

# 深層学習とA\*アルゴリズム による高速なCCG解析

A\* CCG parsing with category and dependency  
factored model

2018/09/28 TokyoCL勉強会

NAIST D2 吉川将司

# 組み合わせ範疇文法(Combinatory Categorical Grammar; CCG)

- 自然言語の理論：

- 複雑な文構造(e.g. 並列句構造)を説明可能
- 意味の理論へのインターフェース(文の論理式への変換)
- 効率的な構文解析アルゴリズムの存在

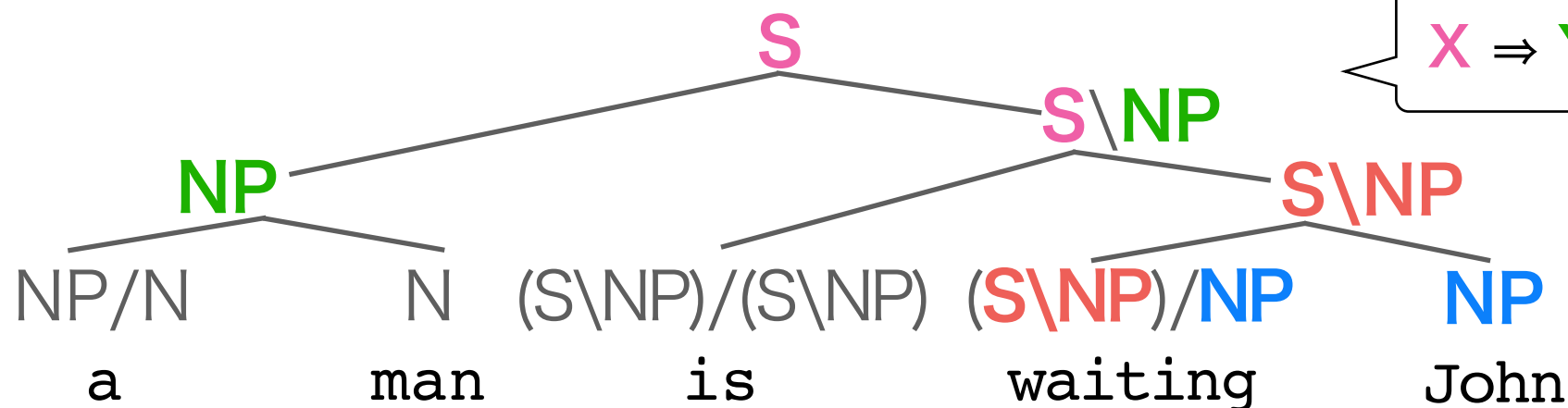
- 自然言語処理での応用：

- CCG木から論理式へ変換し、論理推論
  - 含意関係認識：(Mineshima+, 2015)、(Abzianidze, 2015)
  - 質問応答：(Reddy+, 2014)
  - 文間類似度：(Yanaka+, 2017)

- 特徴：構造がリッチなカテゴリ、少数の文法規則( $\leq 10$ )

