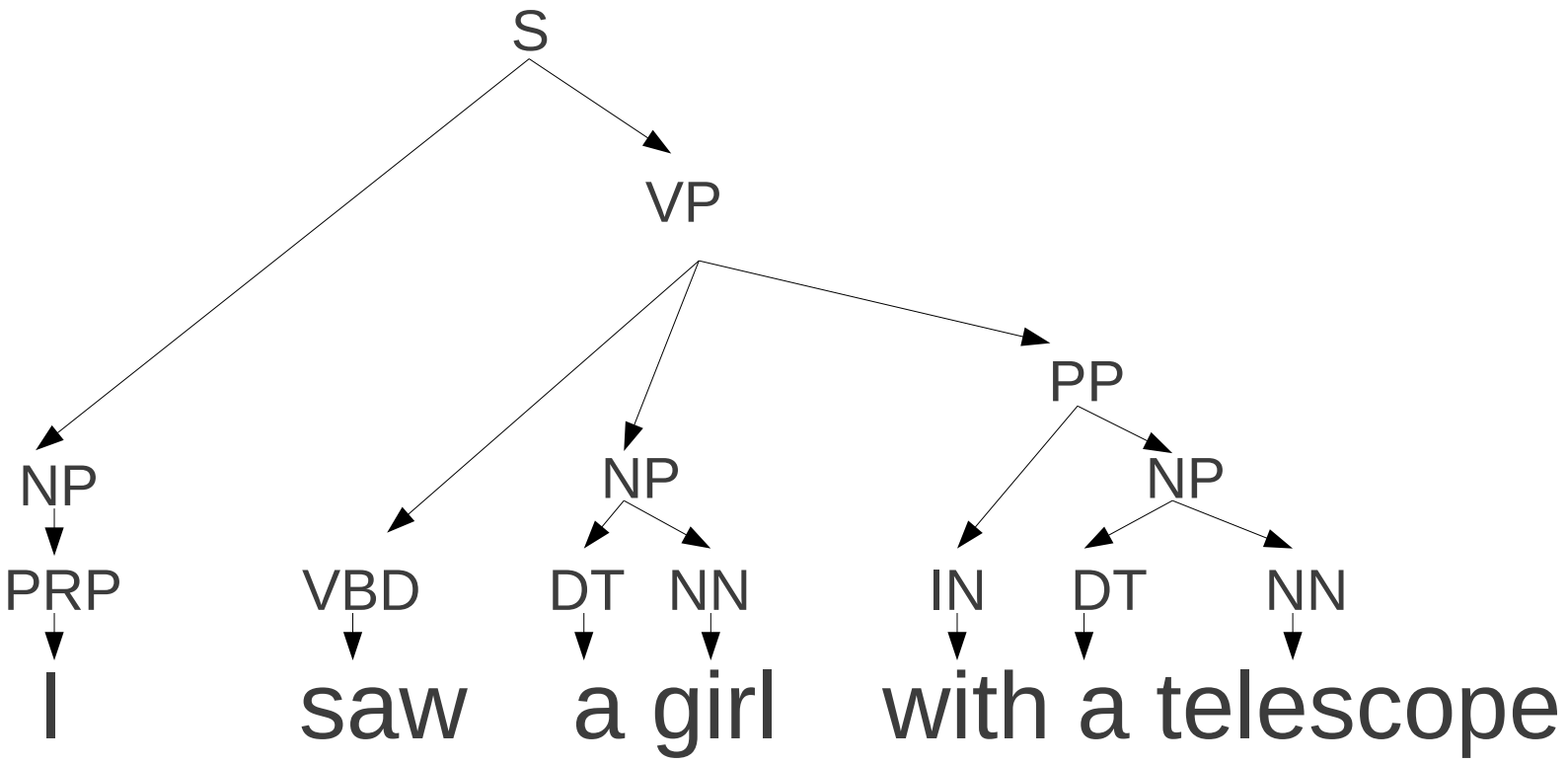
- 係り受け解析：単語と単語のつながりを重視



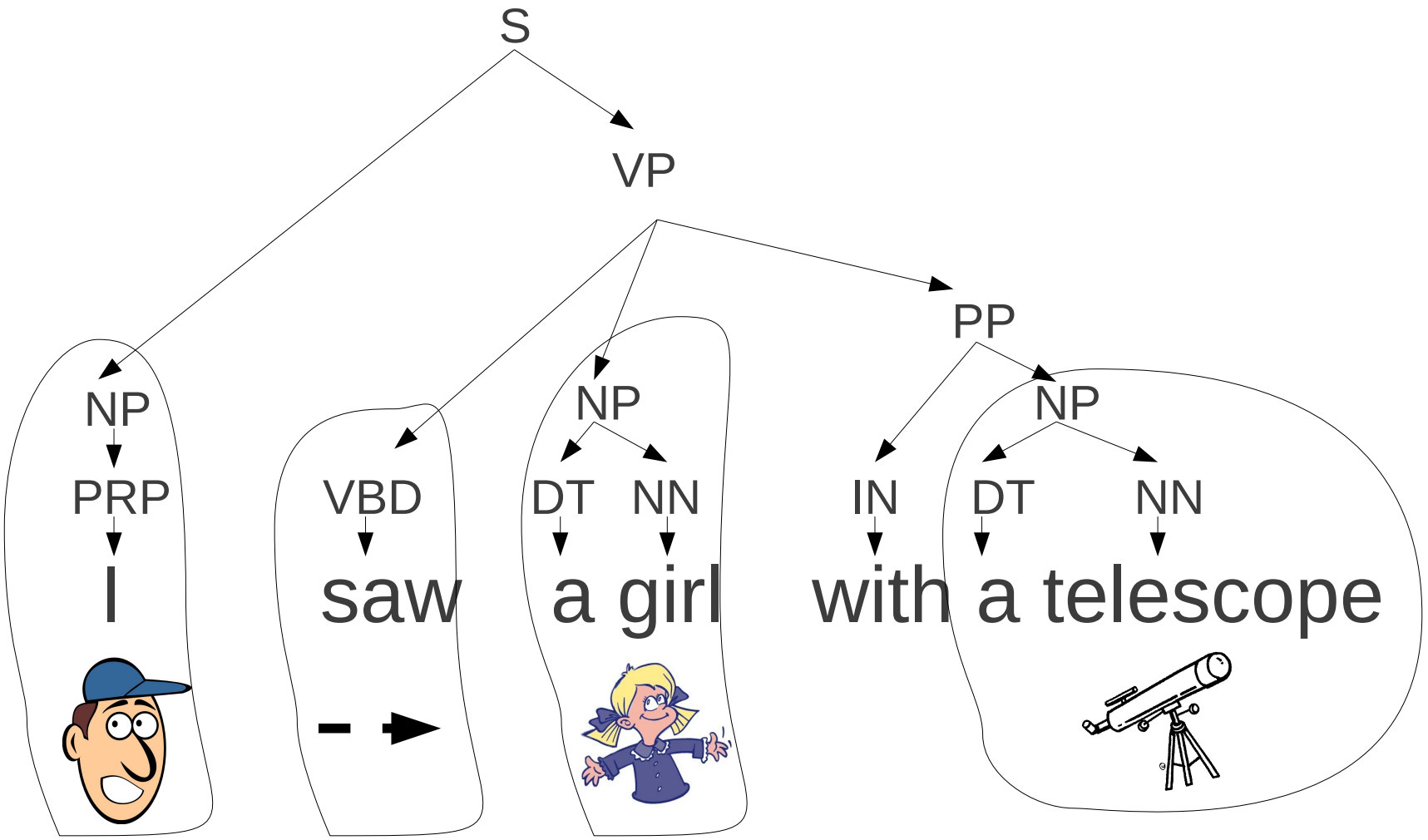
- 句構造解析：句とその再帰的な構造を重視



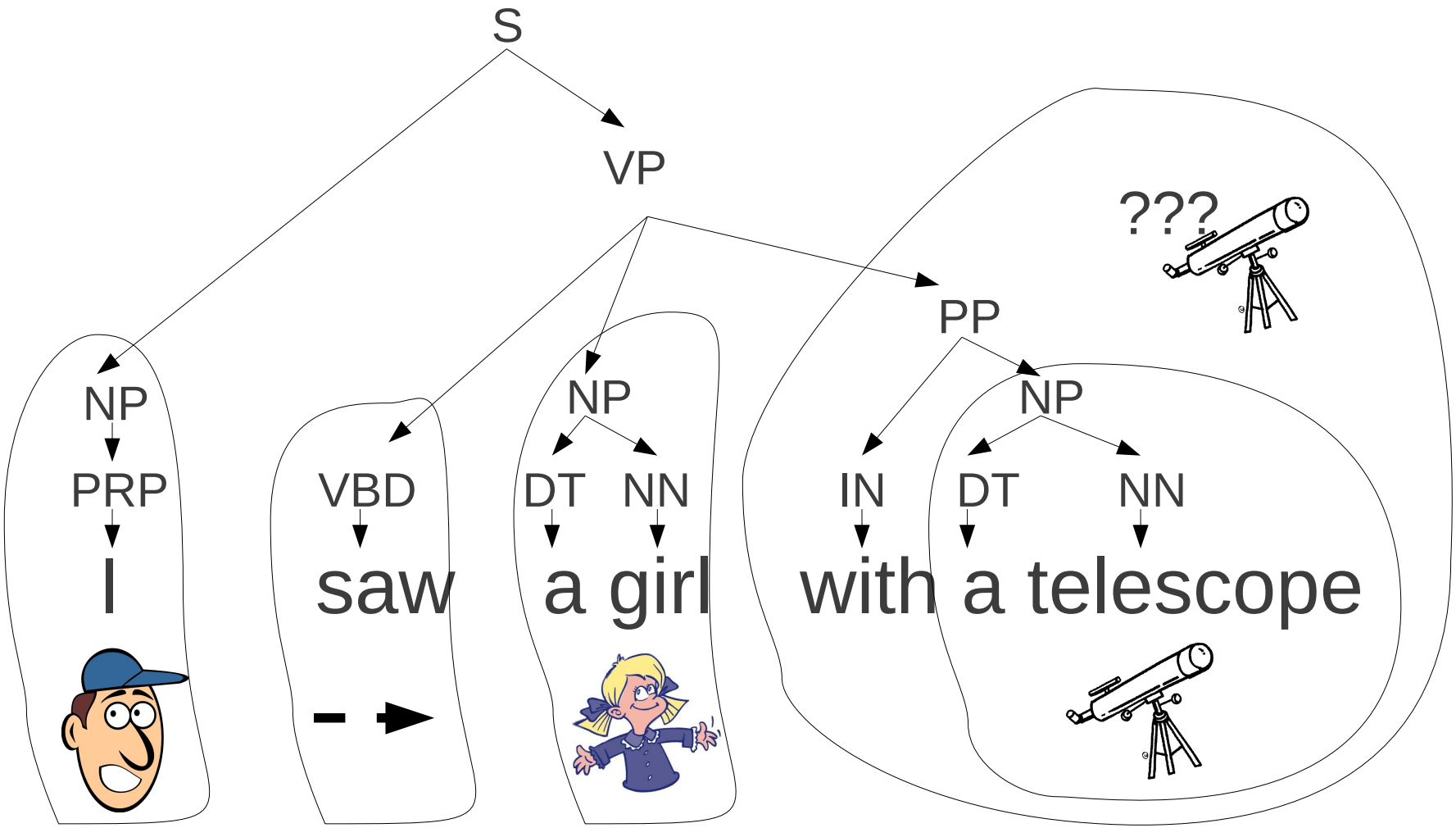
句の再帰的な構造


S
VP
PP
NP
NP
NP
PRP
VBD
DT
NN
IN
DT
NN
I
saw
a girl
with a telescope

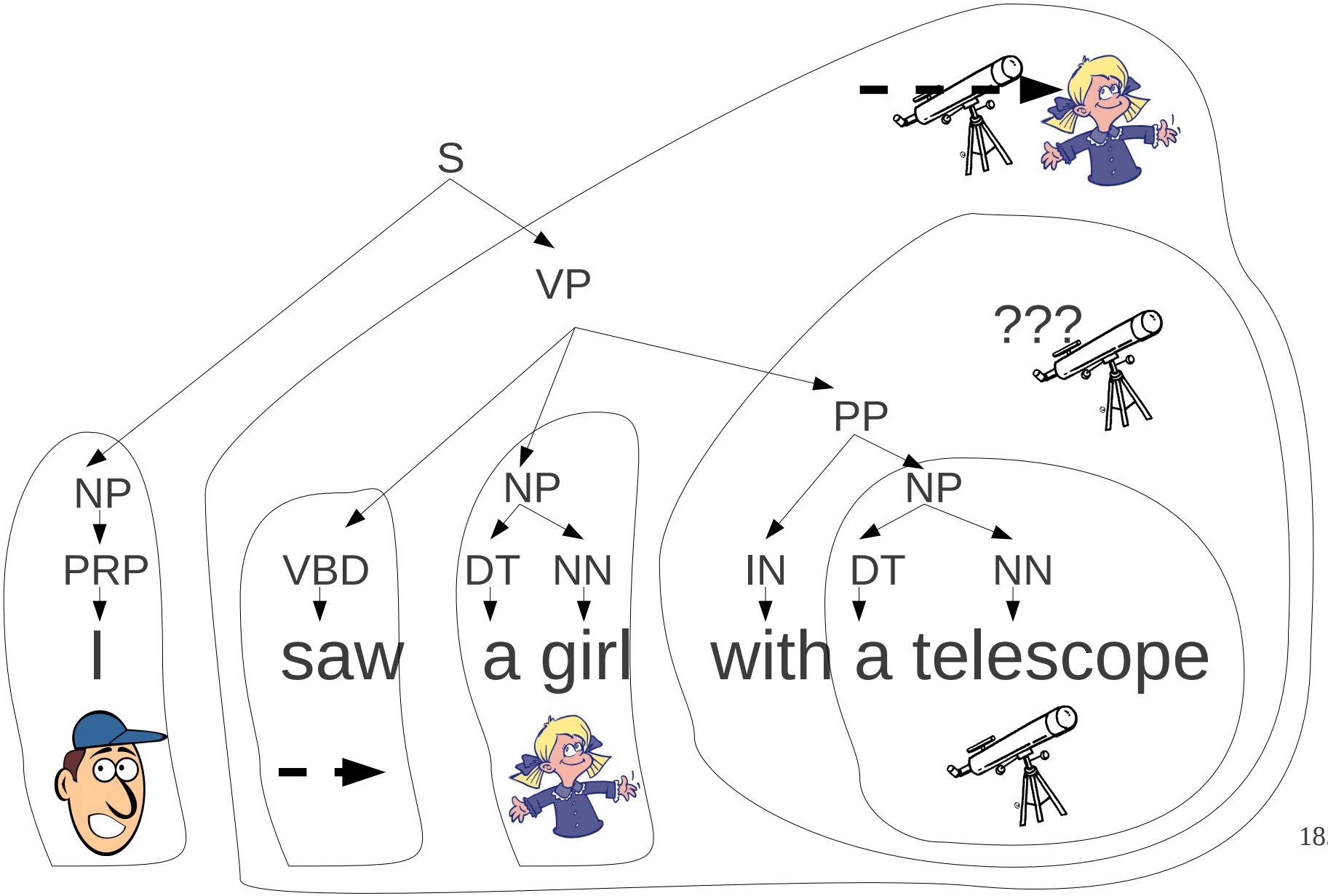
句の再帰的な構造


S
VP
PP
NP
PRP
I
VBD
saw
NP
DT
NN
a girl
IN
with
NP
DT
NN
a telescope

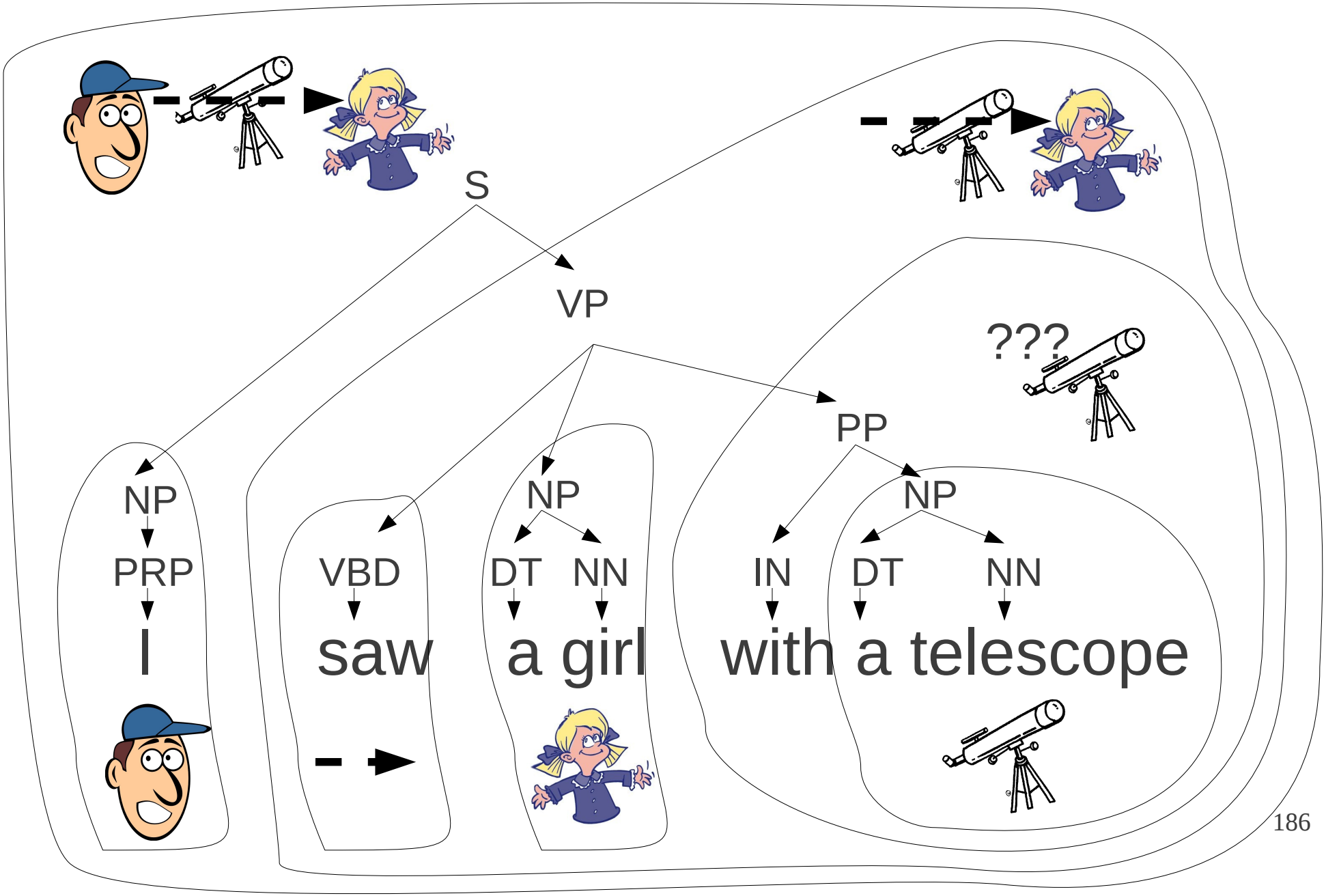
句の再帰的な構造


S
VP
PP
???
NP
PRP
I
VBD
saw
NP
DT
NN
a girl
IN
with
NP
DT
NN
a telescope

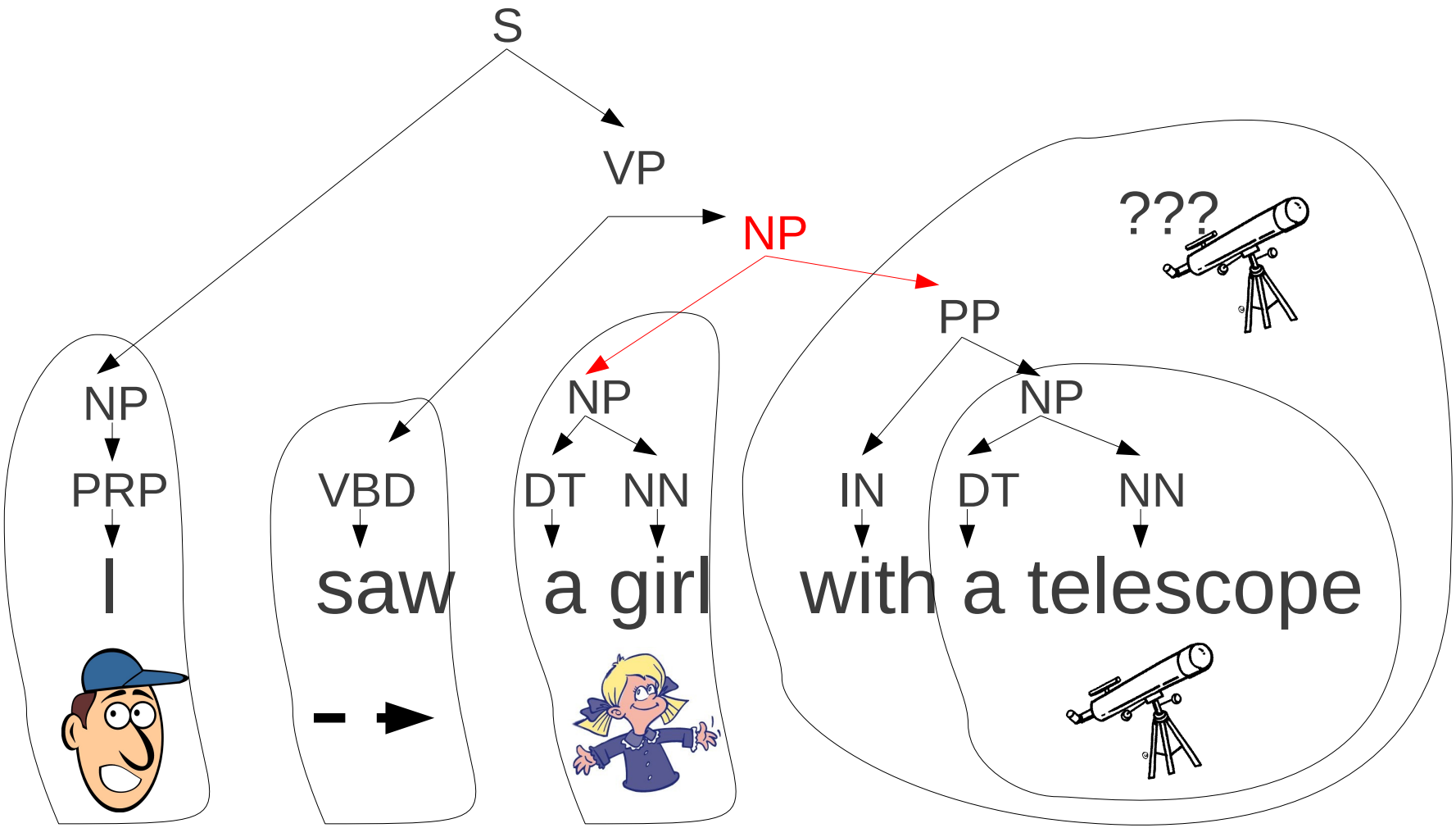
句の再帰的な構造


S
VP
PP
???
NP
PRP
I
VBD
saw
NP
DT
NN
a girl
IN
with
NP
DT
NN
a telescope

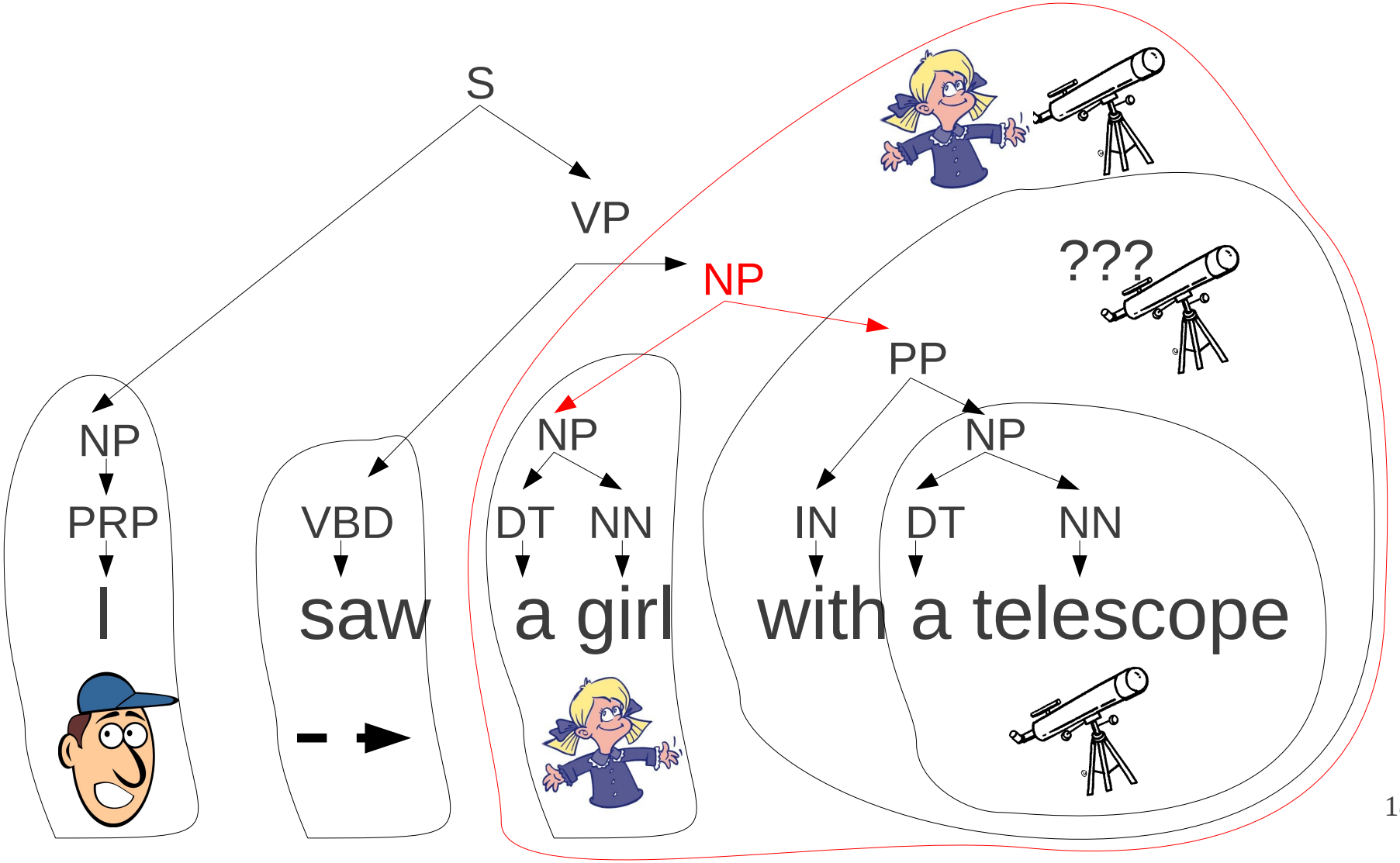
句の再帰的な構造


S
VP
PP
???
NP
PRP
I
VBD
saw
NP
DT NN
a girl
IN
with
NP
DT NN
a telescope

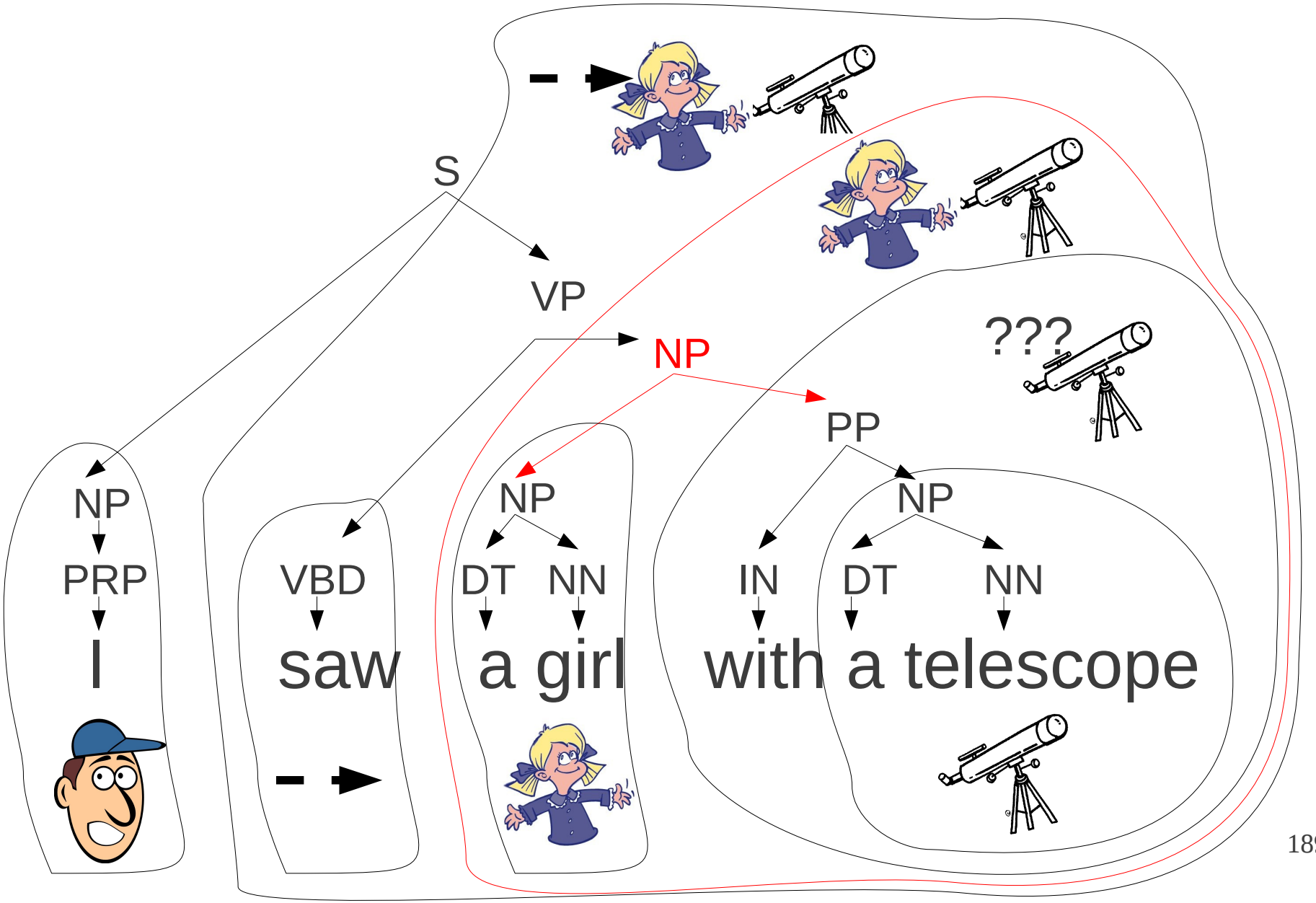
違う構造→違う解釈


S
VP
NP
PP
???
NP
PRP
VBD
NP
DT
NN
IN
NP
DT
NN
I
saw
a girl
with a telescope

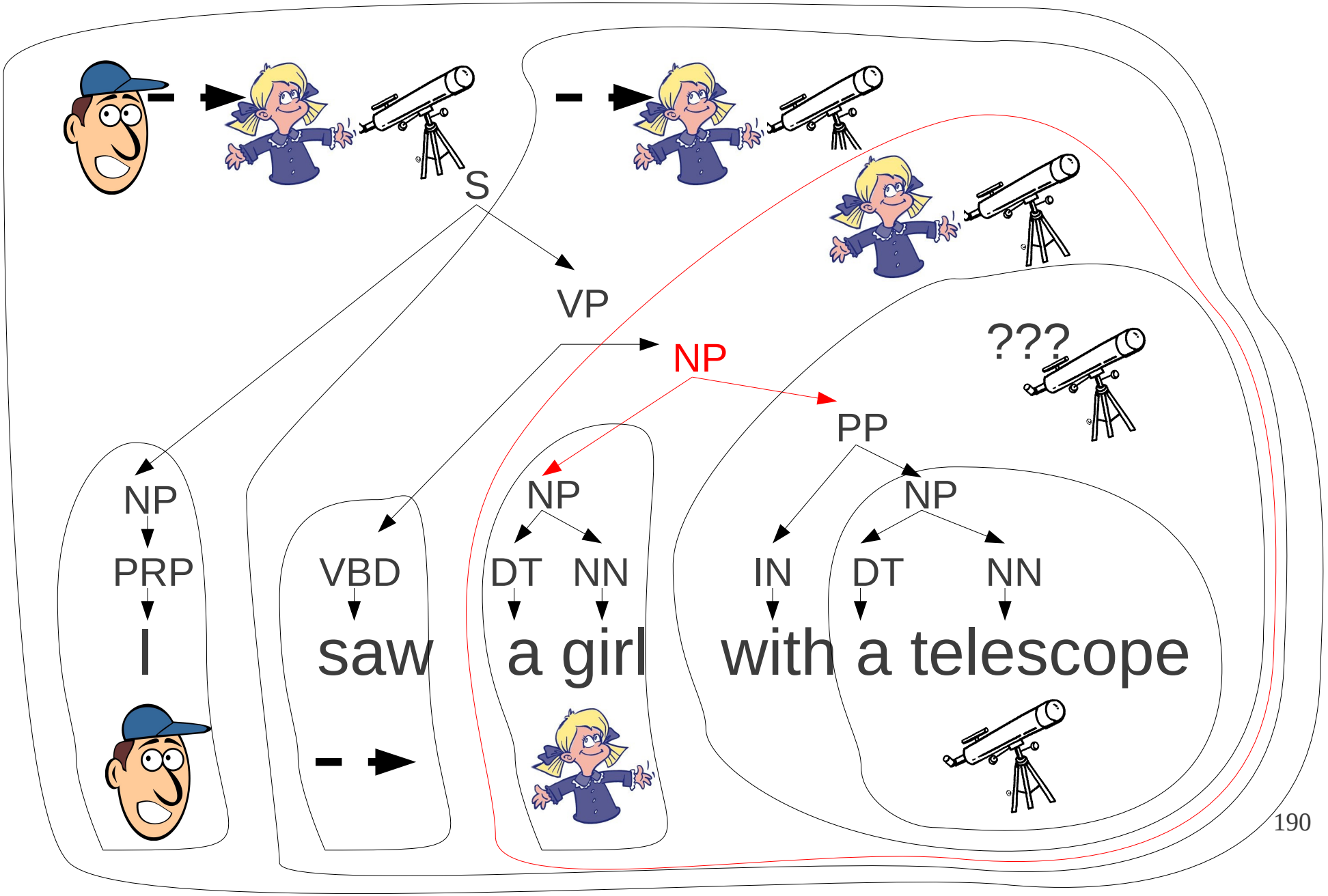
違う構造→違う解釈


S
VP
NP
PP
NP
NP
NP
PRP
VBD
DT
NN
IN
DT
NN
I
saw
a girl
with a telescope
???

違う構造→違う解釈

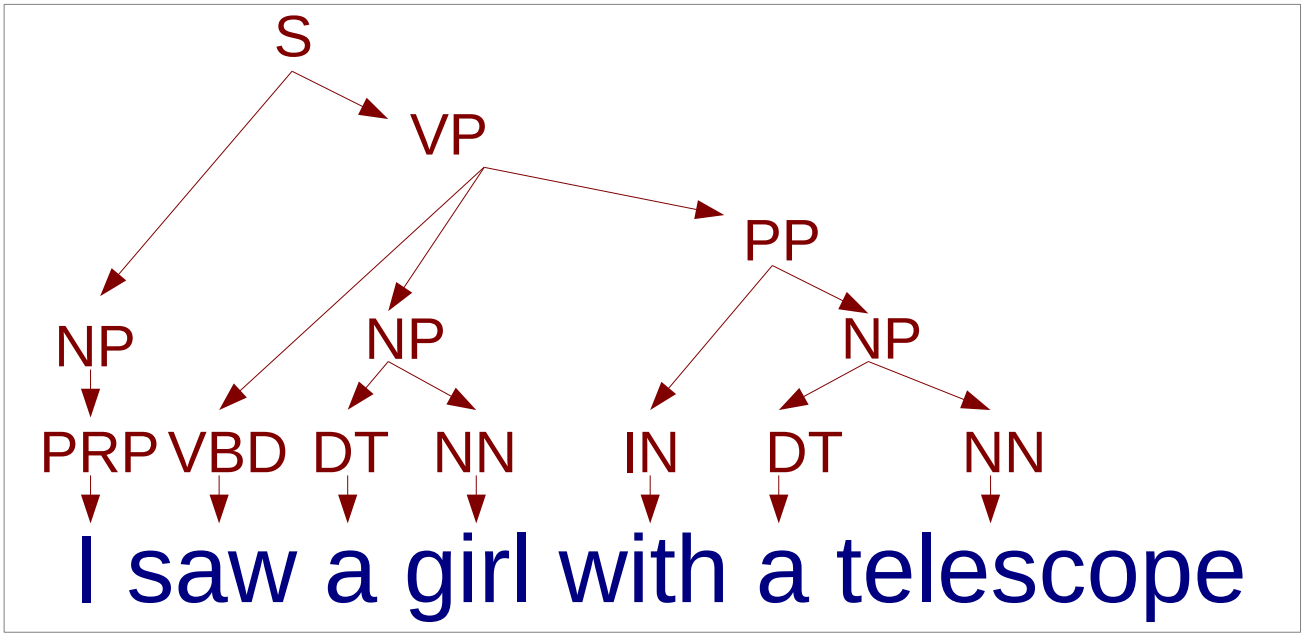