出力：

入力：



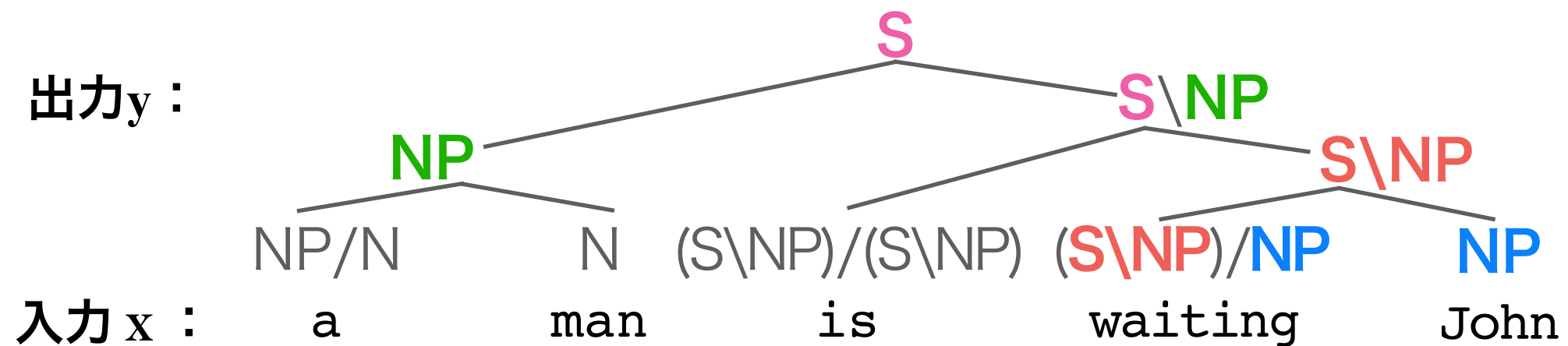
逆方向規則

$X \Rightarrow Y \ X \backslash Y$

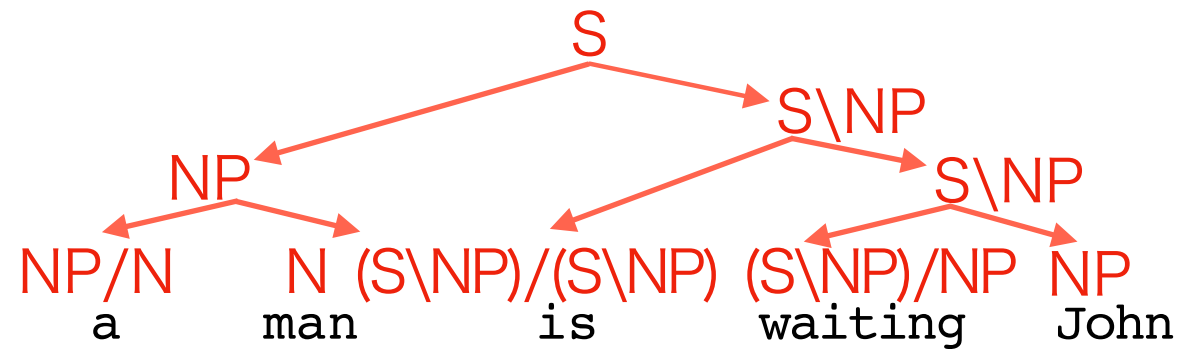
順方向規則

$X \Rightarrow X/Y \ Y$

# 背景：既存の CCG解析手法

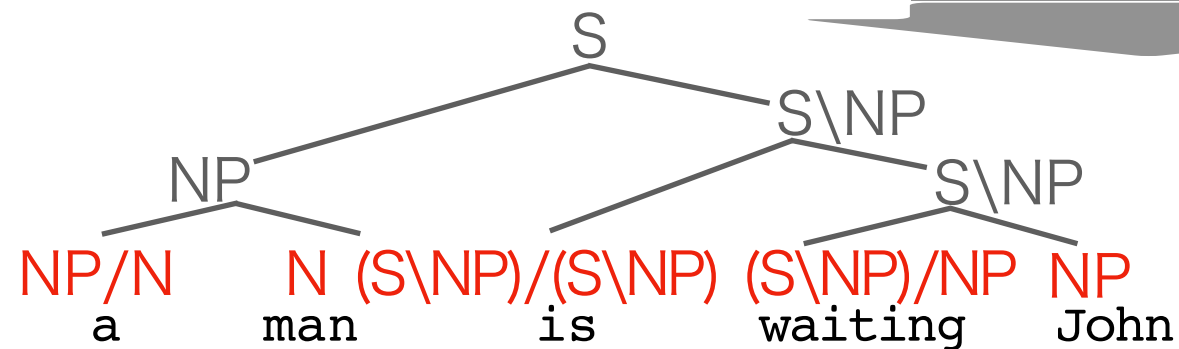


# 既存：Category-factoredモデル(Lewis+, 2014)



これまで：**Full derivationモデル**

$p(\mathbf{y} | \mathbf{x})$  は木の導出全体に依存

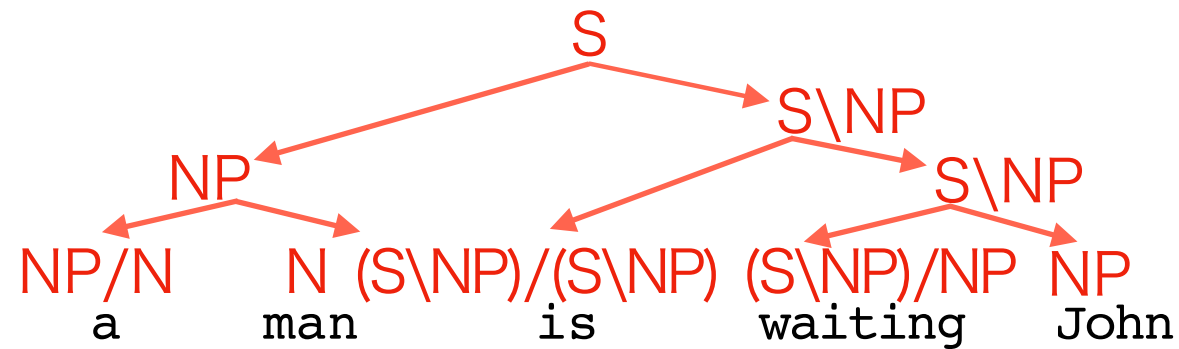


**Category-factoredモデル**

木の確率は終端カテゴリのみで表現可

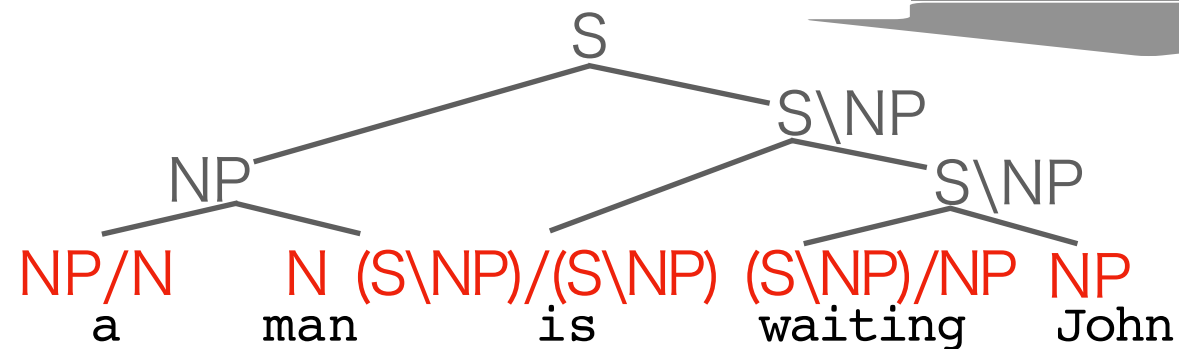
$$p(\mathbf{y} | \mathbf{x}) = \prod p_{tag}(c_i | \mathbf{x})$$