S
VP
NP
PP
NP
NP
NP
PRP
VBD
DT
NN
IN
DT
NN
I
saw
a girl
with a telescope
???

違う構造→違う解釈


S
VP
NP
PP
???
NP
PRP
I
VBD
saw
NP
DT
NN
a girl
IN
with
NP
DT
NN
a telescope

予測問題としての構文解析

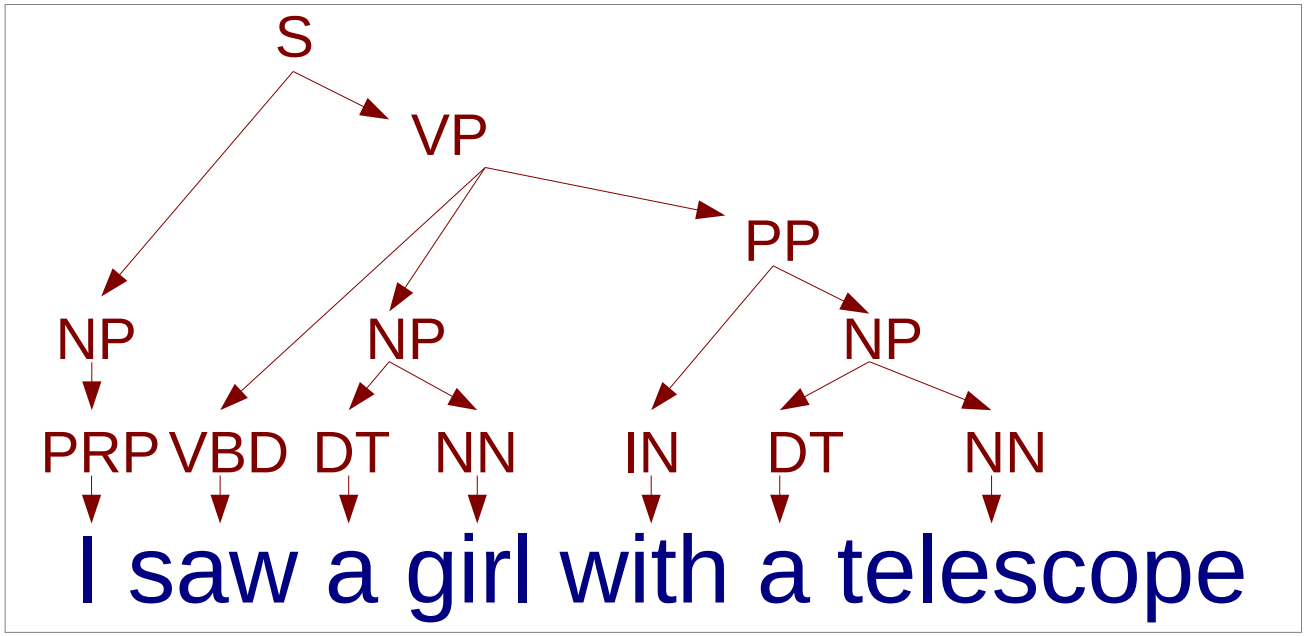
- 文 X が与えられ、構文木 Y を予測



- 「構造予測」の問題（品詞推定、単語分割と同様）

構文解析の確率モデル

- 文 X が与えられ、事後確率の最も高い構文木 Y を予測



$$\underset{Y}{\operatorname{argmax}} P(Y|X)$$

生成モデル

- 構文木 Y と文 X が同時に確率モデルにより生成されたとする

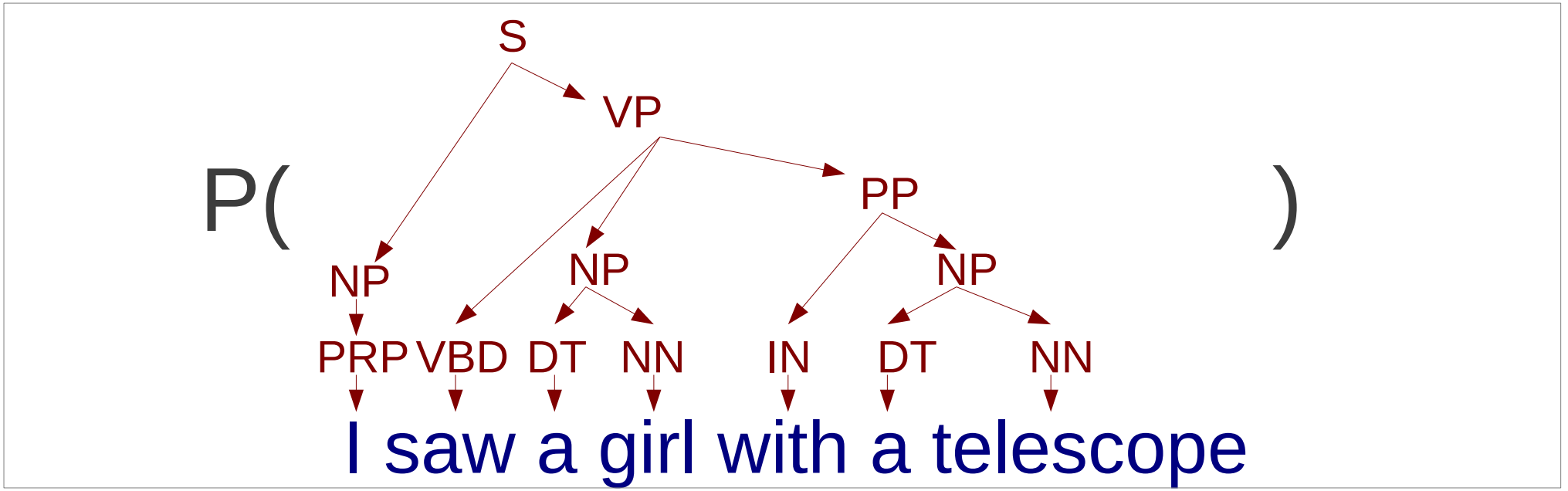
$$P(Y, X)$$

- X を固定すると、同時確率が最も高い Y は事後確率も最も高い

$$\underset{Y}{\operatorname{argmax}}\, P(Y|X) = \underset{Y}{\operatorname{argmax}}\, P(Y, X)$$

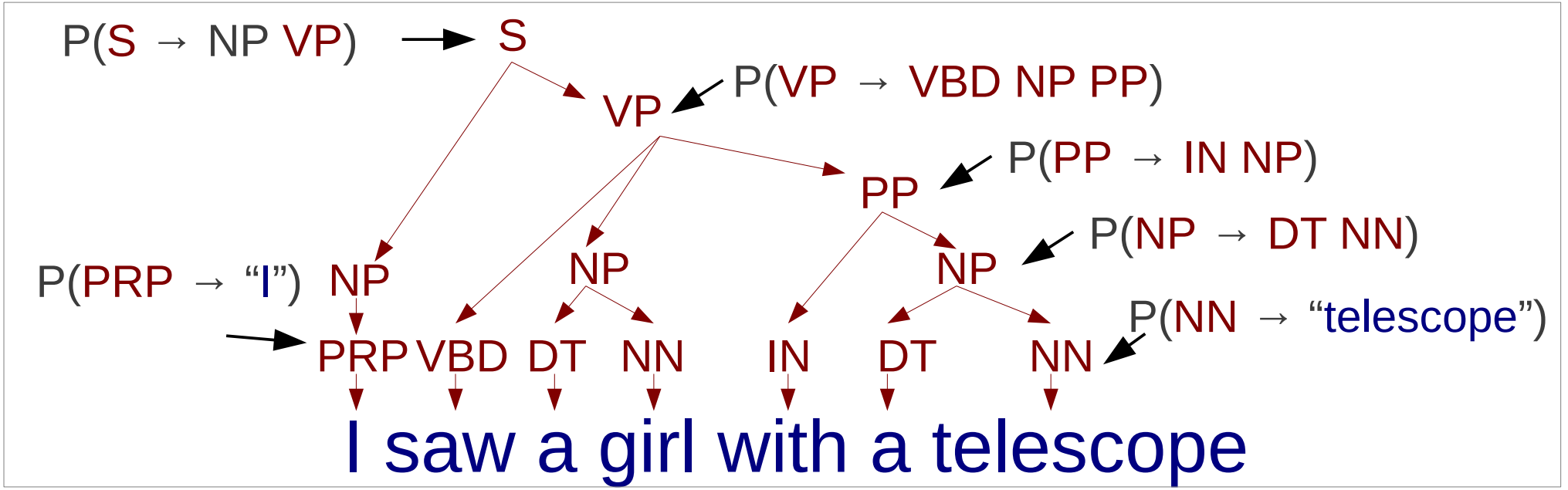
確率的文脈自由文法 (PCFG)

- 構文木の同時確率をどう定義するか？



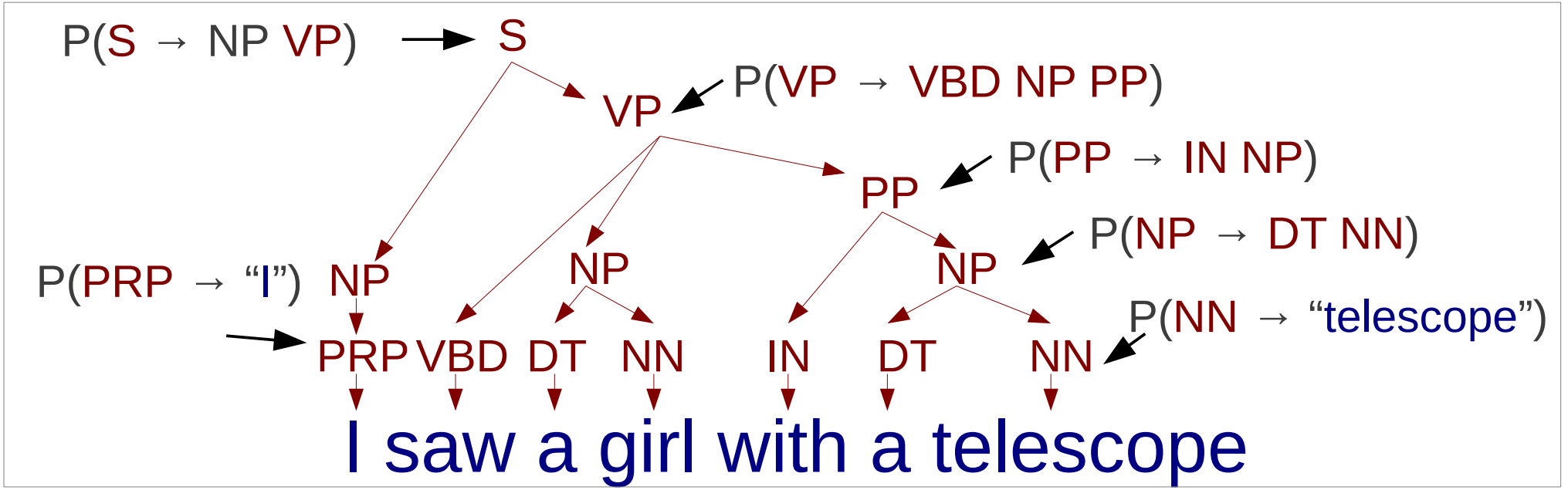
確率的文脈自由文法 (PCFG)

- PCFG ：各ノードの確率を個別に定義



確率的文脈自由文法 (PCFG)

- PCFG ：各ノードの確率を個別に定義



- 構文木の確率はノードの確率の積

P(S → NP VP) * P(NP → PRP) * P(PRP → “I”)
* P(VP → VBD NP PP) * P(VBD → “saw”) * P(NP → DT NN)
* P(DT → “a”) * P(NN → “girl”) * P(PP → IN NP) * P(IN → “with”)
* P(NP → DT NN) * P(DT → “a”) * P(NN → “telescope”)