# 既存：Category-factoredモデル(Lewis+, 2014)



これまで：Full derivationモデル

$p(\mathbf{y} | \mathbf{x})$ は木の導出全体に依存



Category-factoredモデル

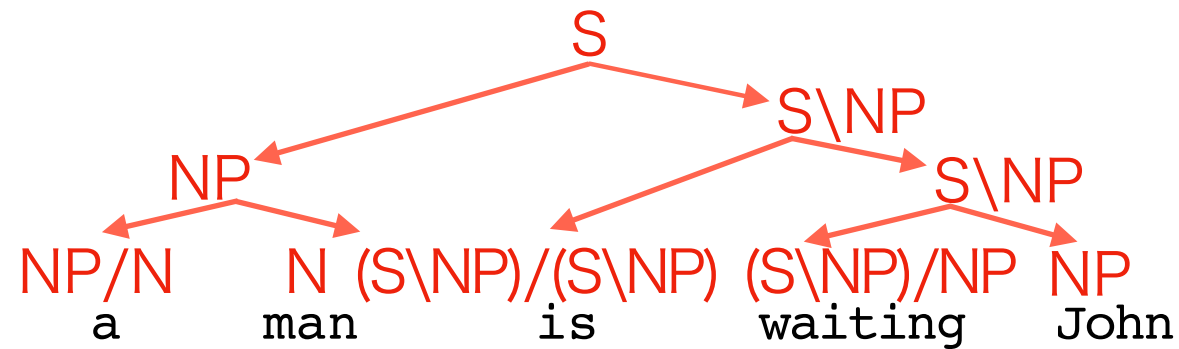
木の確率は終端カテゴリのみで表現可

$$p(\mathbf{y} | \mathbf{x}) = \prod p_{tag}(c_i | \mathbf{x})$$

- なぜこのようなモデルが可能か？

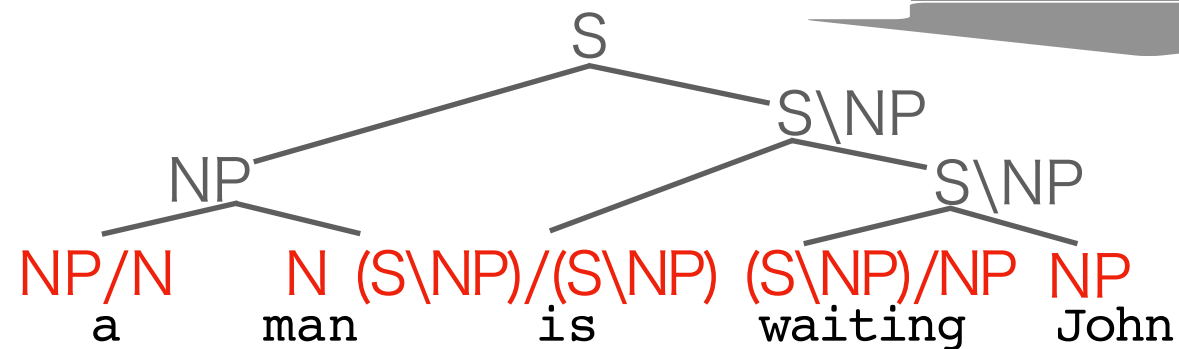
- カテゴリーが構造について多くの情報を持つ(標準形のもとで一意)