確率的構文解析

- 構文解析は確率が最大の構文木を探索すること

$$\underset{Y}{\operatorname{argmax}}\, P\left(Y, X\right)$$

- ビタビアルゴリズムは利用可能か？

確率的構文解析

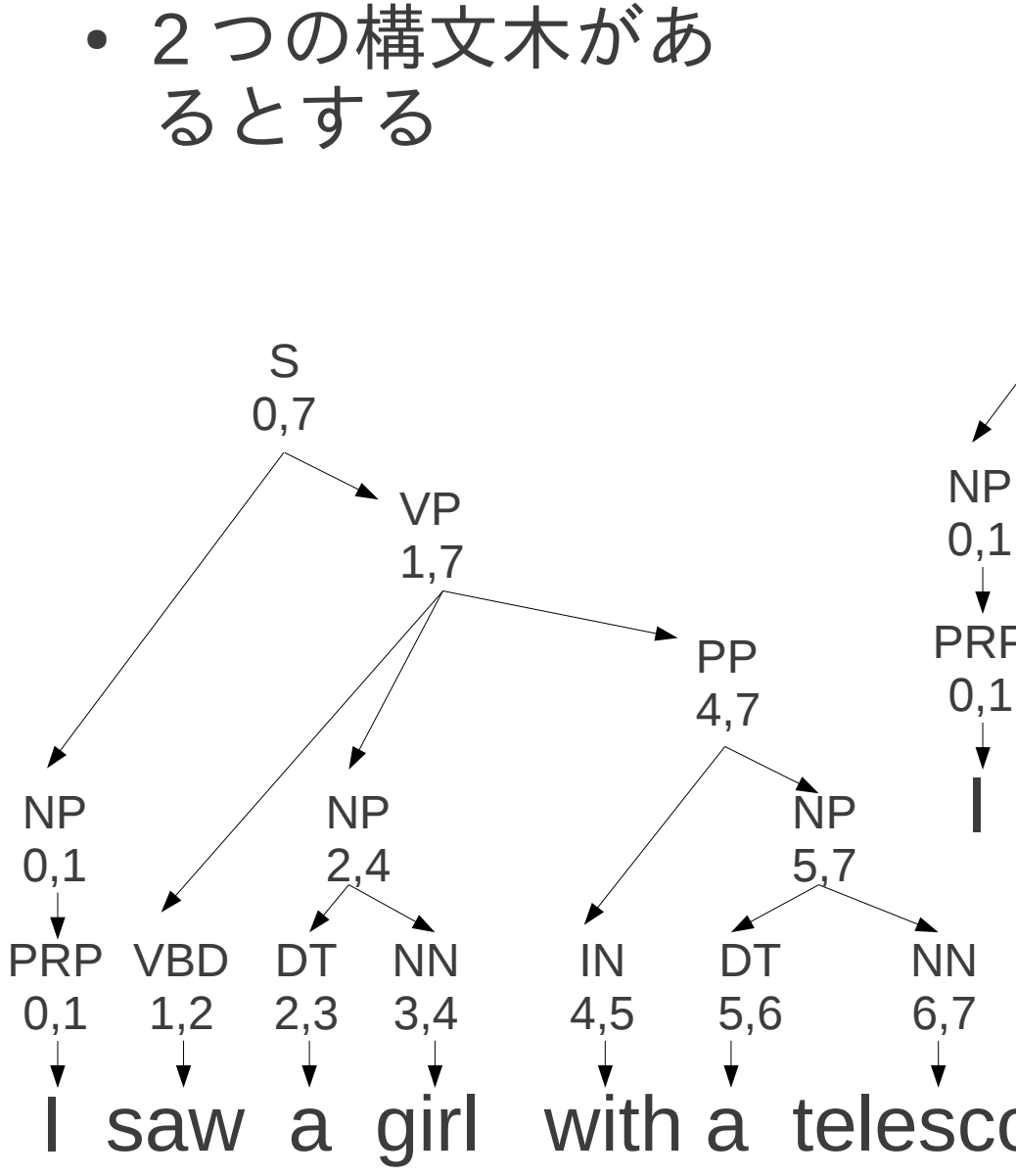
- 構文解析は確率が最大の構文木を探索すること

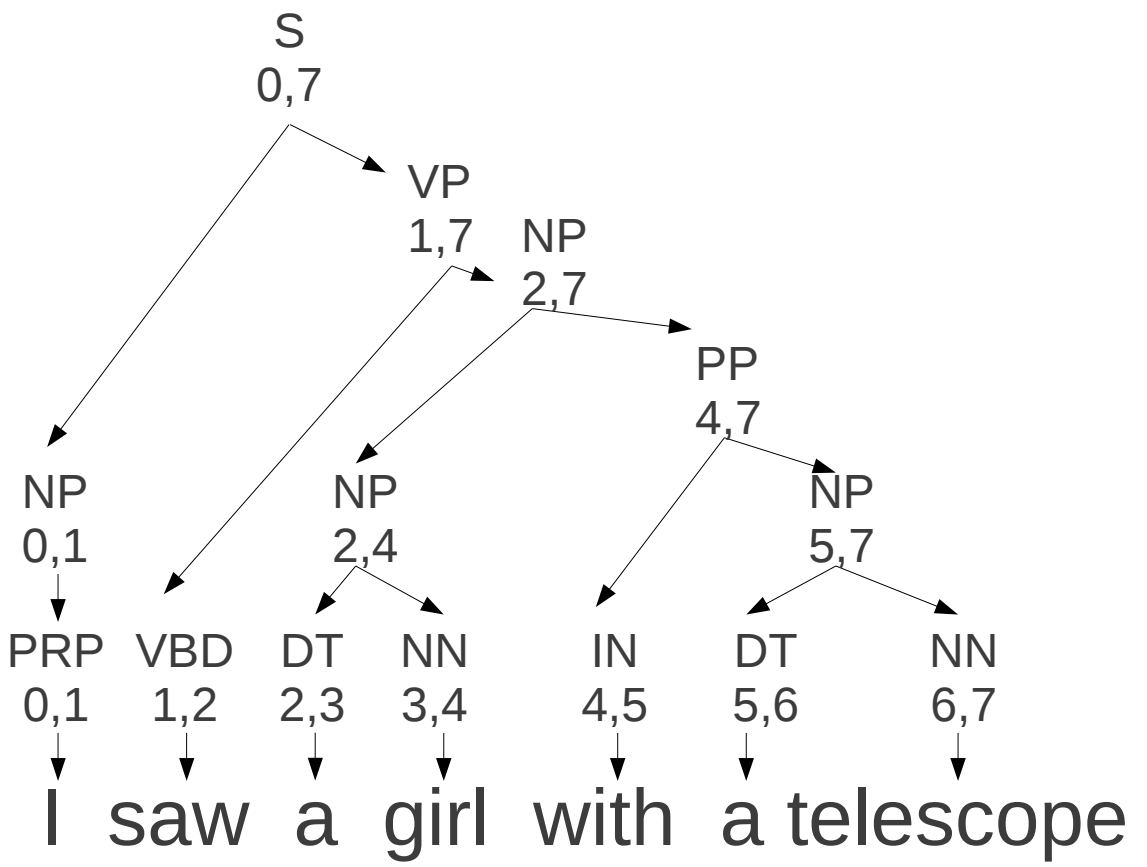
$$\underset{Y}{\operatorname{argmax}} P(Y, X)$$

- ビタビアルゴリズムは利用可能か？
  - 答え：いいえ！
  - 理由：構文木の候補はグラフで表せず超グラフとなる

超グラフとは？

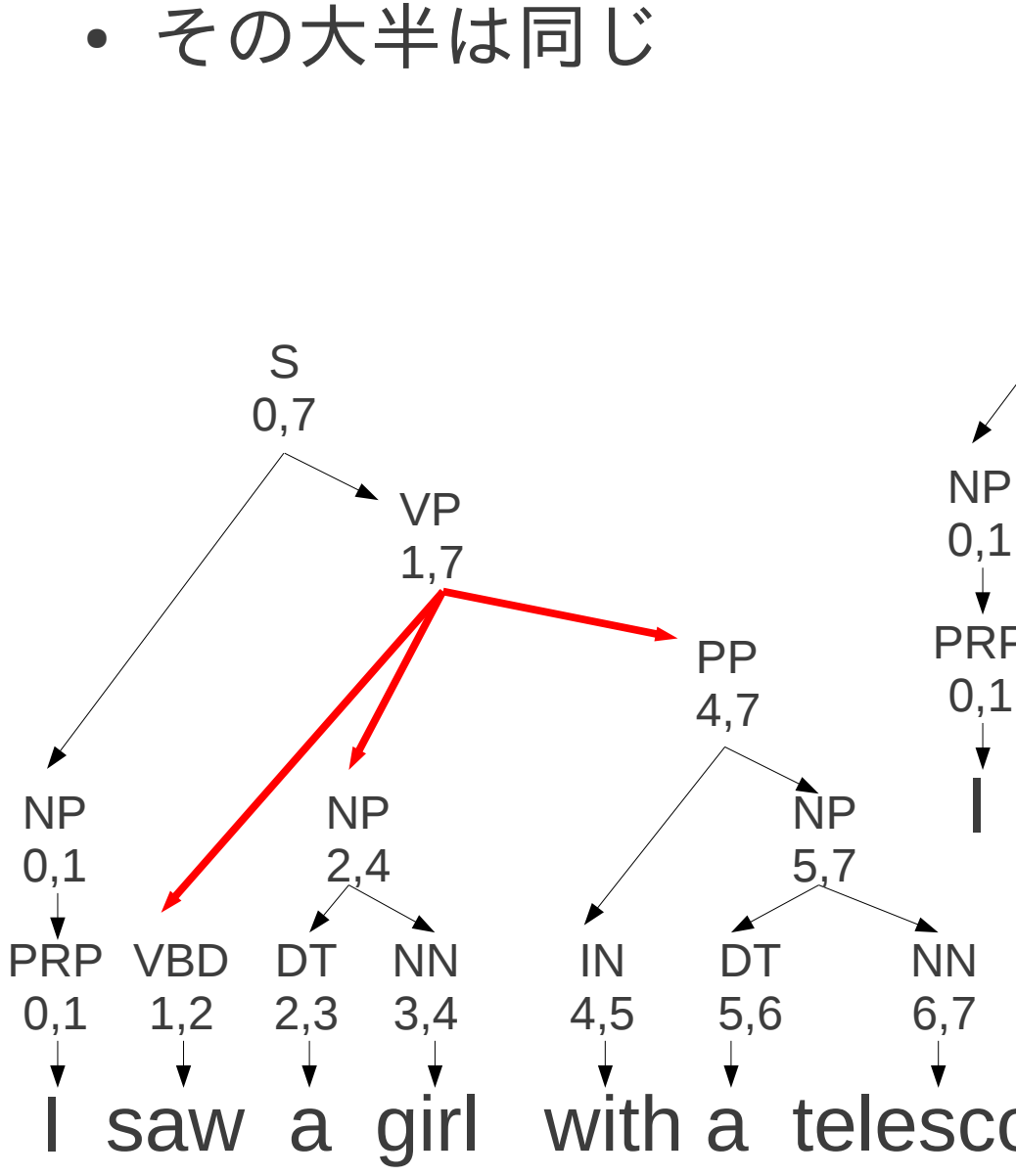
- 2つの構文木があるとする

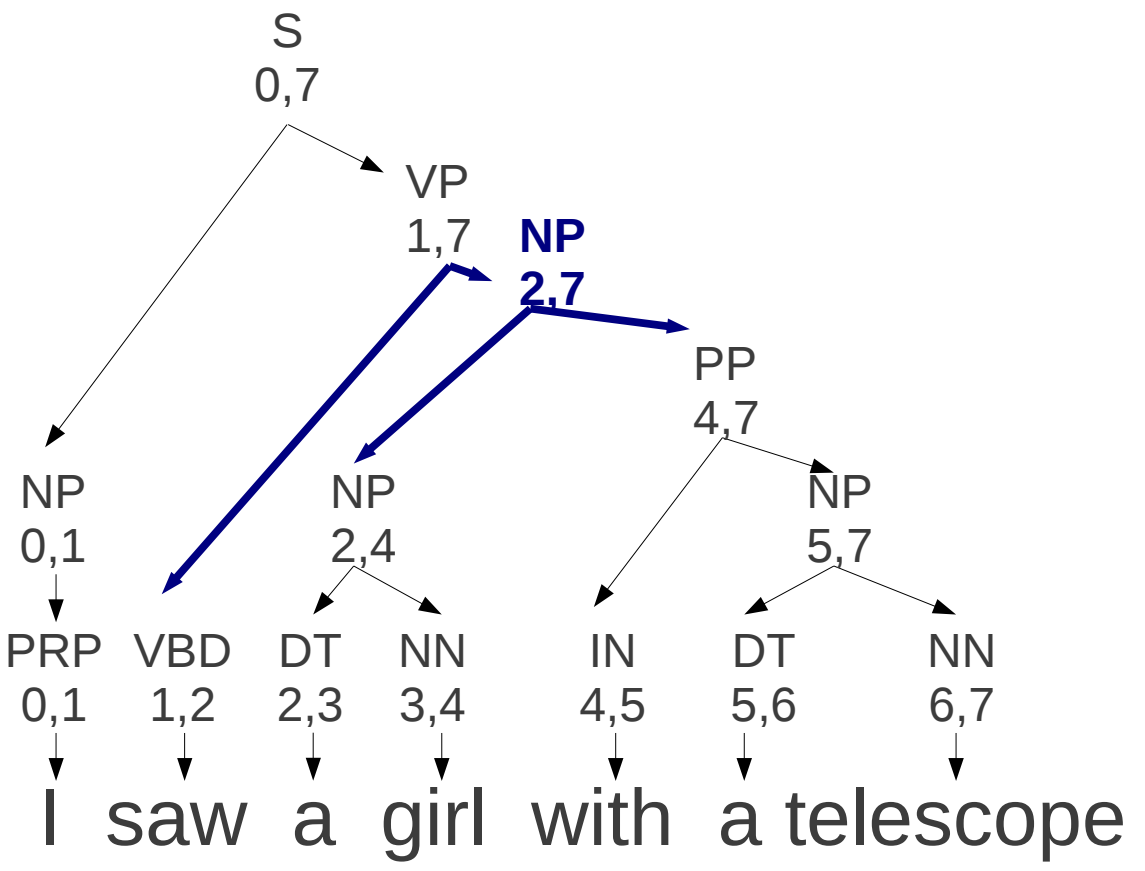




超グラフとは？

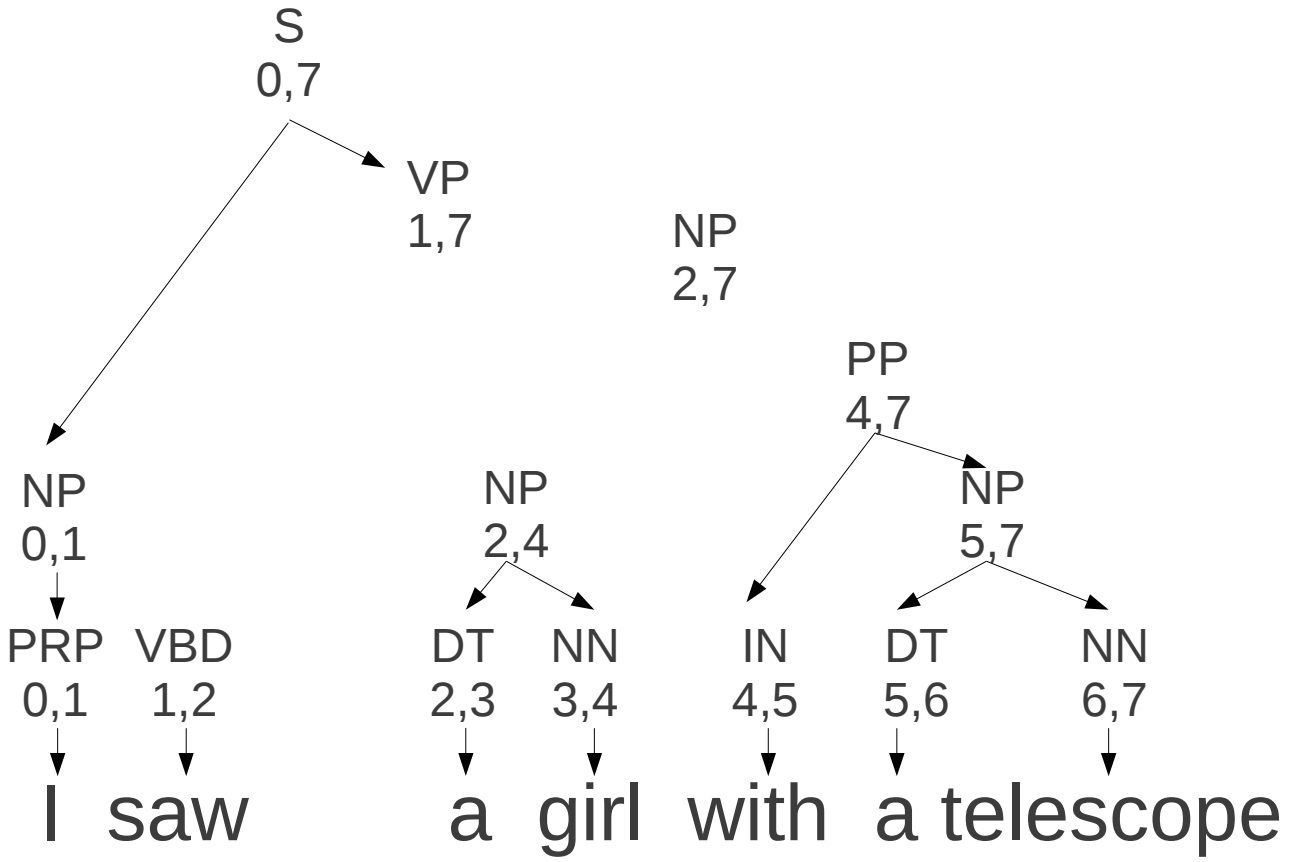
- その大半は同じ





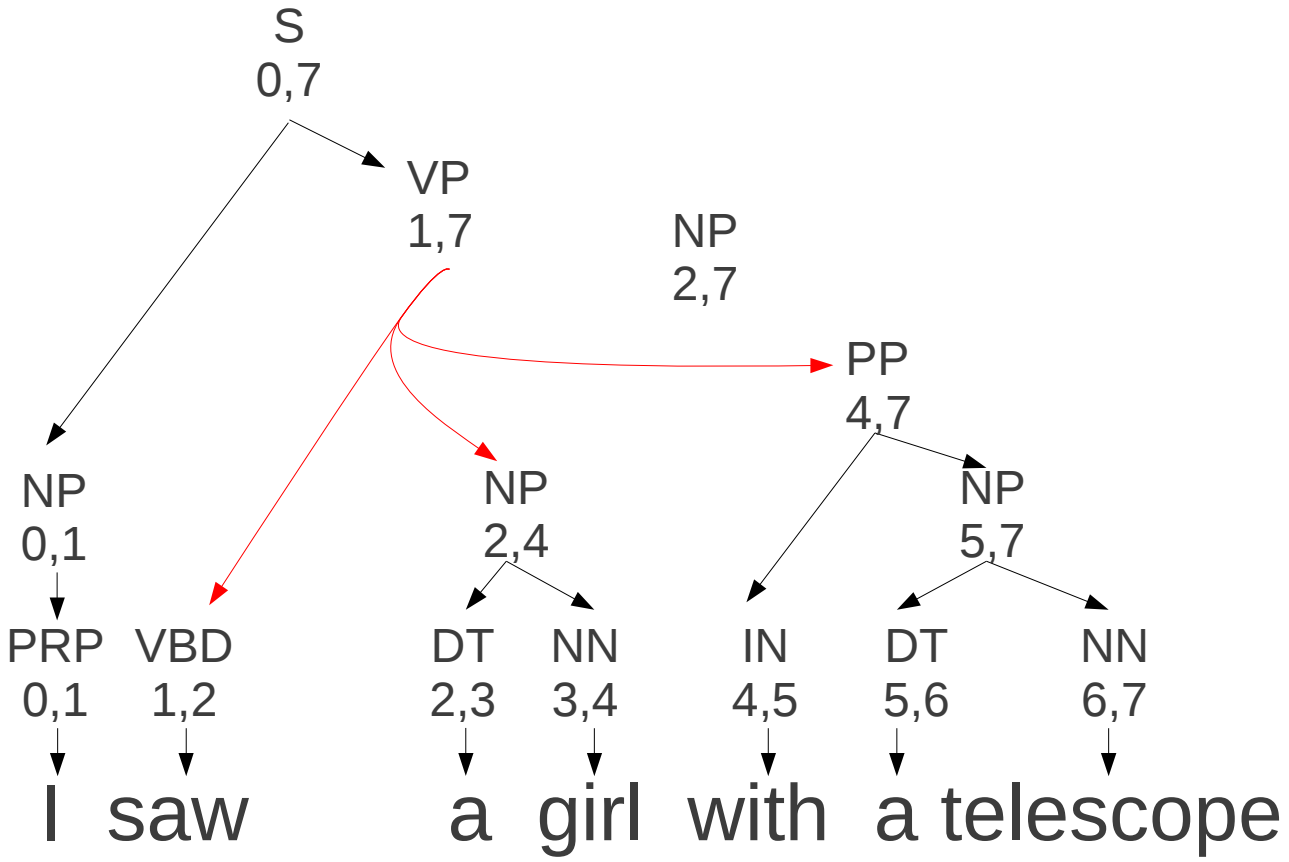
超グラフとは？

- 両方に現れるエッジだけを残すと：



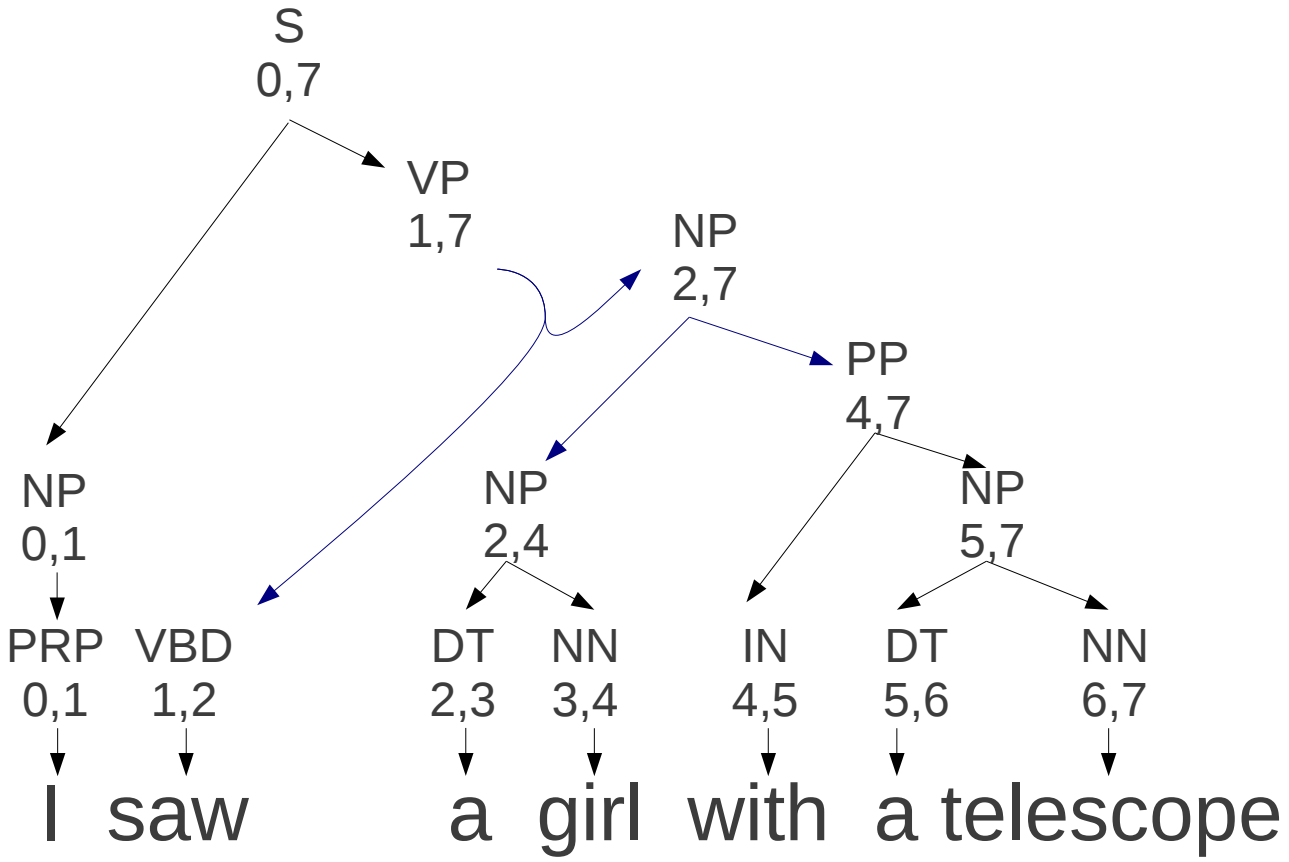
超グラフとは？

- 1 番目の構文木のみに存在するエッジを追加：



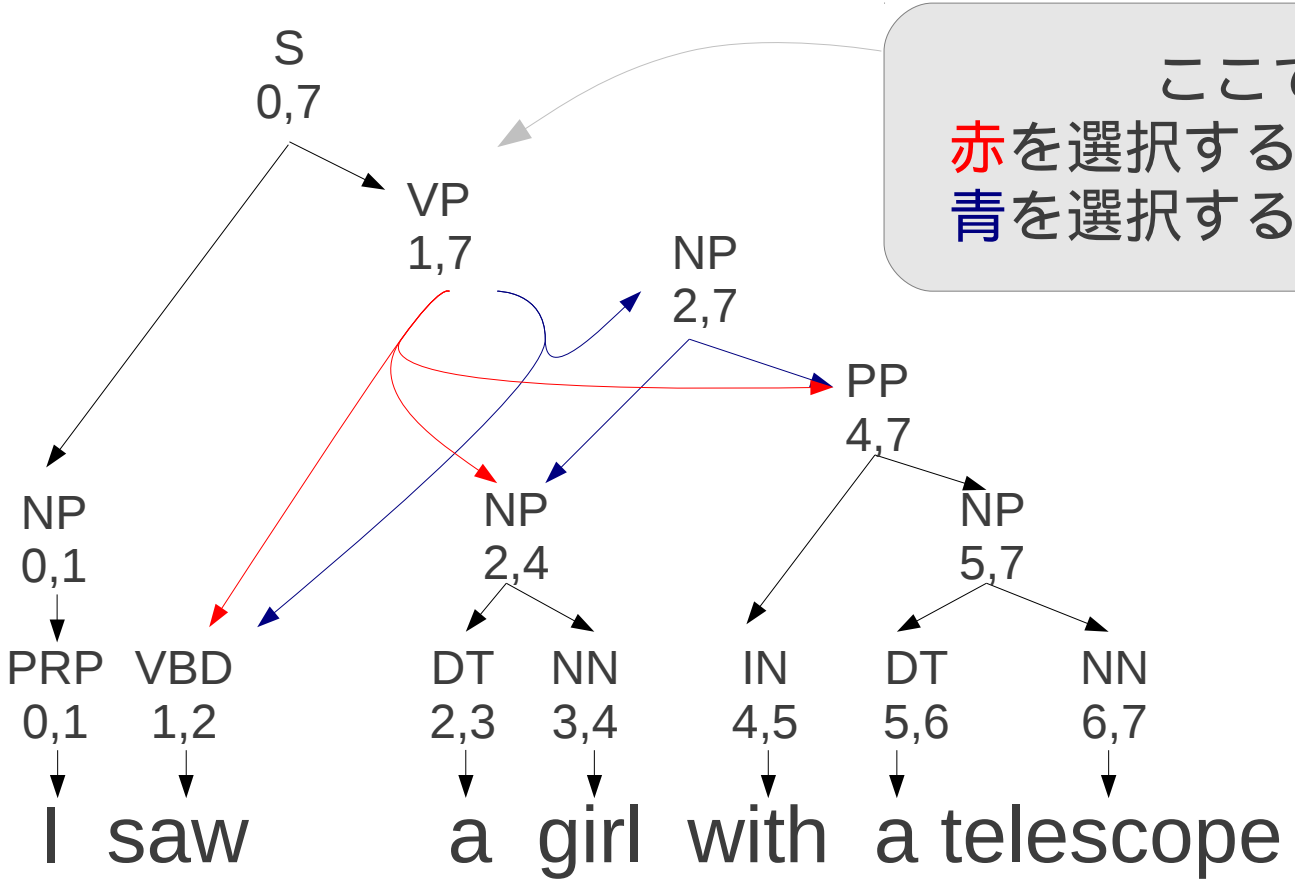
超グラフとは？

- 2 番目の構文木のみに存在するエッジを追加：



超グラフとは？

- 両方の構文木のみに存在するエッジを追加：



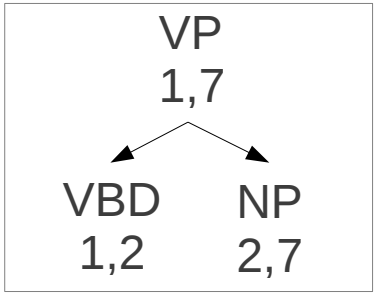
ここで２択：
赤を選択すると１番目構文木
青を選択すると２番目構文木

なぜ「超」グラフ？

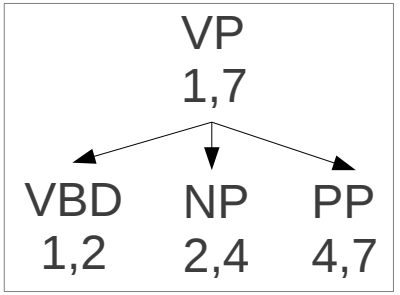
- エッジの「次数」は子の数

次数 1

次数 2

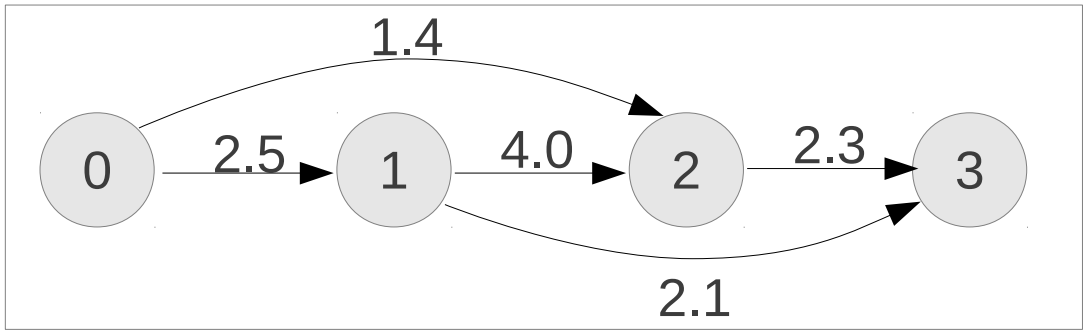


次数 3



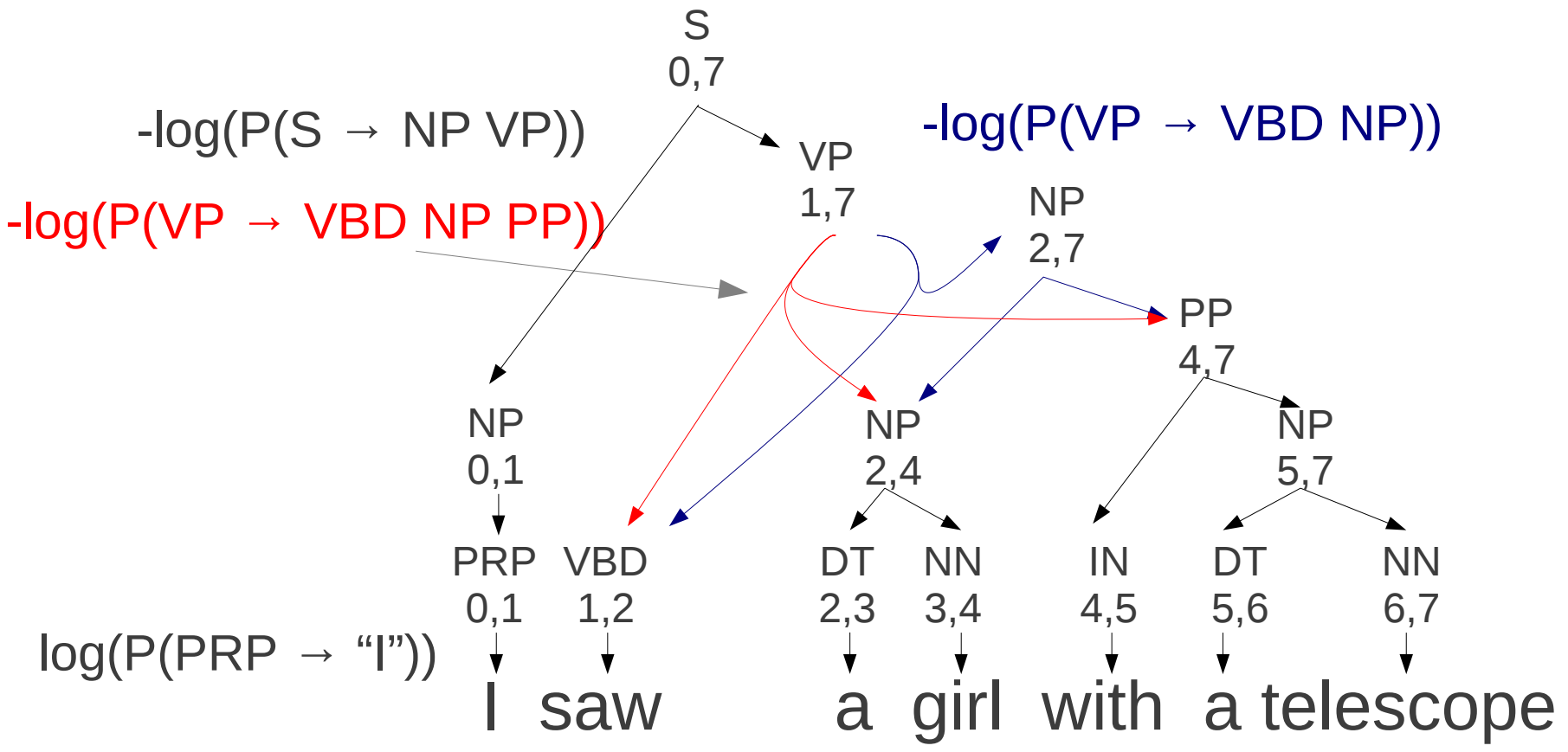
- 超グラフの次数はエッジの次数の最大値
- グラフは次数 1 の超グラフ！

例 →



重み付き超グラフ

- グラフと同じく：
  - 超グラフのエッジに重みを付与
  - 負の対数確率（ビタビアルゴリズムと同等の理由）

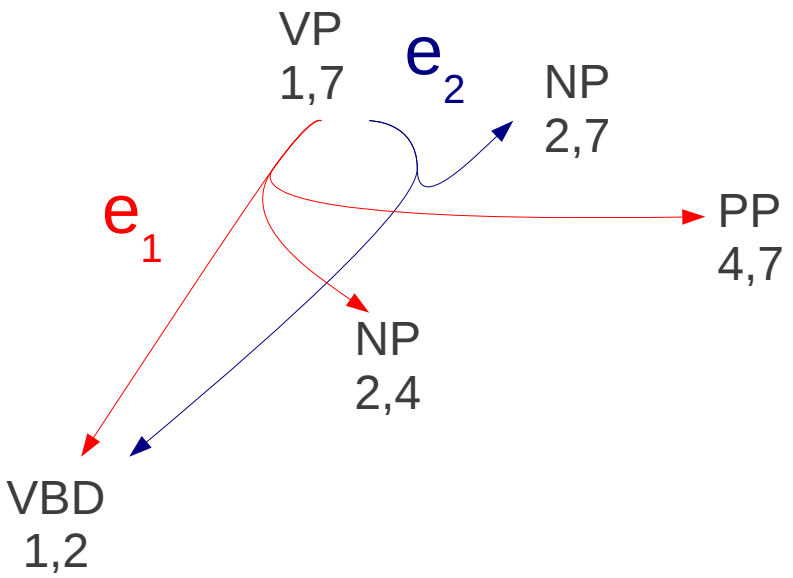


超グラフの探索法

- 構文解析＝超グラフの最もスコアの小さい木を探索
- グラフではビタビアルゴリズムを利用
  - 前向きステップ: 各ノードまでの最短経路を計算
  - 後ろ向き: 最短経路を復元
- 超グラフもほとんど同等のアルゴリズム
  - 内ステップ: 各ノードの最小部分木のスコアを計算
  - 外ステップ: スコア最小の木を復元

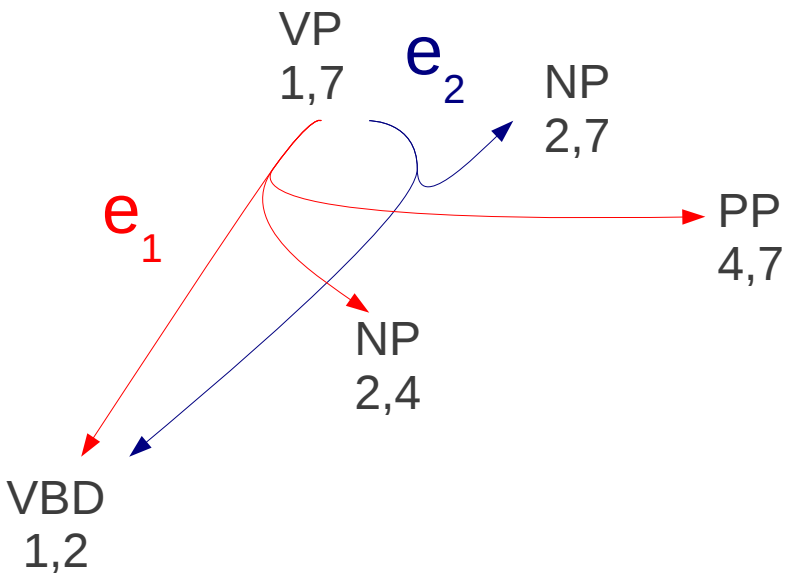
ノードの内ステップ

- VP1,7 の最小スコアを計算



ノードの内ステップ

- VP1,7 の最小スコアを計算

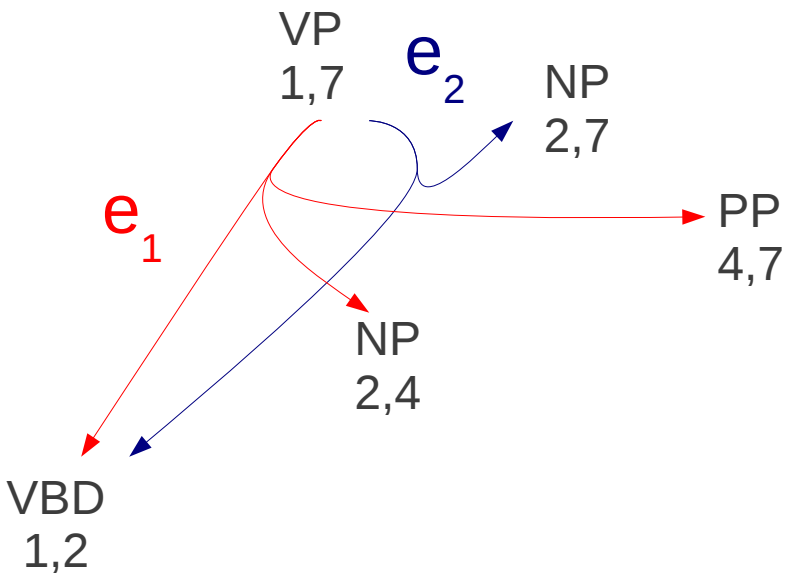


score(e_1) =
-log(P(VP → VBD NP PP)) +
best_score[VBD1,2] +
best_score[NP2,4] +
best_score[NP2,7]

score(e_2) =
-log(P(VP → VBD NP)) +
best_score[VBD1,2] +
best_score[VBD2,7]

ノードの内ステップ

- VP1,7 の最小スコアを計算



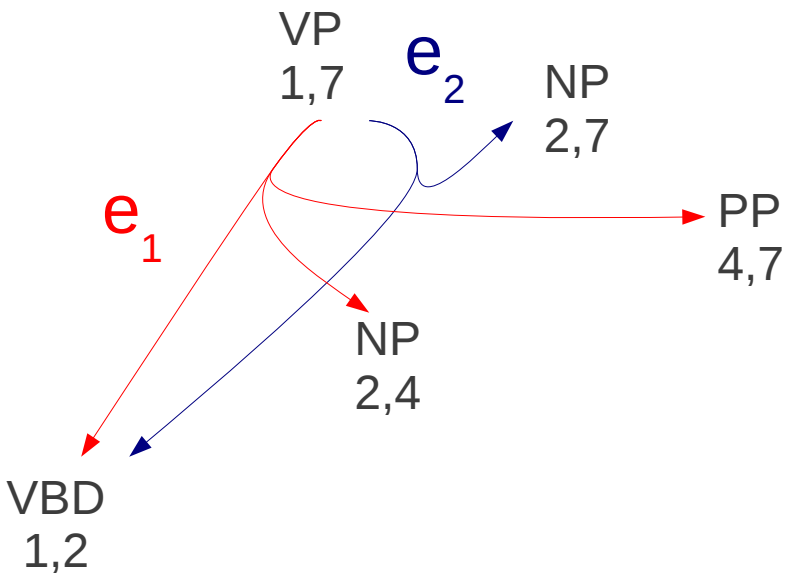
score(e_1) =
  -log(P(VP → VBD NP PP)) +
  best_score[VBD1,2] +
  best_score[NP2,4] +
  best_score[NP2,7]

score(e_2) =
  -log(P(VP → VBD NP)) +
  best_score[VBD1,2] +
  best_score[VBD2,7]

best_edge[VB1,7] = $\text{argmin}_{e1,e2}$ score

ノードの内ステップ

- VP1,7 の最小スコアを計算



score(e_1) =
  -log(P(VP → VBD NP PP)) +
  best_score[VBD1,2] +
  best_score[NP2,4] +
  best_score[NP2,7]

score(e_2) =
  -log(P(VP → VBD NP)) +
  best_score[VBD1,2] +
  best_score[VBD2,7]

best_edge[VB1,7] = $\text{argmin}_{e1,e2}$ score
best_score[VB1,7] =
  score(best_edge[VB1,7])

文法からの超グラフ構築

- 超グラフは解けるが、構文解析で与えられるのは

文法	文
P(S → NP VP) = 0.8 P(S → PRP VP) = 0.2 P(VP → VBD NP PP) = 0.6 P(VP → VBD NP)= 0.4 P(NP → DT NN) = 0.5 P(NP → NN) = 0.5 P(PRP → “I”) = 0.4 P(VBD → “saw”) = 0.05 P(DT → “a”) = 0.6 …	I saw a girl with a telescope

- 解くための超グラフをどうやって構築するか？

CKY アルゴリズム

- CKY(Cocke-Kasami-Younger) アルゴリズムは文法に基づいてハイパーグラフを構築して解く
- 文法はチョムスキー標準形 (CNF)
  - ルールの右側は非終端記号 2 つもしくは終端記号 1 つ

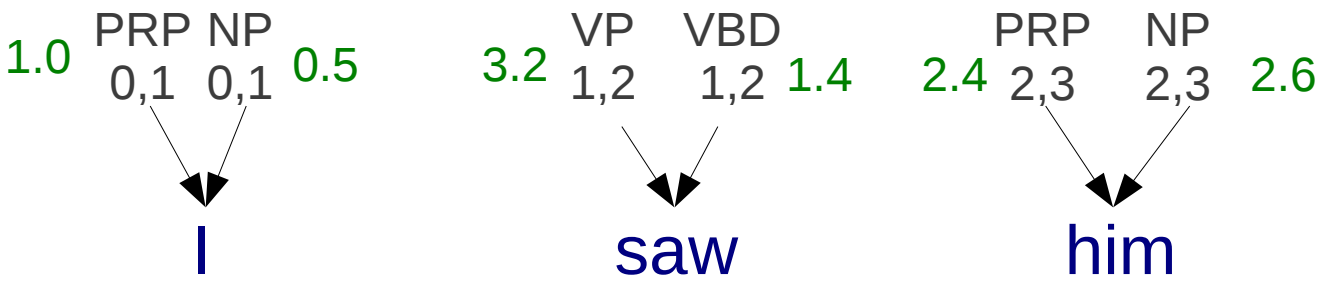
OK	OK	Not OK!
S → NP VP	PRP → “I”	VP → VBD NP PP
S → PRP VP	VBD → “saw”	NP → NN
VP → VBD NP	DT → “a”	NP → PRP

  - この条件を満たさないルールは変更可能

VP → VBD NP PP	➡	VP → VBD NP_PP NP_PP → NP PP
NP → PRP + PRP → “I”	➡	NP → “I”