# 既存：Category-factoredモデル(Lewis+, 2014)



これまで：**Full derivationモデル**

$p(\mathbf{y} | \mathbf{x})$ は木の導出全体に依存



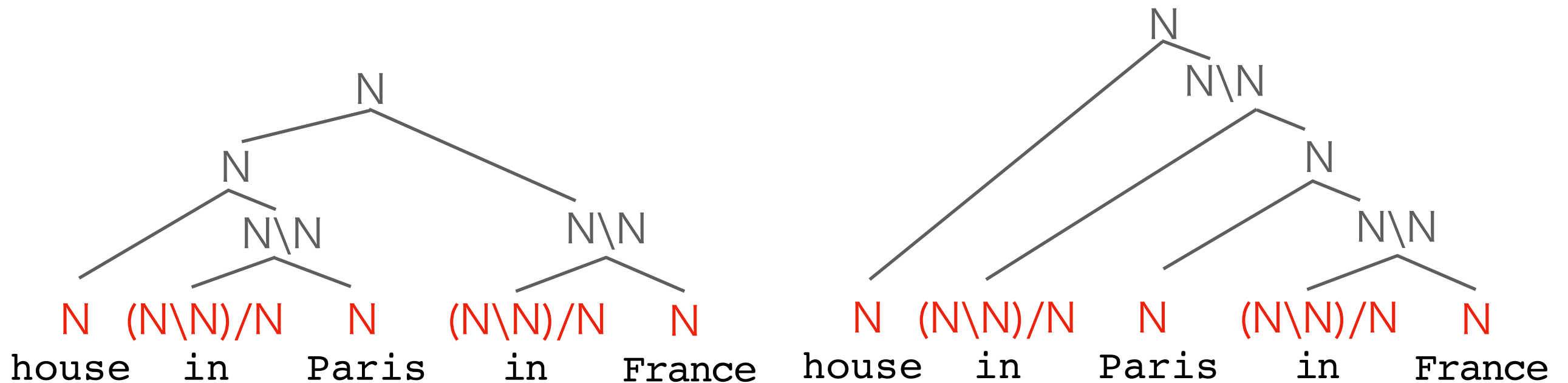
**Category-factoredモデル**

木の確率は終端カテゴリのみで表現可

$$p(\mathbf{y} | \mathbf{x}) = \prod p_{tag}(c_i | \mathbf{x})$$

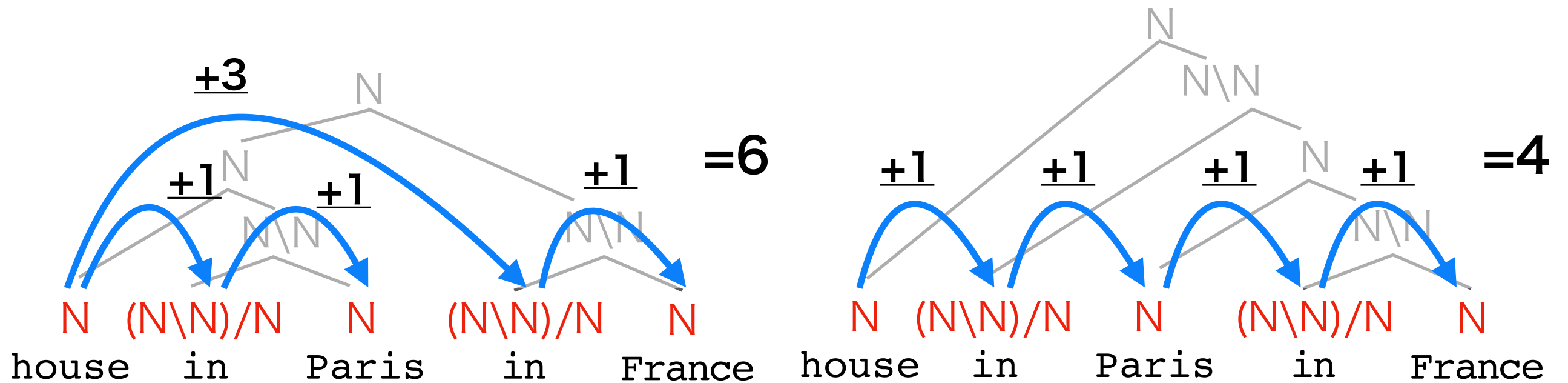
- なぜこのようなモデルが可能か？
  - カテゴリが構造について多くの情報を持つ(標準形のもとで一意)
- **高速**なチャートによる構文解析で最適解が求まる
  - すべての単語とカテゴリに対する $p_{tag}(c_i | \mathbf{x})$ は事前に計算可 → 複数文を並列処理
  - 外側確率(=ゴールまでのコスト)が容易に求まる → A\*解析へ拡張