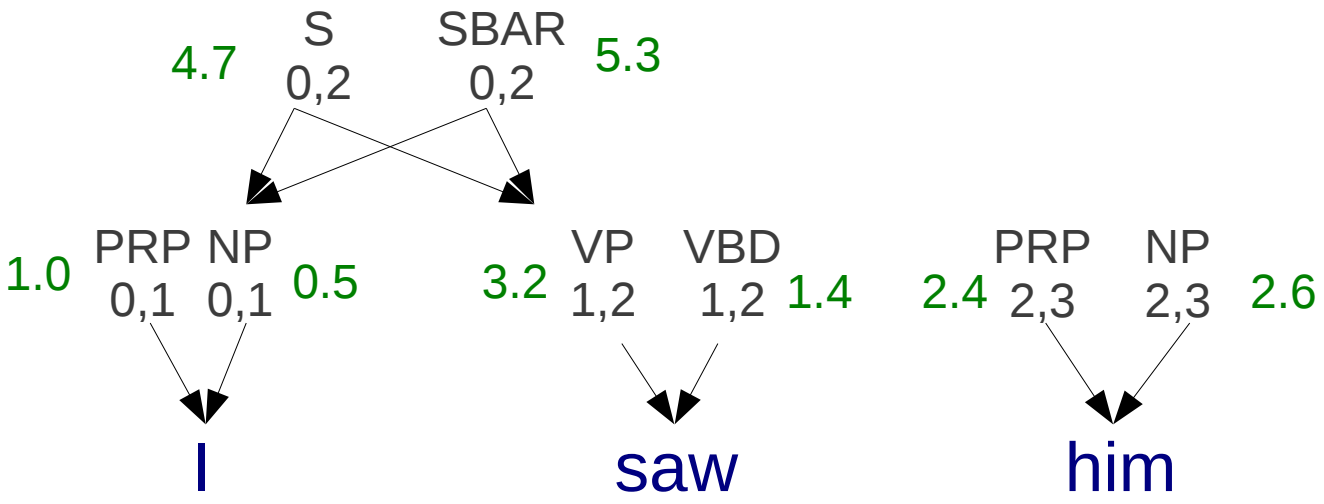
CKY アルゴリズム

- まずは終端記号のルールをスコア付きで展開



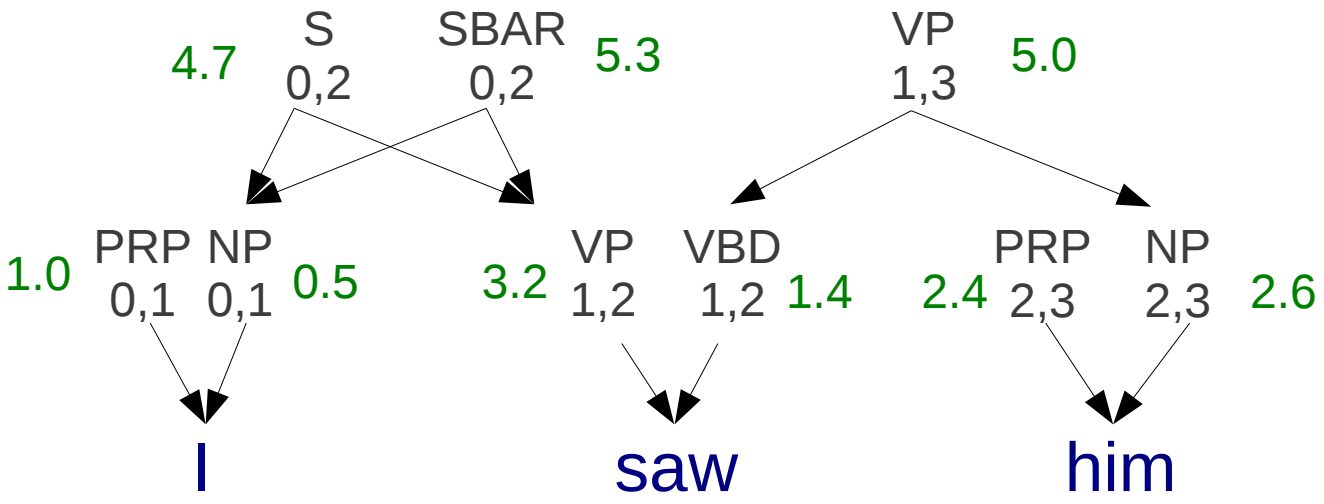
CKY アルゴリズム

- 0,2 のノードを全て展開



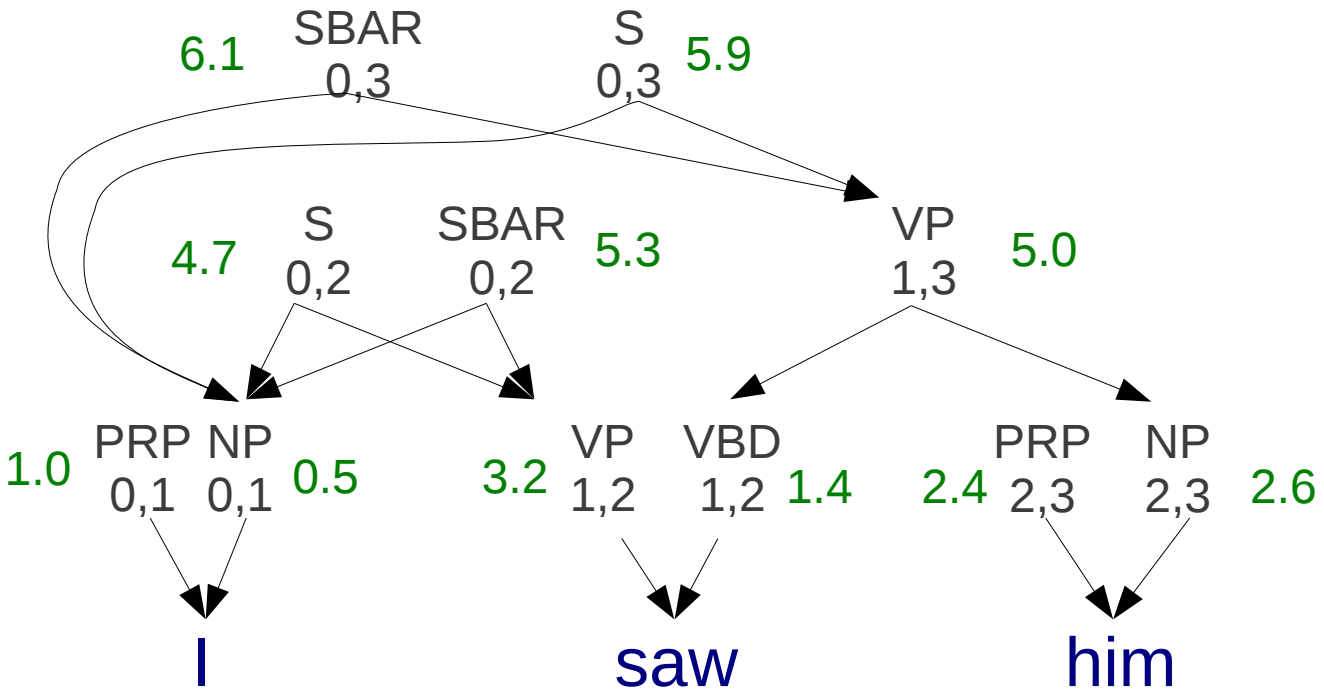
CKY アルゴリズム

- 1,3 のノードを全て展開



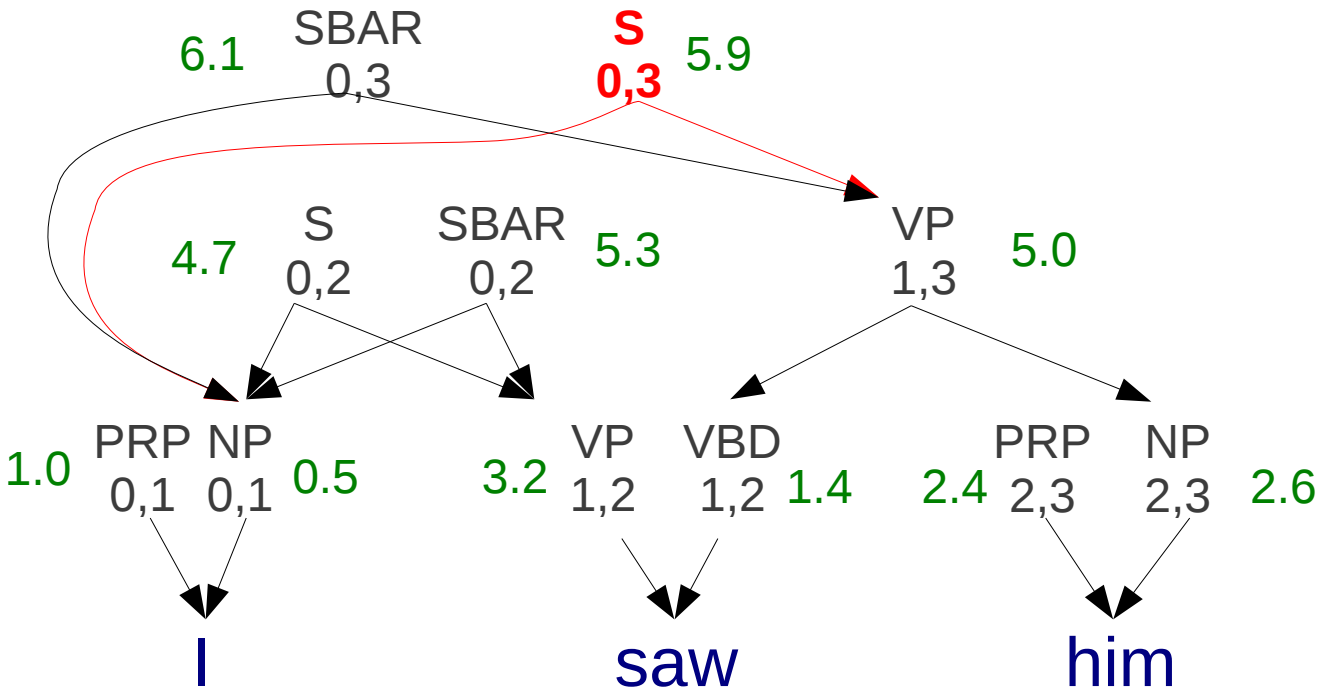
CKY アルゴリズム

- 0,3 のノードを全て展開



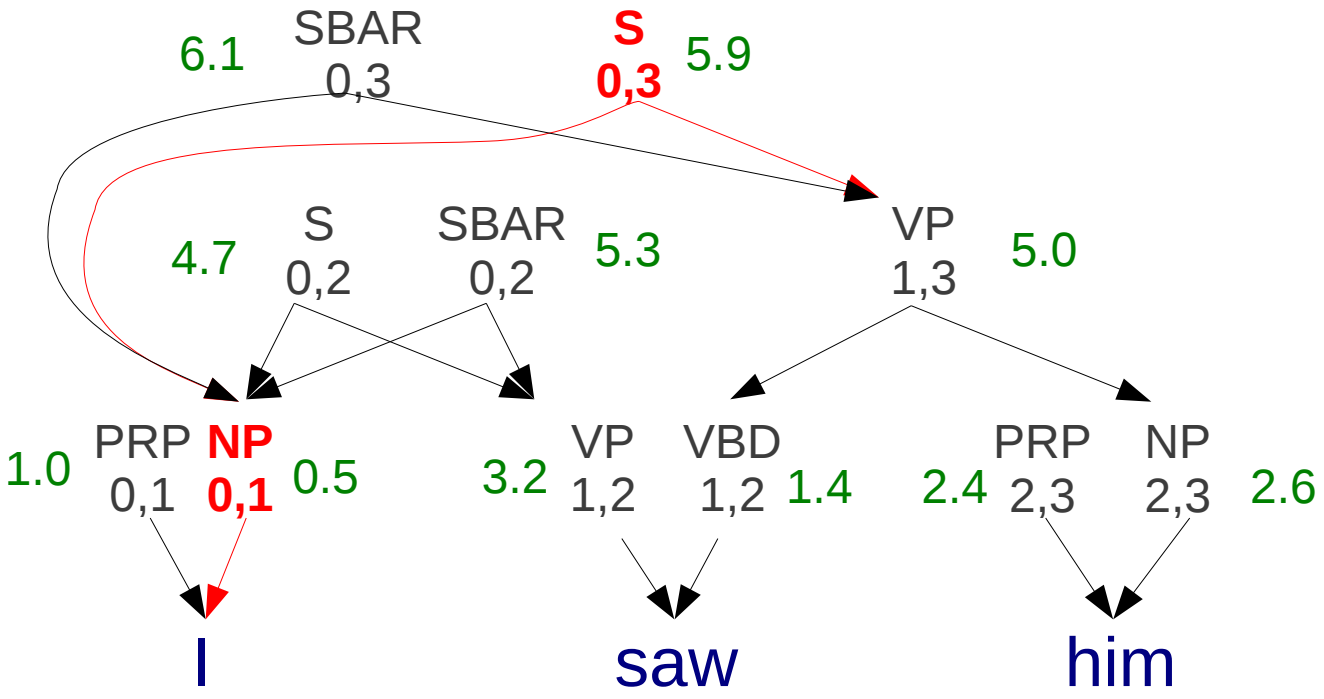
CKY アルゴリズム

- 文を全てカバーする「S」ノードを見つけて、エッジを展開



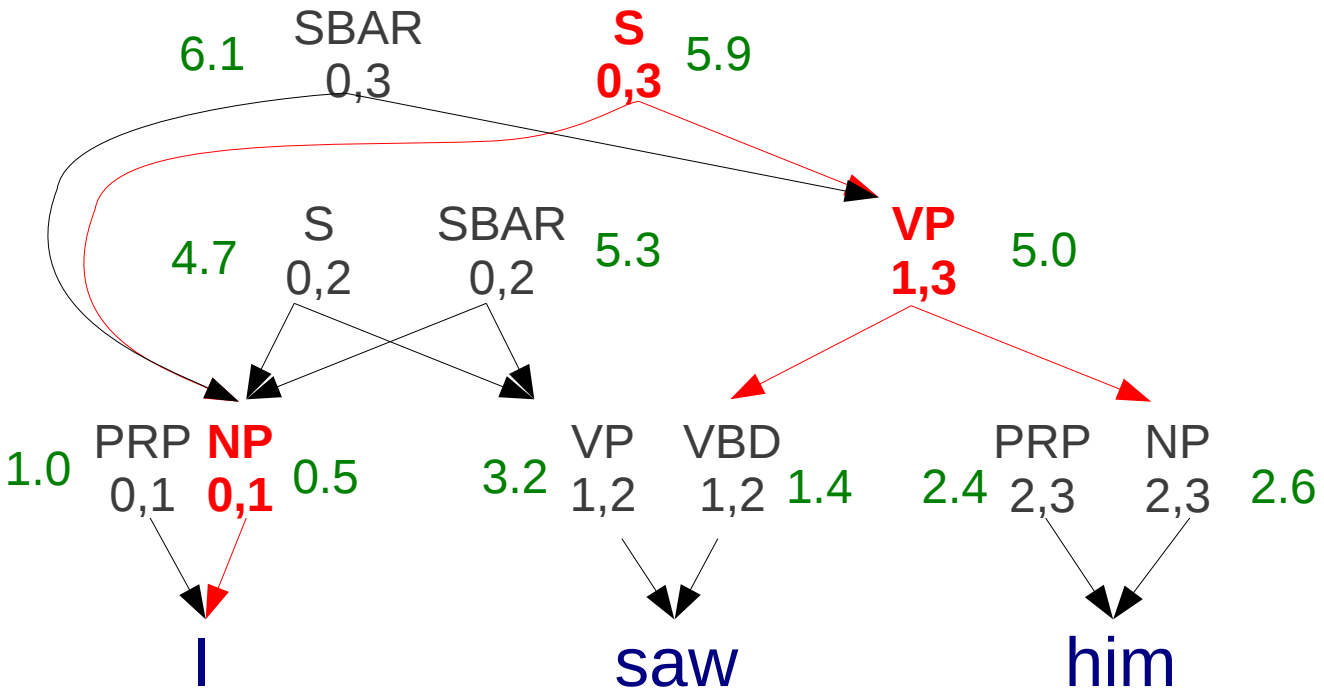
CKY アルゴリズム

- 左の子、右の子を再帰的に展開



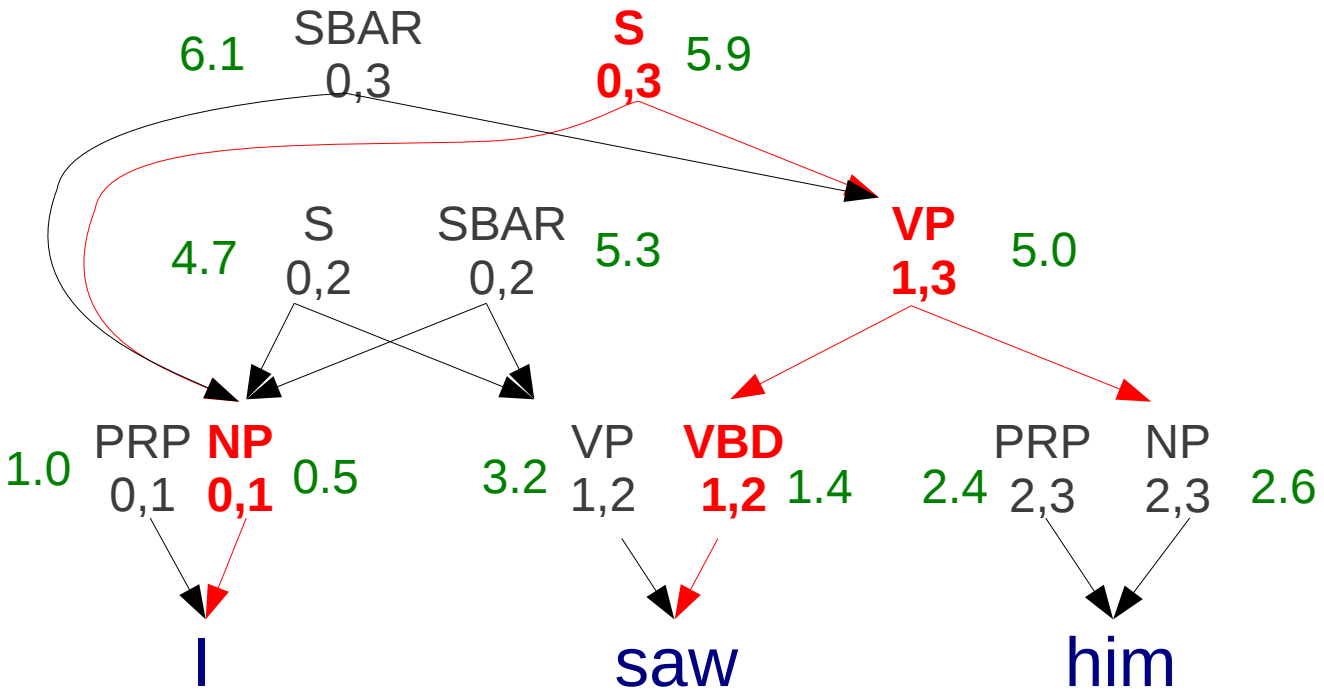
CKY アルゴリズム

- 左の子、右の子を再帰的に展開



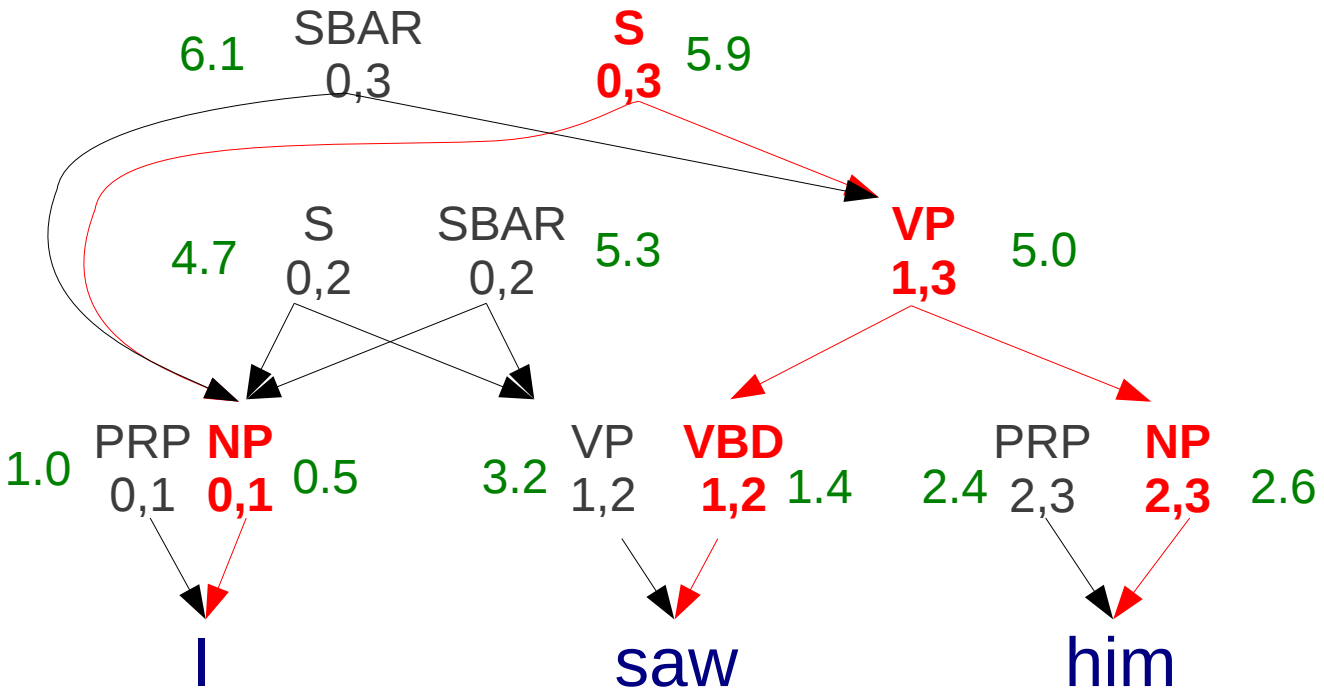
CKY アルゴリズム

- 左の子、右の子を再帰的に展開



CKY アルゴリズム

- 左の子、右の子を再帰的に展開



構文解析ツール

- 英語の句構造解析器：
  - Stanford Parser: 頑健に解析が可能
  - Berkeley Parser: Stanford より高性能
- 日本語は係り受け解析が主流：
  - EDA: 単語ベース係り受け解析、 KyTea と利用
  - CaboCha: 文節ベース係り受け解析 MeCab と利用
  - 係り受け解析結果は変更可能

演習：英語の構文解析

- 構文解析する例文をファイルに書き込む

```
$ echo "I saw a girl with a telescope." > mydata/to-parse.en
```

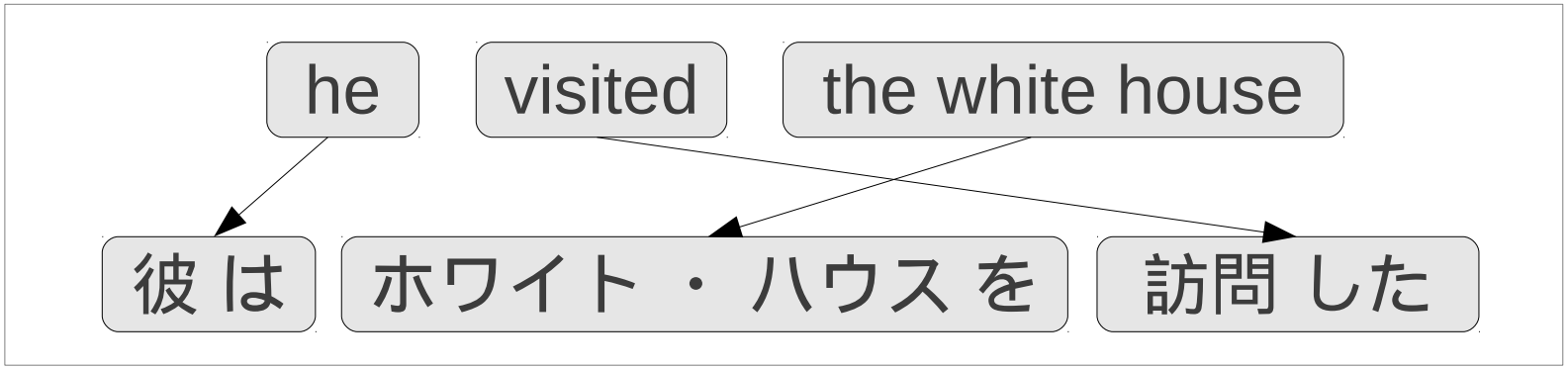
- Stanford Parser で英語の構文解析を行い、表示する

```
$ usr/local/stanford-parser-2012-11-12/lexparser.sh
                        mydata/to-parse.en > mydata/parsed.en
$ head mydata/parsed.en
(ROOT
  (S
    (NP (PRP I))
    (VP (VBD saw)
      (NP (DT a) (NN girl))
      (PP (IN with)
        (NP (DT a) (NN telescope))))
    (. .)))
```

階層的フレーズベース翻訳

階層的フレーズベース翻訳 (Hiero) [Chiang 07]

- フレーズベース翻訳は連続した単語列の変換



- しかし、連続しない良質なパターンも数多くある

例：

X_1 visited X_2 → X_1 は X_2 を 訪問 した

X_1 that was X_2 → X_2 で あった X_1

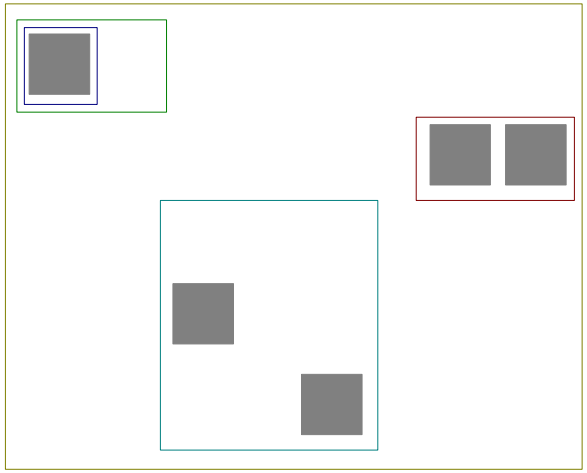
- これを利用するのは階層的フレーズベース

フレーズベースとの差

- 学習の変更：不連続なフレーズも抽出
- 訳出の変更：構文解析と類似した訳出アルゴリズム

不連続なフレーズの抽出

ホ
ワ　ハ
イ　ウ　訪し
彼はト・スを問た



1．まずは連続フレーズを抽出

he → 彼
he → 彼 は
the white house → ホワイト・ハウス
visited → 訪問 した
he visited the white house
→ 彼 は ホワイト・ハウス を 訪問 した
…

2．一部を変数に置き換える

the X_1 house → X_1 ・ハウス
he X_1 the white house → 彼 は ホワイト・ハウス を X_1
he visited X_1 → 彼 は X_1 を 訪問 した
…

階層的フレーズベース デコーディング

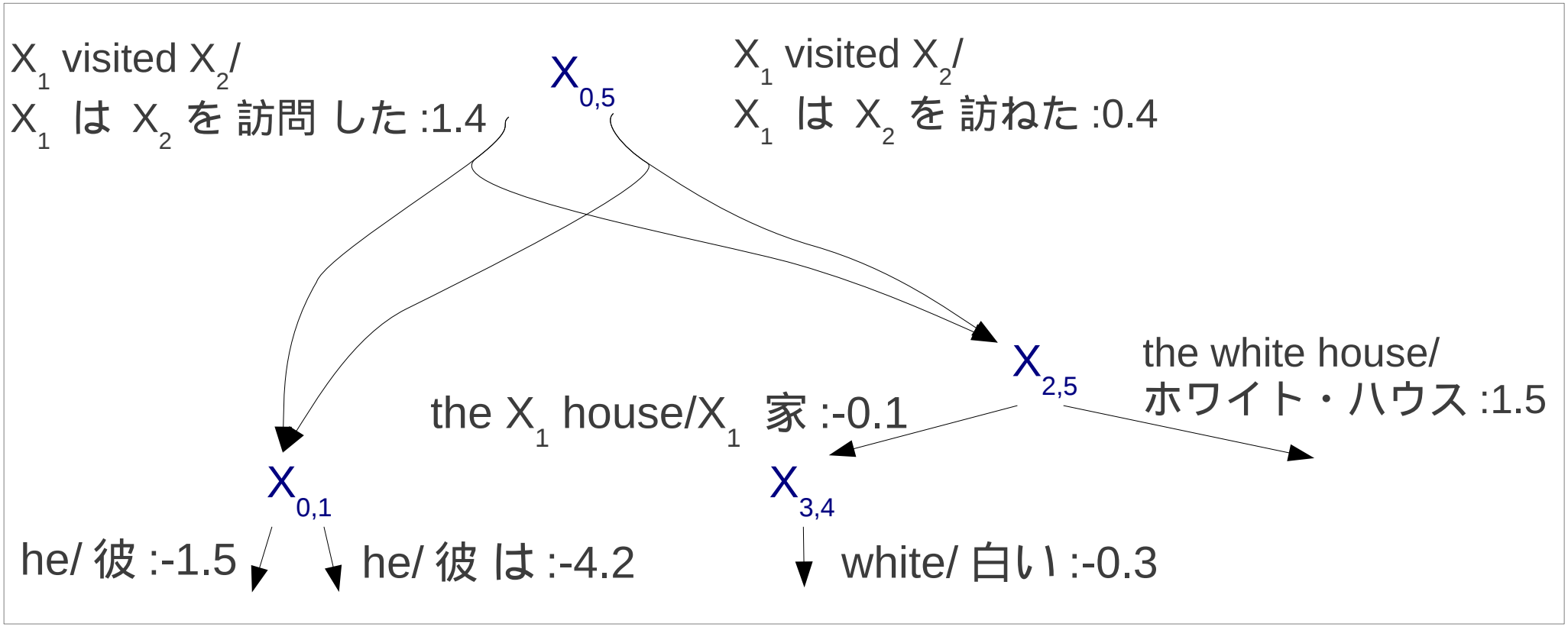
- 超グラフのスコア最大の木を探索することで訳出
- まずは文にマッチするルールを探索

he visited the white house

原言語	目的言語	スコア
he	彼	-1.5
he	彼 は	-4.2
X_1 visited X_2	X_1 は X_2 を 訪問 した	1.4
X_1 visited X_2	X_1 は X_2 を 訪ねた	0.4
the white house	ホワイト・ハウス	1.5
the X_1 house	X_1 家	-0.1
white	白い	-0.3