# Category-factoredモデルの限界



- 同じカテゴリ列から複数の木が導出できるケース
- 等確率になってしまい文構造を一意に決定できない

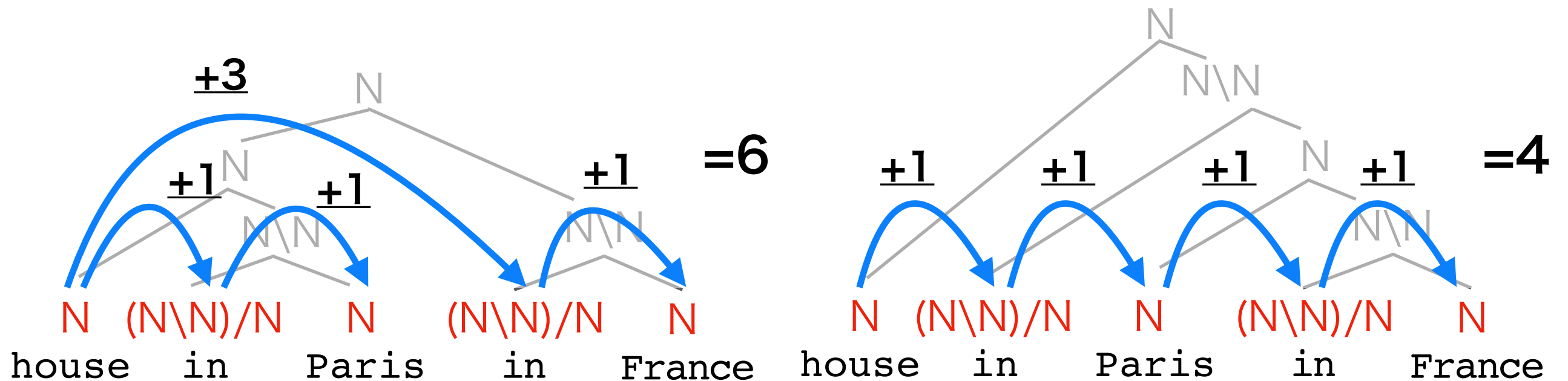
# 対応①：人手によるルール



- ・ 長距離依存を伴うものを優先(Lewis+, 2014)
- ・ CCG木から係り受け木への変換規則を定義

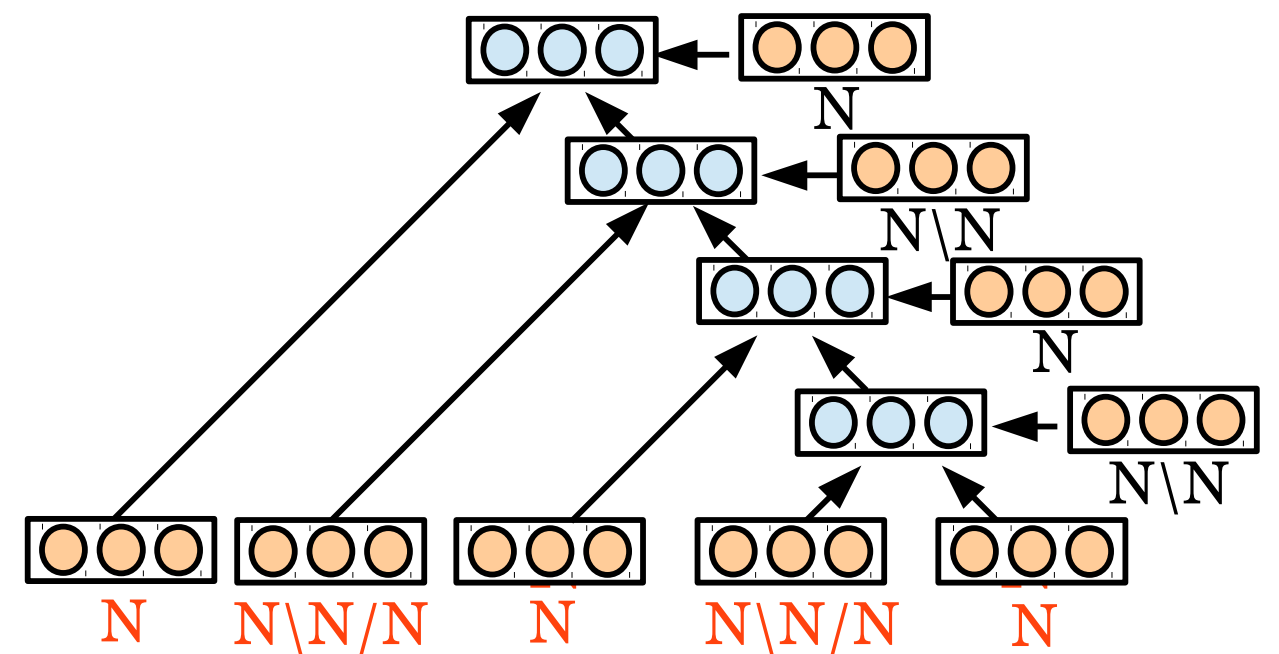
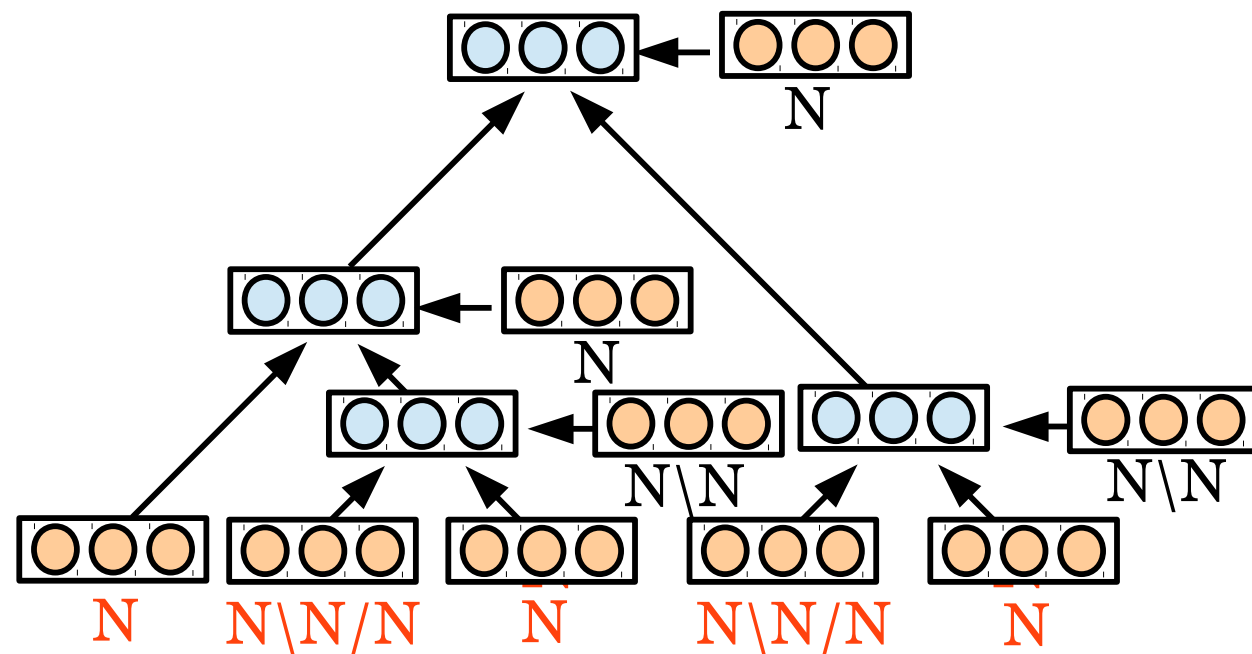