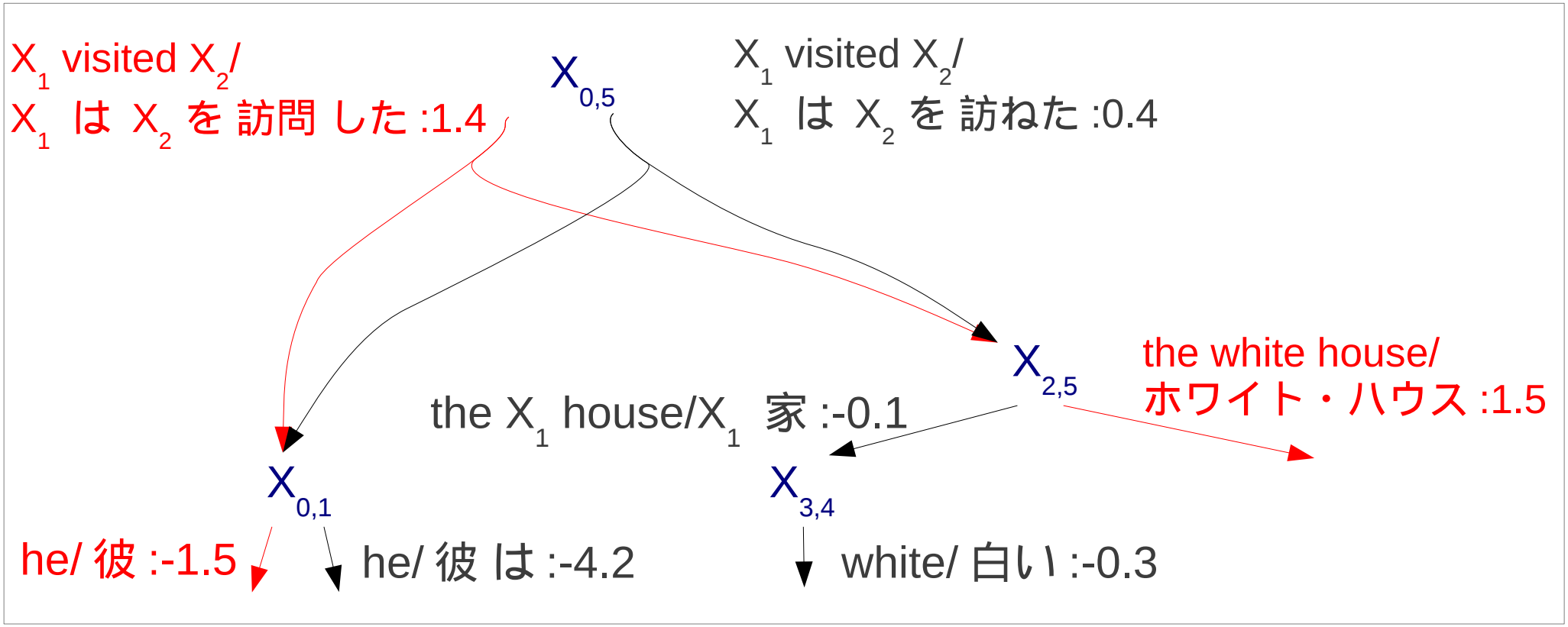
階層的フレーズベース デコーディング

- ルールとその関係を表す超グラフを作成



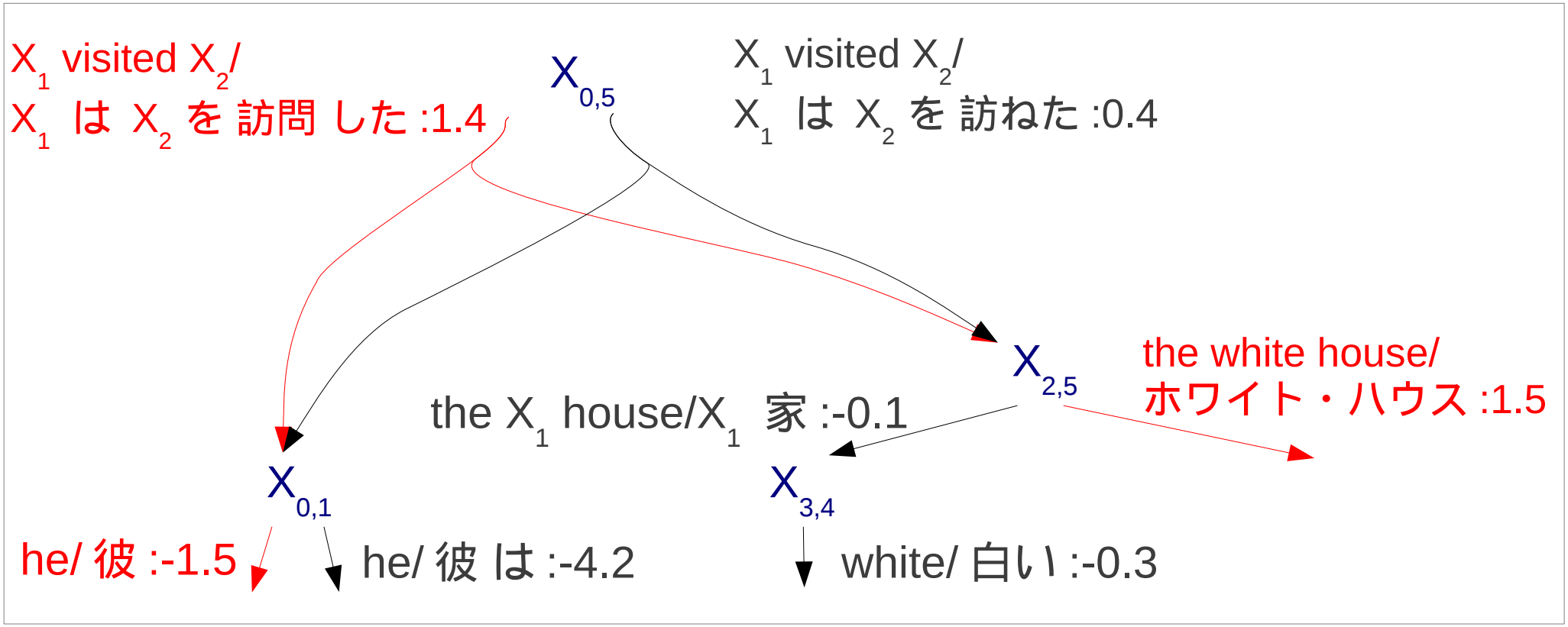
階層的フレーズベース デコーディング

- スコア最大の木を求める



階層的フレーズベース デコーディング

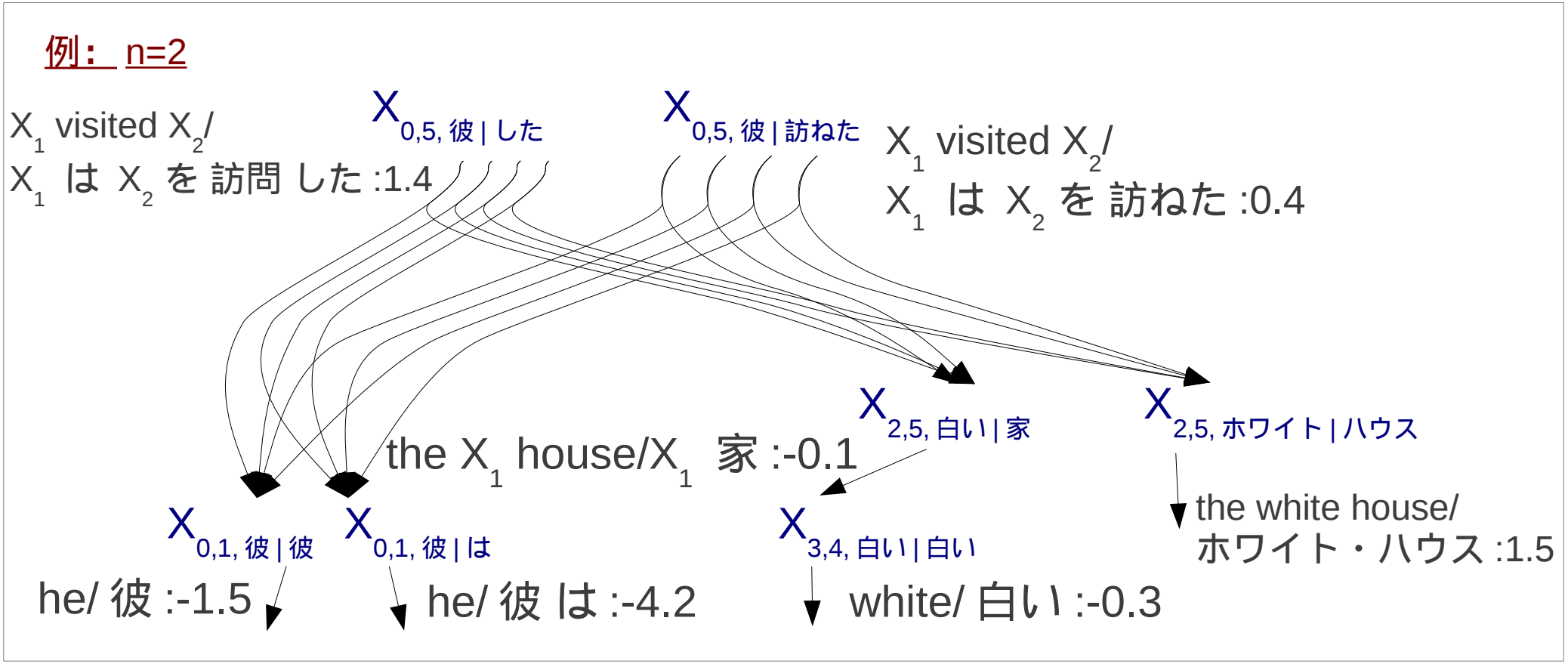
- スコア最大の木による目的言語訳を出力



彼 は ホワイト ・ ハウス を 訪問 した

言語モデルを考慮したデコーディング

- 言語モデルの考慮：
  各ノードの前 n-1、後ろ n-1 単語を保持する必要



言語モデルを考慮したデコーディング

- 探索空間が拡大！
- 近似的な探索が必要
  - ビーム探索を導入：原言語文の文字列に対する仮説数の最大値を設ける
  - Cube Pruning ：子の組み合わせの効率化 [Chiang 07]
  - 並べ替えの制限

演習：階層的フレーズベースモデルの学習

- Moses の通常の学習に
  「-hierarchical -glue-grammar -max-phrase-length 5」
  を追加するだけ

```
$ mosesdecoder/scripts/training/train-model.perl
    -hierarchical -glue-grammar -max-phrase-length 5
    -parallel
    -external-bin-dir $HOME/alagin/usr/local/giza-pp
    -root-dir $HOME/alagin/trainhiero
    -corpus $HOME/alagin/mydata/low/kyoto-train.small
    -f en -e ja
    -alignment grow-diag-final-and
    -reordering msd-bidirectional-fe
    -lm 8:3:$HOME/alagin/lm/kyoto-train.ja.blm &> traininghiero.log
```

- モデルを閲覧： trainhiero/model/rule-table.gz

統語ベース機械翻訳

統語ベース機械翻訳 [Sato+ 90]

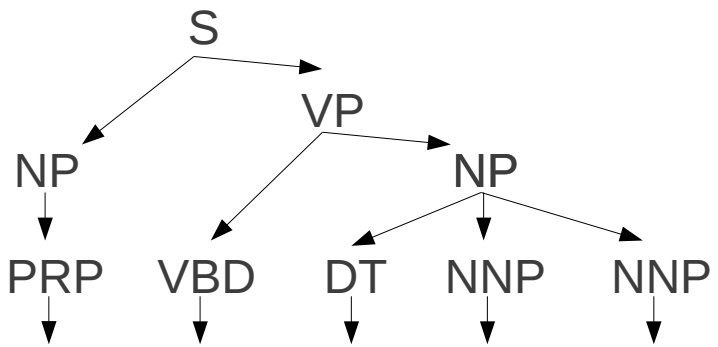
- 今まで紹介した手法は構文解析を利用しない
- 構文解析は句を同定し、曖昧性を解消
  →訳の質向上につながると考えられる
- 原言語でも目的言語でも利用可能

翻訳モデルの種類

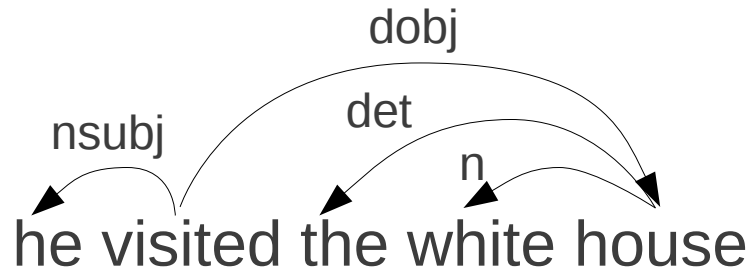
string

he visited the white house

tree



dependency

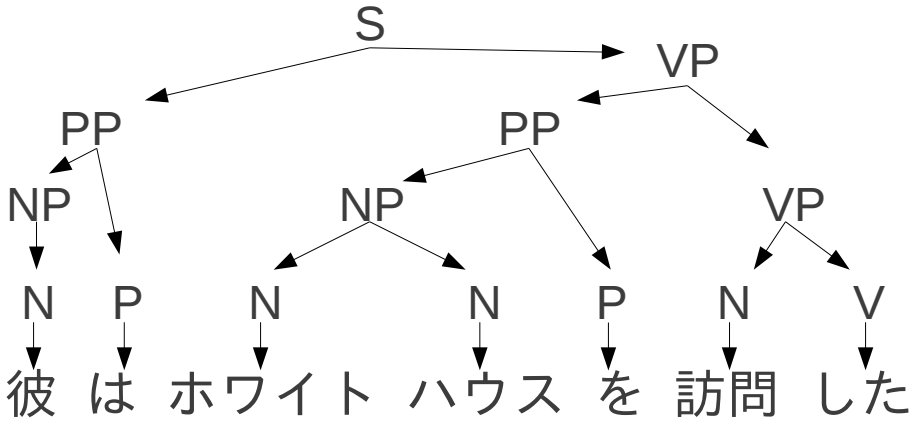


to

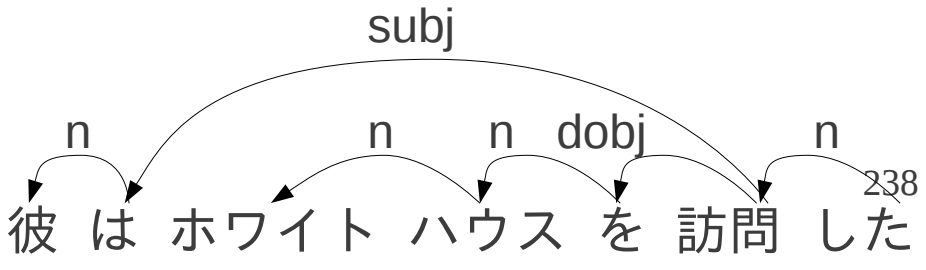
string

彼 は ホワイト ハウス を 訪問 した

tree



dependency



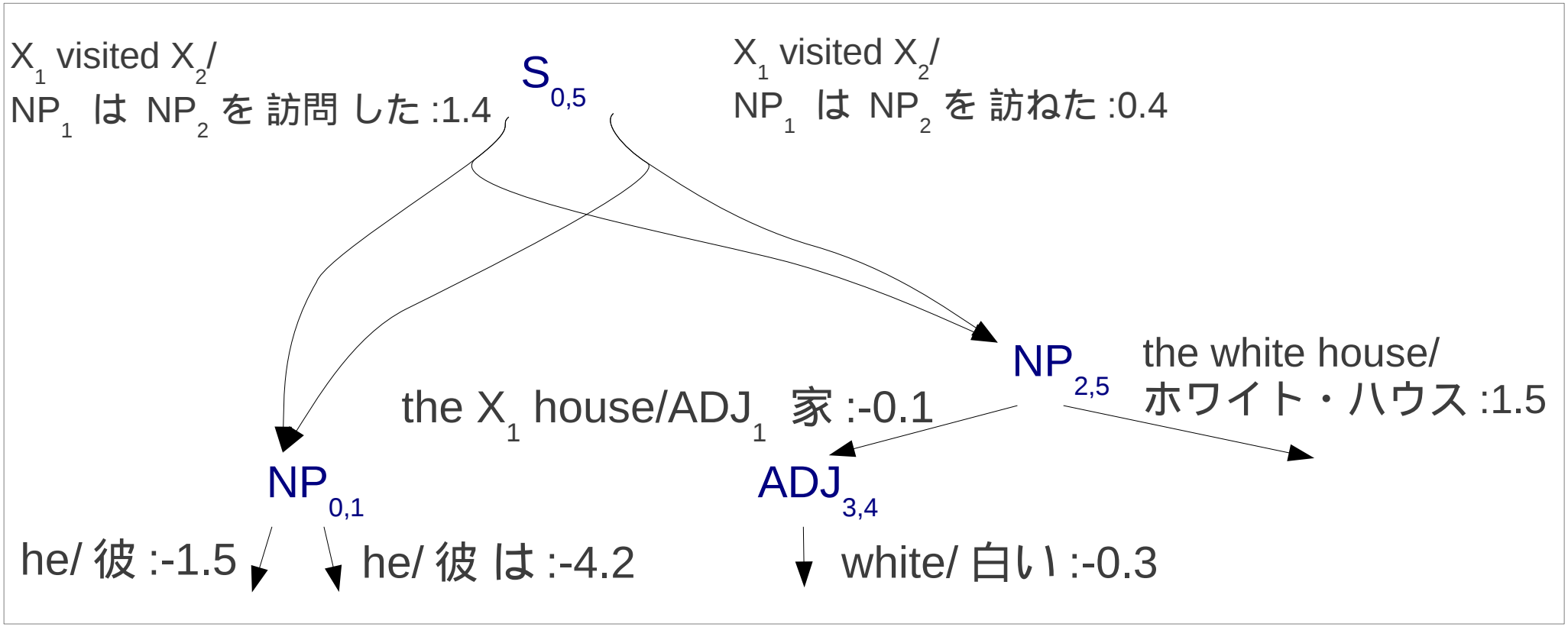
string-to-tree 翻訳 [Galley+ 06]

- 目的言語側のみで統語情報を利用
- 階層的フレーズベースとほとんど同じ仕組み
- ルールの目的言語側に句のラベルを付与する

原言語	目的言語	句	スコア
he	彼	NP	-1.5
he	彼 は	NP	-4.2
X_1 visited X_2	NP_1 は NP_2 を 訪問 した	S	1.4
X_1 visited X_2	NP_1 は NP_2 を 訪ねた	S	0.4
the white house	ホワイト ・ ハウス	NP	1.5
the X_1 house	ADJ_1 家	NP	-0.1
white	白い	ADJ	-0.3

string-to-tree 翻訳 [Galley+ 06]

- デコーディングの際は目的言語の句ラベルを考慮



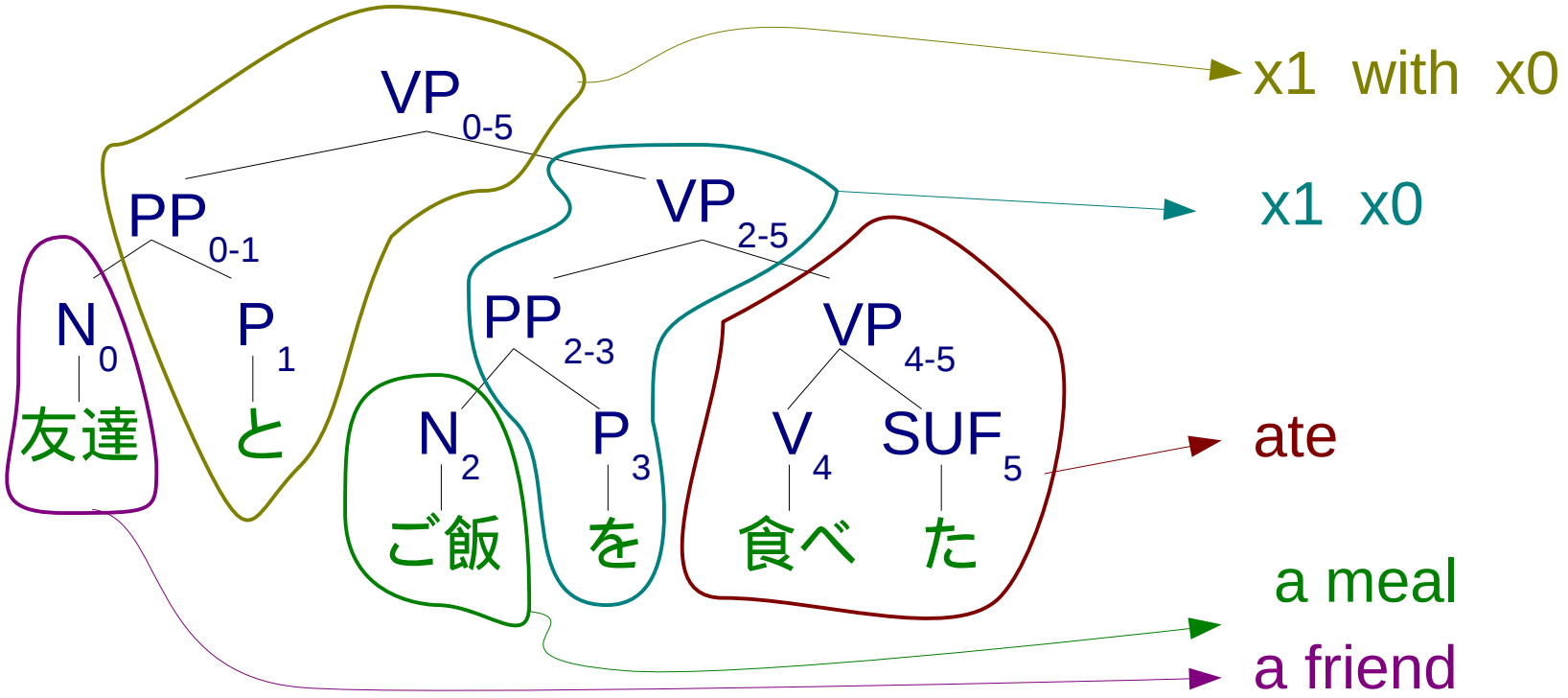
- ルールと合わないラベルのノードを利用しない
  （NP のところに ADJ を入れない）

tree-to-string 翻訳

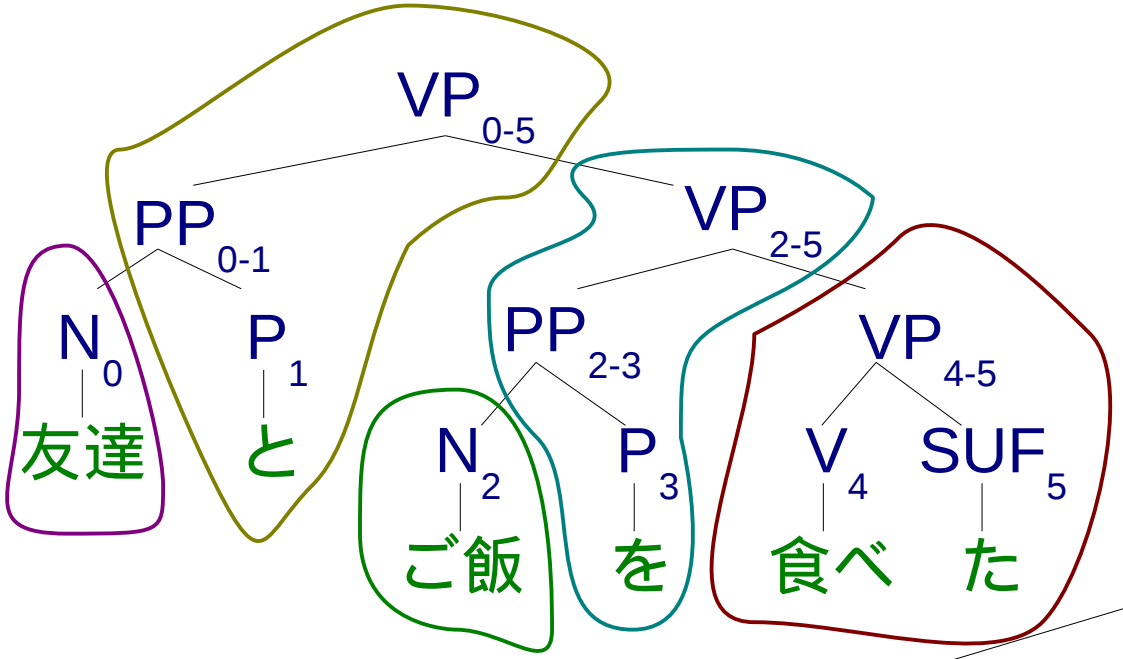
- 原言語側のみに統語情報を利用
- ２種類の方式
  - 同時構文解析＋翻訳：仕組みは string-to-tree と同様
    - ＋構文解析誤りに比較的頑健
    - - 遅い
    - - 並べ替え制限が必要
  - 事前構文解析：事前に解析を行ってから翻訳
    - ＋ 速い
    - ＋ 長距離の並べ替えは問題ない
    - - 解析誤りの影響大

句構造に基づく翻訳 [Liu+ 06]

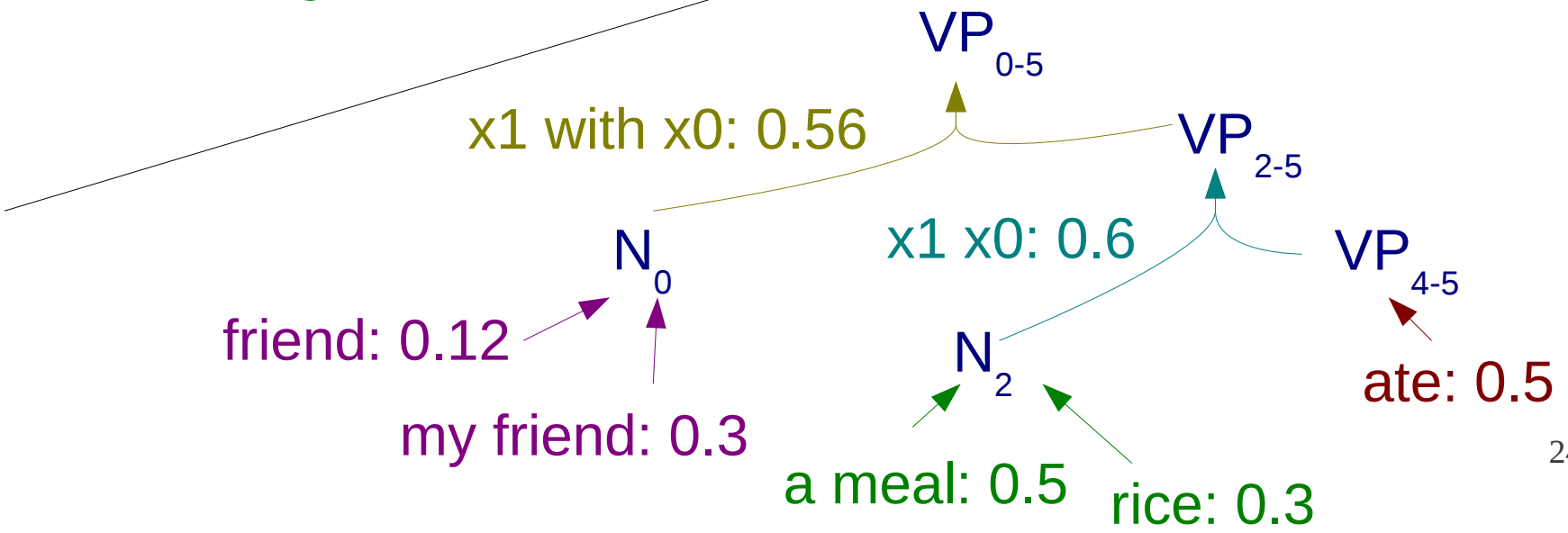
- 構文木上のルールマッチングを行う



句構造に基づく翻訳 [Liu+ 06]

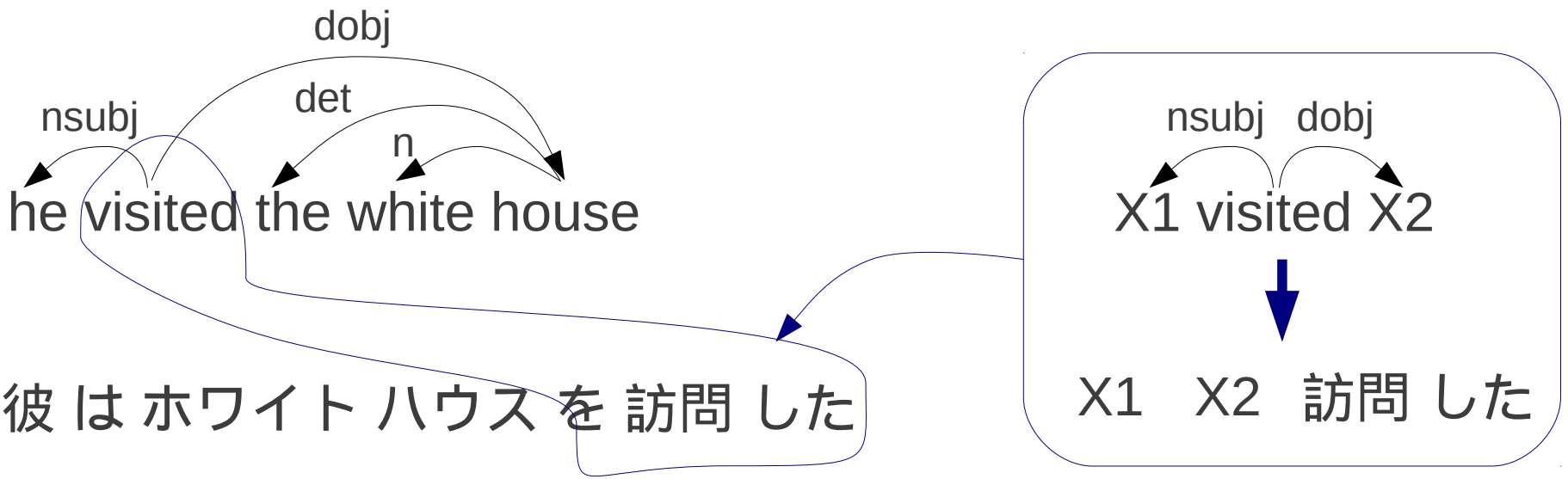


- ルールを表す超グラフを作成
- デコーディングは階層的フレーズベースと類似



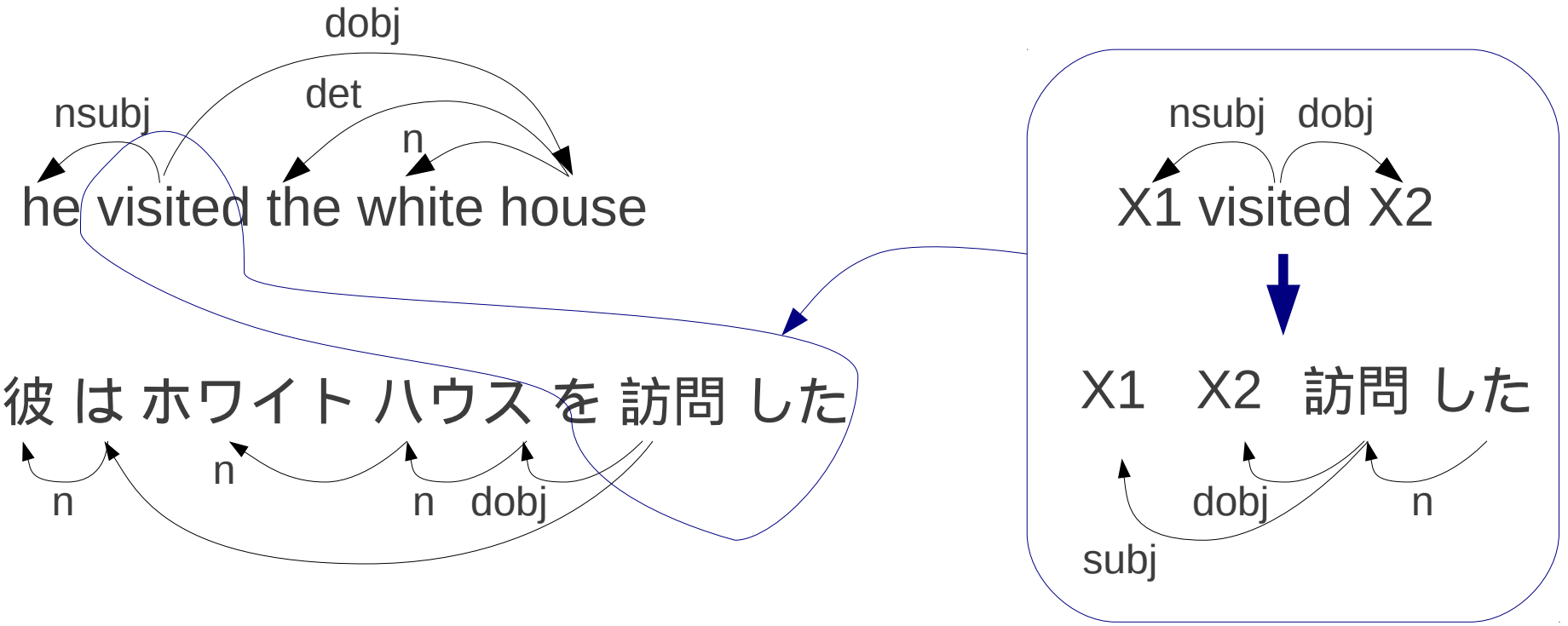
係り受け構造に基づく翻訳 [Quirk+ 05]

- dependency-to-string 翻訳もある



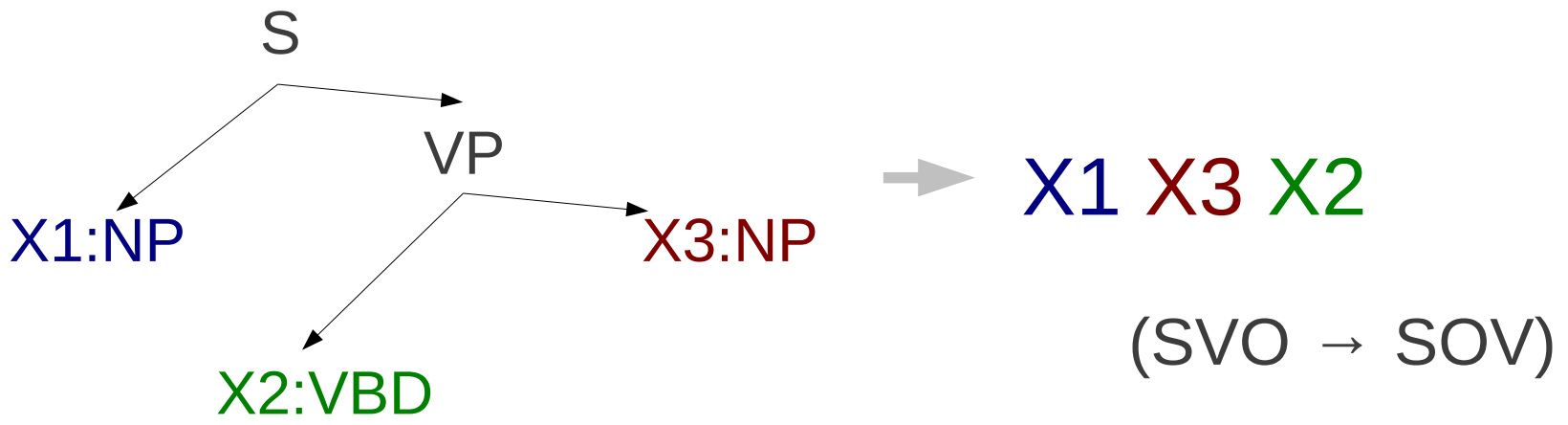
係り受け構造に基づく翻訳 [Sato+ 90, Nakazawa+ 06]

- 京大のシステムは dependency-to-dependency で両言語に対する係り受けを利用



句構造 vs. 係り受け構造

- 句構造：語彙化されていないルールも利用可→一般性



- 係り受け構造：関係のある単語は木上近いところにある→語彙選択に強い？

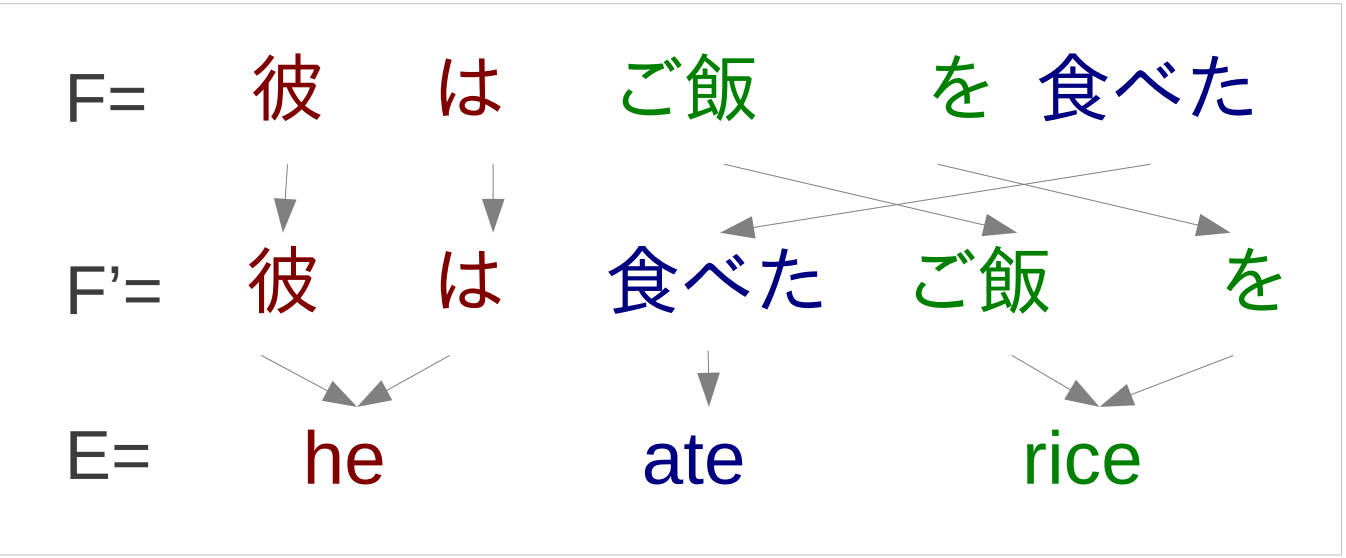
統語情報を使った翻訳の実装

- ツール：
  - 同時パージング・翻訳： Moses は {tree|string}-to-{tree|string} を実装
  - 事前パージング：現在良いオープンソースの実装はない（近日公開？）
- （少し複雑なため本セミナーで演習は行わない）

前並べ替え

前並べ替え [Xia+ 04]

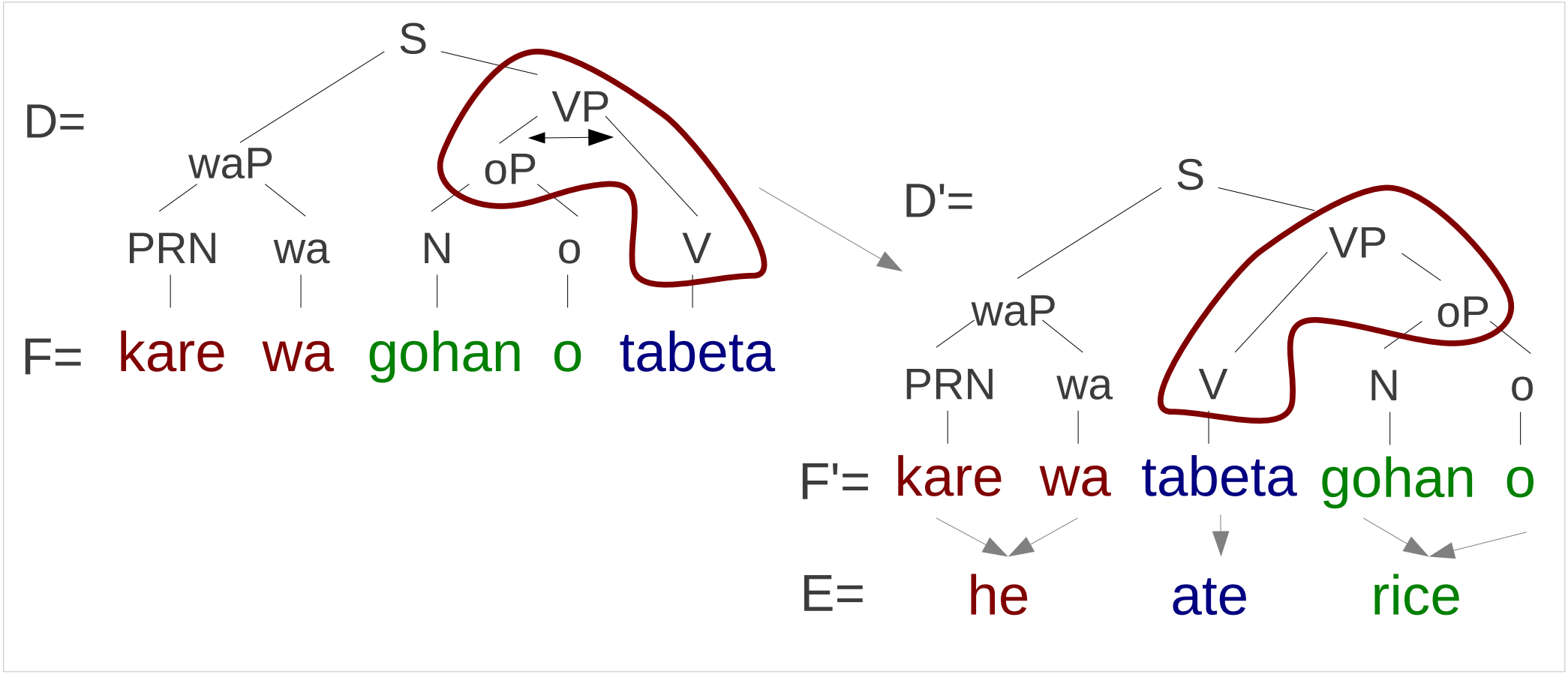
- フレーズベース翻訳は様々な魅力があるが、長距離の並べ替えは弱点
- 前並べ替え：まず原言語文を目的言語文に近い順番に並べ替え



- フレーズベース翻訳の精度向上が見込まれる

統語情報を用いた前並べ替え

- 原言語の構文木に対して並べ替えルールを定義



非常にシンプルなルールで大きな効果

- 数ルールでドイツ語英語翻訳の精度を大幅に向上 [Collins+ 05]
- 英語から SOV 言語への翻訳のためのルール [Xu+ 08]
- 英日翻訳において「統語的主辞を後ろに持っていけば良い」 [Isozaki+ 10]

Head Finalization [Isozaki+ 10]

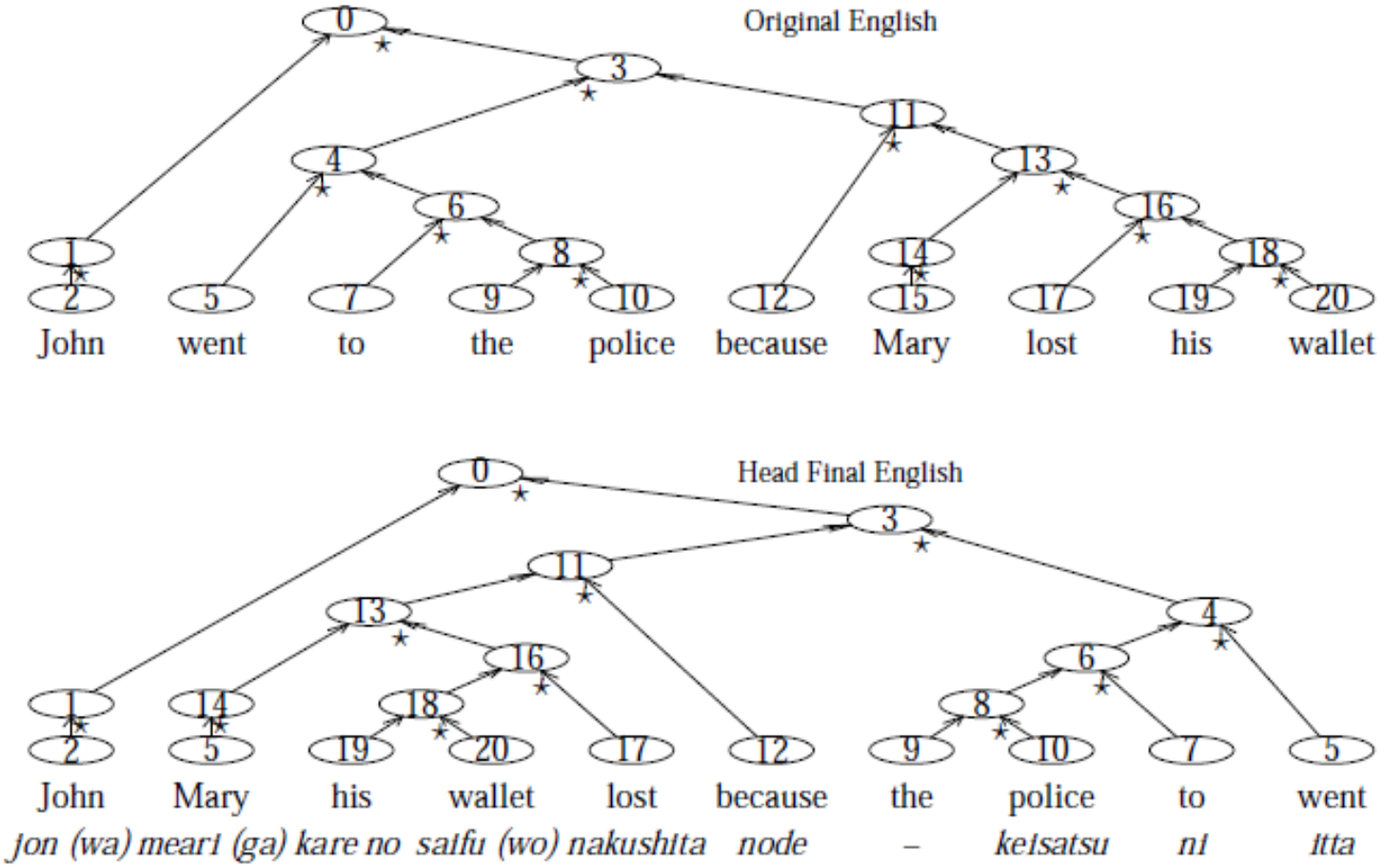
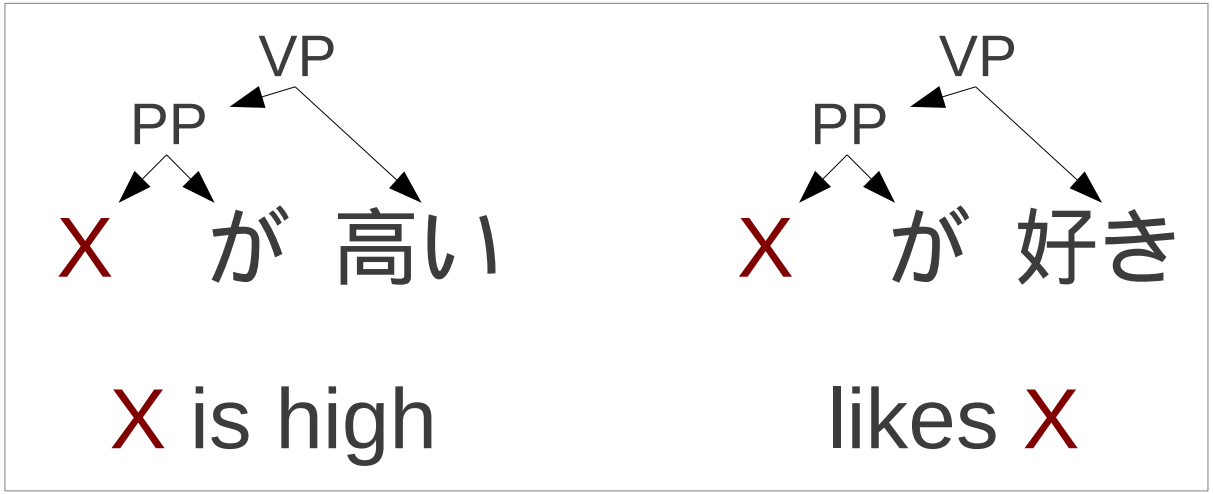


Figure 3: Head-Finalizing a complex sentence.

前並べ替えと tree-to-string

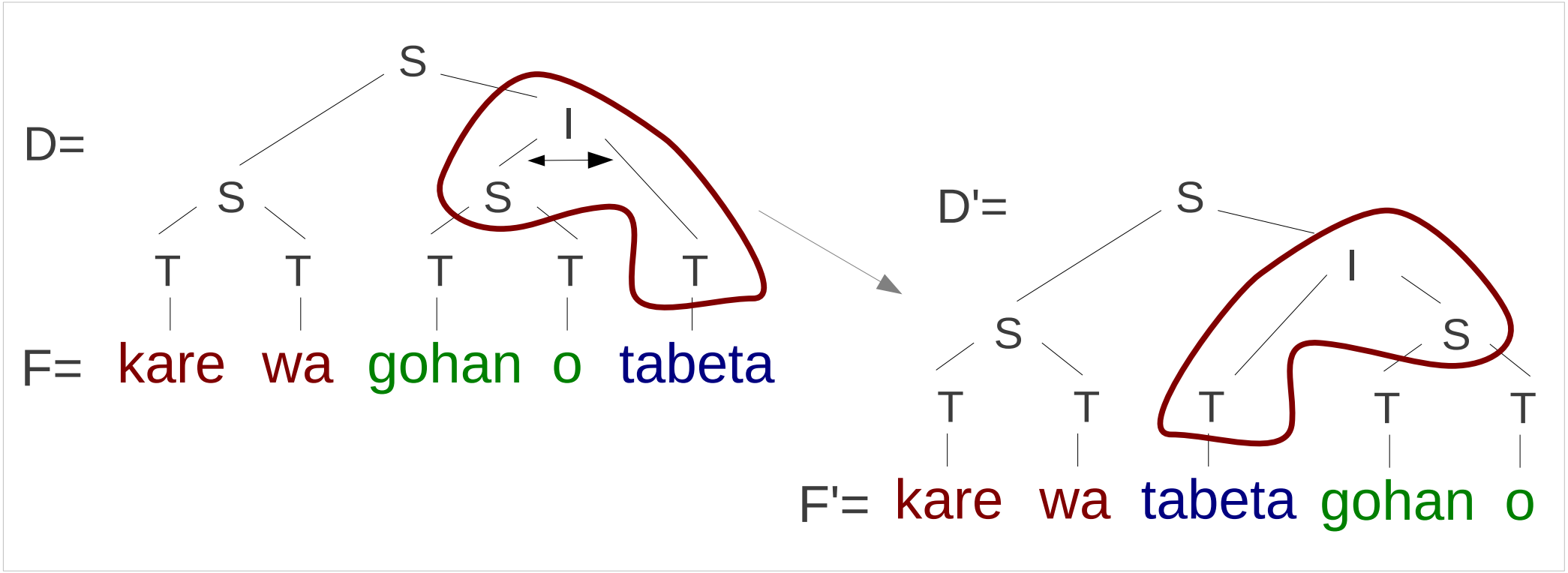
- tree-to-string は並べ替えと翻訳を同時に探索→精度 +
- tree-to-string は語彙化されたルールを簡単に扱える



- 前並べ替えは句をまたぐ翻訳ルールも利用可能

統語情報を利用しない前並べ替え [Neubig+ 12]

- Inversion Transduction Grammar を利用



- 言語非依存、対訳データから学習可能
- lader ツールキットとして実装
  http://www.phontron.com/lader

演習：ITG を使った前並べ替え

- lader の英日翻訳用モデルを利用
- usr/local/lader-model-kftt-en-ja/run-lader-en-ja.sh を編集
  - LADER_DIR を「 $HOME/alagin/usr/local/lader-0.1.3 」に
- usr/local/lader-model-kftt-en-ja/kftt-en-ja.mod を編集
  - 「 /home/ ユーザ /alagin/usr/local/lader-0.1.3/kftt-en-ja.pt 」のようにフルパスを kftt-en-ja.pt のところに置き換える
- 並べ替えるファイルを作成し、並べ替える

```
$ echo "kiyomizu-dera temple is one of the famous sites in kyoto ."
                                                    > mydata/to-parse.en
$ usr/local/lader-model-kftt-en-ja/run-lader-en-ja.sh
kiyomizu-dera temple kyoto in famous sites the of one is .
```

清水 寺 は 京都 の 有名 な 場所 の 一つ である 。

実用的な話

翻訳システムの改良過程

- 初期システムを作成
- 繰り返し：
  - 翻訳を生成し、人手評価を行う
  - 最も問題となっている箇所を定量的に分析
  - その問題を解決する方法を調べる
  - 実装して導入する
    - 確認のため自動評価を利用

オープンソースソフトの使い方

- 機械翻訳はオープンソースソフトで恵まれている
- 使い方がよく分からない時は遠慮なく作者に連絡
  - Moses のメーリングリスト

翻訳がうまく行かない理由

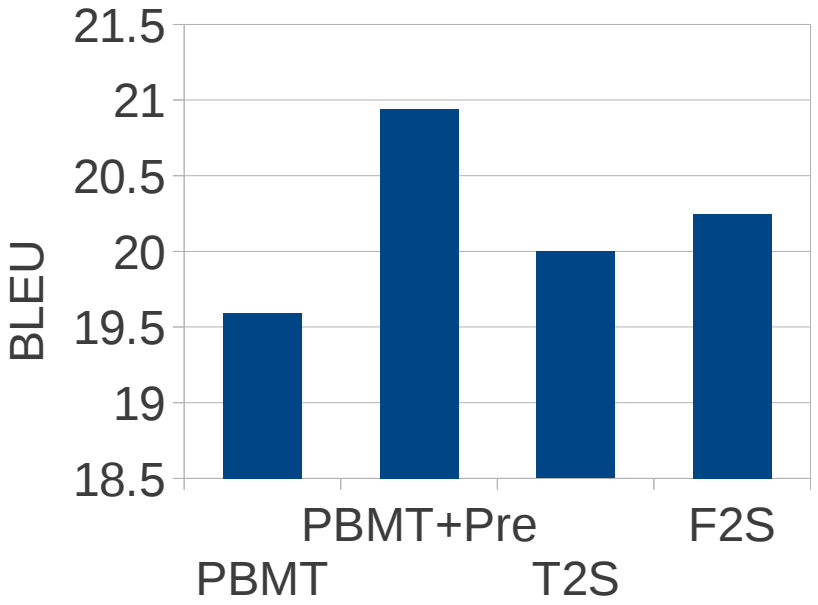
- BLEU が 1.0 ぐらい、何が間違っている？
- よくある犯人：
  - データの行数が合っていない
  - 前処理を行っていない、学習とテストで違う
- ログファイルを読み、エラーを探す

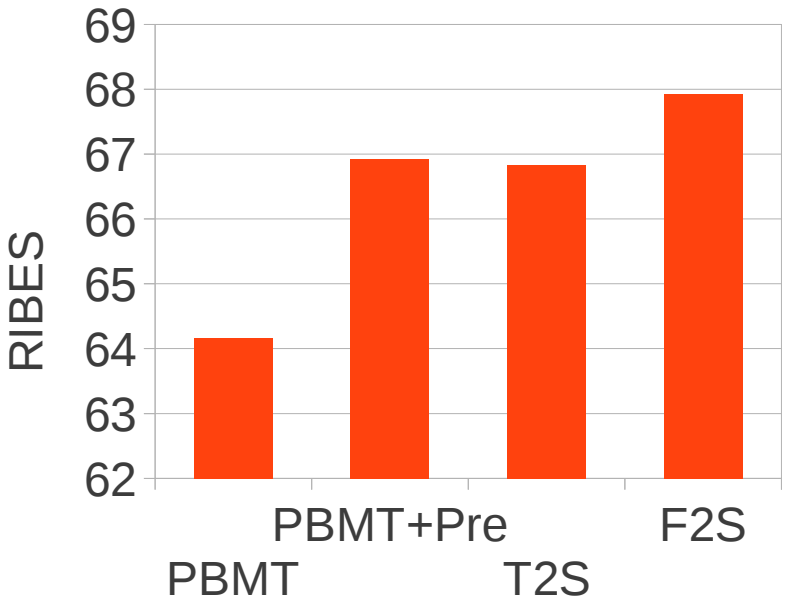
現在・これからの機械翻訳

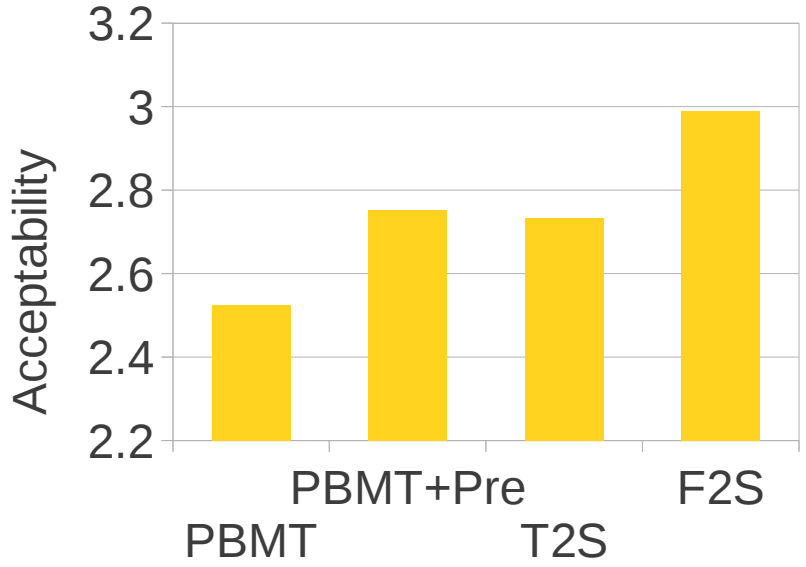
日英翻訳精度の一例

- システム：フレーズベース、前並べ替え (lader) 、 tree-to-string
  - アライメント GIZA++ 、チューニング MERT
- データ：京都関連 Wikipedia 文、学習には ~350k 文
- 評価 : BLEU, RIBES, Acceptability (0-5)
- 構文解析日： KyTea ＋ EDA 、英： Egret

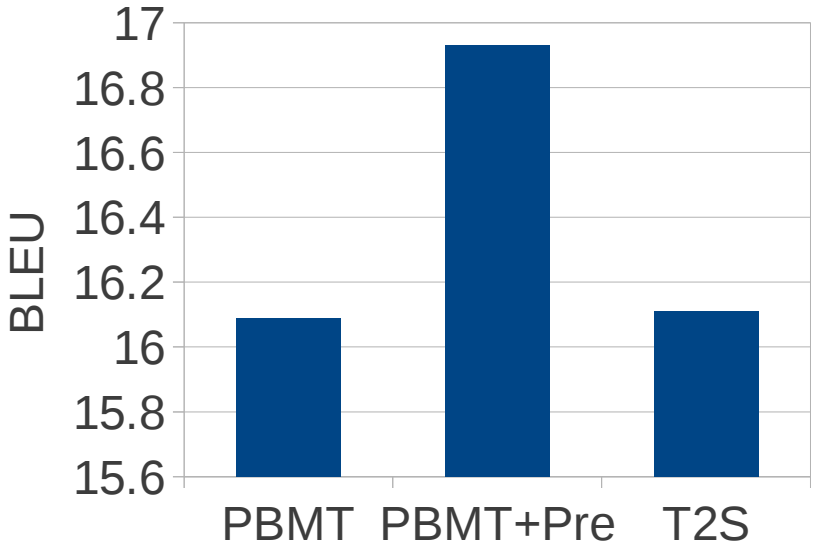
実験結果 (英日)

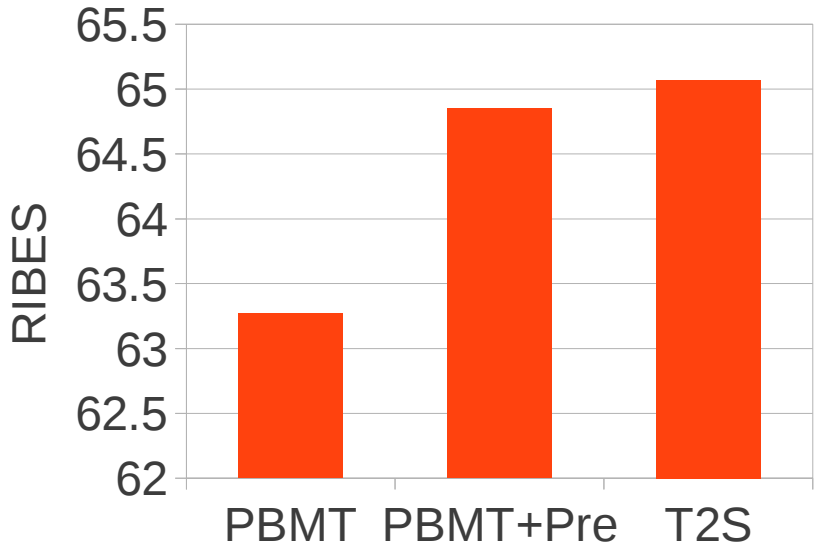

BLEU
21.5
21
20.5
20
19.5
19
18.5
PBMT
PBMT+Pre
T2S
F2S

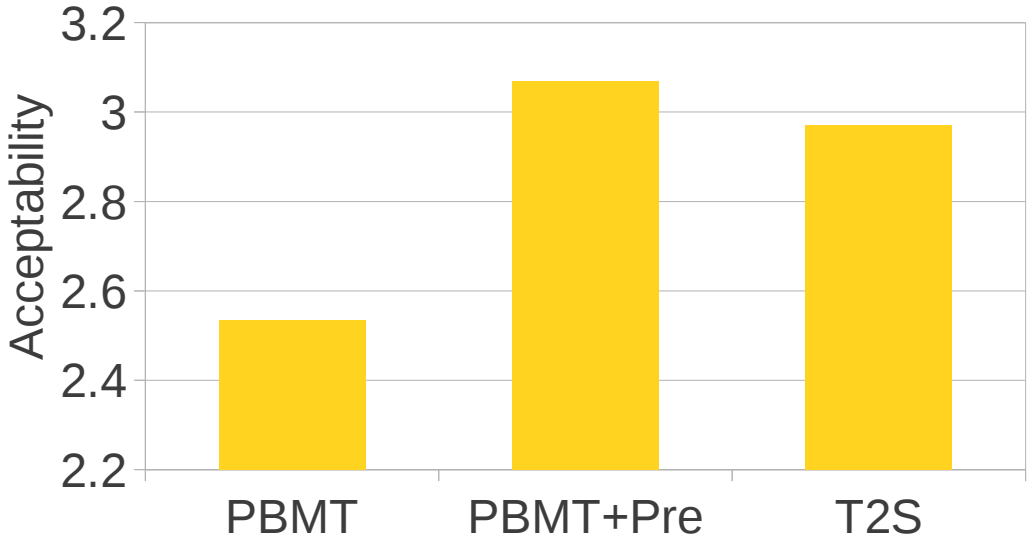

RIBES
69
68
67
66
65
64
63
62
PBMT
PBMT+Pre
T2S
F2S


Acceptability
3.2
3
2.8
2.6
2.4
2.2
PBMT
PBMT+Pre
T2S
F2S

実験結果（日英）


BLEU
17
16.8
16.6
16.4
16.2
16
15.8
15.6
PBMT
PBMT+Pre
T2S


RIBES
65.5
65
64.5
64
63.5
63
62.5
62
PBMT
PBMT+Pre
T2S


Acceptability
3.2
3
2.8
2.6
2.4
2.2
PBMT
PBMT+Pre
T2S

英日　F2S vs. PBMT+Pre

Input:
Department of Sociology in Faculty of Letters opened .

PBMT+Pre:
開業 年 文学 部 社会 学科 。

F2S:
文学 部 社会 学 科 を 開設 。

F2S が名詞句 + 動詞を正しく解釈

英日　F2S vs. PBMT+Pre

Input:
Afterwards it was reconstructed but its influence declined .

PBMT+Pre:
その 後 衰退 し た が 、 その 影響 を 受け て 再建 さ れ た もの で あ る 。

F2S:
その 後 再建 さ れ て い た が 、 影響 力 は 衰え た 。

F2S が 2 つの動詞句の関係を正しく復元

英日　F2S vs. PBMT+Pre

Input:
Introduction of KANSAI THRU PASS Miyako Card

PBMT+Pre:
スルッと kansai 都 カード の 導入

F2S:
伝来 スルッと KANSAI 都 カード

F2S の構文解析誤り
(NP (NP Introduction) (PP of KANSAI THRU PASS) (NP Miyako) (NP Card))

日英 T2S vs. PBMT+Pre

Input:
史実 に は 直接 の 関係 は な い 。

PBMT+Pre:
in the historical fact is not directly related to it .

T2S:
is not directly related to the historical facts .

“ に は … 関係 が” が正しく “ related to” と翻訳

日英 T2S vs. PBMT+Pre

Input:

九条 道家 は 嫡男 ・ 九条 教実 に 先立 た れ 、 次男 ・ 二条 良実 は 事実 上 の 勘当 状態 に あ っ た 。

PBMT+Pre:

michiie kujo was his eldest son and heir , norizane kujo , and his second son , yoshizane nijo was disinherited .

T2S:

michiie kujo to his legitimate son kujo norizane died before him , and the second son , nijo yoshizane was virtually disowned .

T2S が句を正しく分割

日英 T2S vs. PBMT+Pre

Input:
日本 語 日本 文学 科
1474 年 ~ 1478 年 - 山名 政 豊

PBMT+Pre:
the department of japanese language and literature
in 1474 to 1478 - masatoyo yamana

T2S:
japanese language and literature
masatoyo yamana 1474 shokoku-ji in -

T2S が句をまたぐルールの抽出が失敗
T2S の構文解析誤り (“Yamana” が 1 つの名詞句として解釈)

現在の機械翻訳は何が得意？

- 多くの言語対で実用的なレベル
  - 特に英仏、日韓のような近い言語対
  - フレーズベース翻訳が力を発揮
- 大規模の対訳データ
  - 対訳の言語資源が豊富な言語間は高精度な翻訳
  - Web などから対訳データを大量に収集できる分野
- 専門用語と慣用句

現在の機械翻訳は何が苦手？

- 統語や活用の形態が大きく異なる言語対
  - 統語情報を用いた手法で改善を図れる
  - が…、構文解析誤りが翻訳の精度を悪化
  - 大きく異なる言語対では単語対応が取りにくい
- ローカルな情報で解決できない問題
  - 活用の一致、文書の一貫性
- 大規模な学習コーパスが存在しない場合
  - 少数言語
  - 対訳データが存在しない分野、書き方（口語）
  - 英語を含まない言語対

参考文献

参考文献

- J. Albrecht and R. Hwa. A re-examination of machine learning approaches for sentence-level mt evaluation. In Proc. ACL, pages 880-887, 2007.
- V. Ambati, S. Vogel, and J. Carbonell. Active learning and crowdsourcing for machine translation. Proc. LREC, 7:2169-2174, 2010.
- S. Banerjee and A. Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In Proc. ACL Workshop, 2005.
- Y. Bengio, H. Schwenk, J.-S. Senecal, F. Morin, and J.-L. Gauvain. Neural probabilistic language models. In Innovations in Machine Learning, volume 194, pages 137-186. 2006.
- T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. Large language models in machine translation. In Proc. EMNLP, pages 858-867, 2007.
- P. F. Brown, V. J. Pietra, S. A. D. Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. Computational Linguistics, 19:263-312, 1993.
- C. Callison-Burch, P. Koehn, C. Monz, and O. Zaidan. Findings of the 2011 workshop on statistical machine translation. In Proc. WMT, pages 22-64, 2011.
- M. Carpuat and D. Wu. How phrase sense disambiguation outperforms word sense disambiguation for statistical machine translation. In Proc. TMI, pages 43-52, 2007.
- D. Cer, C. Manning, and D. Jurafsky. The best lexical metric for phrasebased statistical MT system optimization. In NAACL HLT, 2010.
- P.-C. Chang, M. Galley, and C. D. Manning. Optimizing Chinese word segmentation for machine translation performance. In Proc. WMT, 2008.
- E. Charniak, K. Knight, and K. Yamada. Syntax-based language models for statistical machine translation. In MT Summit IX, pages 40-46, 2003.
- S. Chen. Shrinking exponential language models. In Proc. NAACL, pages 468-476, 2009.

参考文献

- D. Chiang. Hierarchical phrase-based translation. Computational Linguistics, 33(2), 2007.
- T. Chung and D. Gildea. Unsupervised tokenization for machine translation. In Proc. EMNLP, 2009.
- M. Collins, P. Koehn, and I. Kucerova. Clause restructuring for statistical machine translation. In Proc. ACL, 2005.
- J. DeNero, A. Bouchard-Cote, and D. Klein. Sampling alignment structure under a Bayesian translation model. In Proc. EMNLP, 2008.
- J. DeNero and D. Klein. Tailoring word alignments to syntactic machine translation. In Proc. ACL, volume 45, 2007.
- K. Duh, K. Sudoh, X. Wu, H. Tsukada, and M. Nagata. Learning to translate with multiple objectives. In Proc. ACL, 2012.
- A. Fraser and D. Marcu. Semi-supervised training for statistical word alignment. In Proc. ACL, pages 769-776, 2006.
- M. Galley, J. Graehl, K. Knight, D. Marcu, S. DeNeefe, W. Wang, and I. Thayer. Scalable inference and training of context-rich syntactic translation models. In Proc. ACL, pages 961-968, 2006.
- K. Ganchev, J. V. Graca, and B. Taskar. Better alignments = better translations? In Proc. ACL, 2008.
- J. T. Goodman. A bit of progress in language modeling. Computer Speech & Language, 15(4), 2001.
- I. Goto, B. Lu, K. P. Chow, E. Sumita, and B. K. Tsou. Overview of the patent machine translation task at the ntcir-9 workshop. In Proceedings of NTCIR, volume 9, pages 559-578, 2011.
- A. Haghighi, J. Blitzer, J. DeNero, and D. Klein. Better word alignments with supervised ITG models. In Proc. ACL, 2009.

参考文献

- M. Hopkins and J. May. Tuning as ranking. In Proc. EMNLP, 2011.
- H. Isozaki, T. Hirao, K. Duh, K. Sudoh, and H. Tsukada. Automatic evaluation of translation quality for distant language pairs. In Proc. EMNLP, pages 944-952, 2010.
- H. Isozaki, K. Sudoh, H. Tsukada, and K. Duh. Head nalization: A simple reordering rule for sov languages. In Proc. WMT and MetricsMATR, 2010.
- J. H. Johnson, J. Martin, G. Foster, and R. Kuhn. Improving translation quality by discarding most of the phrasetable. In Proc. EMNLP, pages 967-975, 2007.
- P. Koehn, A. Axelrod, A. B. Mayne, C. Callison-Burch, M. Osborne, and D. Talbot. Edinburgh system description for the 2005 IWSLT speech translation evaluation. In Proc. IWSLT, 2005.
- P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In Proc. HLT, pages 48-54, 2003.
- P. Koehn and J. Schroeder. Experiments in domain adaptation for statistical machine translation. In Proc. WMT, 2007.
- P. Liang, A. Bouchard-C^ote, D. Klein, and B. Taskar. An end-to-end discriminative approach to machine translation. In Proc. ACL, pages 761-768, 2006.
- Y. Liu, Q. Liu, and S. Lin. Tree-to-string alignment template for statistical machine translation. In Proc. ACL, 2006.
- C.-k. Lo and D. Wu. Meant: An inexpensive, high-accuracy, semiautomatic metric for evaluating translation utility based on semantic roles. In Proc. ACL, pages 220-229, 2011.
- W. Macherey, F. Och, I. Thayer, and J. Uszkoreit. Lattice-based minimum error rate training for statistical machine translation. In Proc. EMNLP, 2008.
- D. Marcu and W. Wong. A phrase-based, joint probability model for statistical machine translation. In Proc. EMNLP, 2002.

参考文献

- S. Matsoukas, A.-V. I. Rosti, and B. Zhang. Discriminative corpus weight estimation for machine translation. In Proc. EMNLP, pages 708717, 2009.
- H. Mi, L. Huang, and Q. Liu. Forest-based translation. In Proc. ACL, pages 192-199, 2008.
- R. Moore. Fast and accurate sentence alignment of bilingual corpora. Machine Translation: From Research to Real Users, pages 135-144, 2002.
- T. Nakazawa, K. Yu, D. Kawahara, and S. Kurohashi. Example-based machine translation based on deeper NLP. In Proc. IWSLT, pages 6470, 2006.
- G. Neubig, T. Watanabe, and S. Mori. Inducing a discriminative parser to optimize machine translation reordering. In Proc. EMNLP, pages 843-853, Korea, July 2012.
- G. Neubig, T. Watanabe, S. Mori, and T. Kawahara. Machine translation without words through substring alignment. In Proc. ACL, pages 165-174, Jeju, Korea, July 2012.
- G. Neubig, T. Watanabe, E. Sumita, S. Mori, and T. Kawahara. An unsupervised model for joint phrase alignment and extraction. In Proc. ACL, pages 632-641, Portland, USA, June 2011.
- S. Niessen, H. Ney, et al. Morpho-syntactic analysis for reordering in statistical machine translation. In Proc. MT Summit, 2001.
- F. J. Och. Minimum error rate training in statistical machine translation. In Proc. ACL, 2003.
- F. J. Och and H. Ney. Discriminative training and maximum entropy models for statistical machine translation. In Proc. ACL, 2002.
- K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. BLEU: a method for automatic evaluation of machine translation. In Proc. COLING, pages 311-318, 2002.
- C. Quirk and A. Menezes. Dependency treelet translation: the convergence of statistical and example-based machine-translation? Machine Translation, 20(1):43-65, 2006.

参考文献

- P. Resnik and N. A. Smith. The web as a parallel corpus. Computational Linguistics, 29(3):349-380, 2003.
- S. Sato and M. Nagao. Toward memory-based translation. In Proceedings of the 13th conference on Computational linguistics-Volume 3, pages 247-252. Association for Computational Linguistics, 1990.
- J. Suzuki, K. Duh, and M. Nagata. Distributed minimum error rate training of smt using particle swarm optimization. In Proc. IJCNLP, pages 649-657, 2011.
- S. Vogel, H. Ney, and C. Tillmann. HMM-based word alignment in statistical translation. In Proc. COLING, 1996.
- T. Watanabe, J. Suzuki, H. Tsukada, and H. Isozaki. Online largemargin training for statistical machine translation. In Proc. EMNLP, pages 764-773, 2007.
- F. Xia and M. McCord. Improving a statistical MT system with automatically learned rewrite patterns. In Proc. COLING, 2004.
- 森 信介 , 土屋 雅稔 , 山地 治 , and 長尾 真 . 確率的モデルによる仮名漢字変換 . In 情報処理学会第 125 回自然言語処理研究会 (NL-125), 1998.