# 対応①：人手によるルール



- ・ 長距離依存を伴うものを優先(Lewis+, 2014)
- ・ CCG木から係り受け木への変換規則を定義
- ・ **問題**：必ずしも長距離依存が正解でない
  - ・ e.g. "house of suburbs of Paris"は短距離が正解  
 $N \quad (N \backslash N) / N \quad N \quad (N \backslash N) / N \quad N$

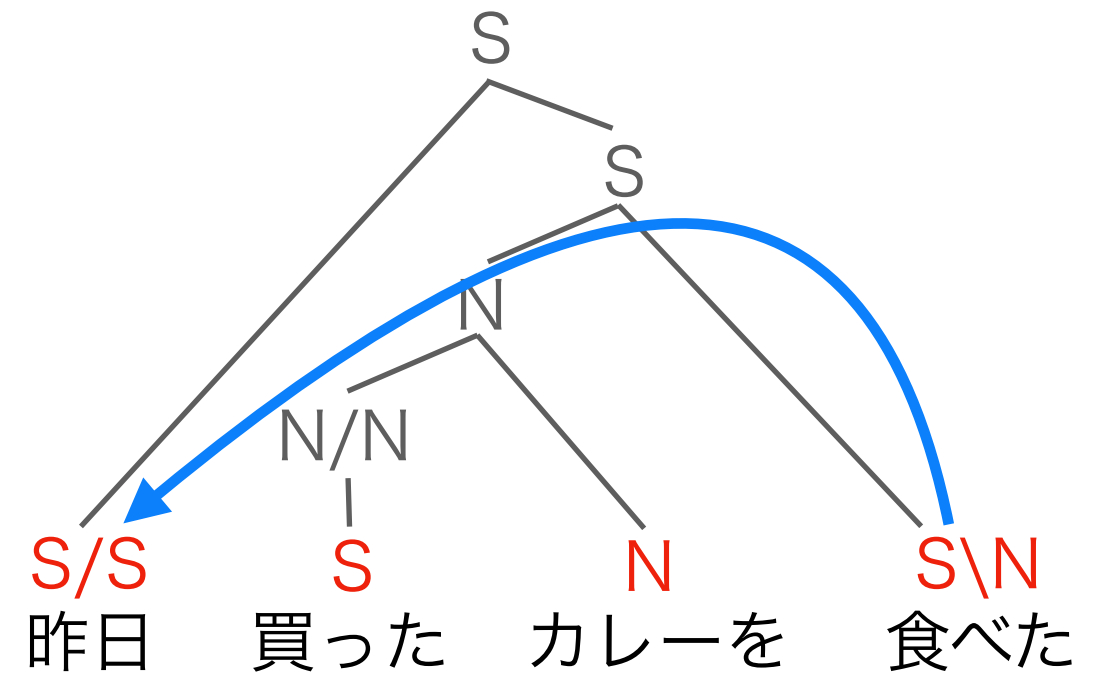
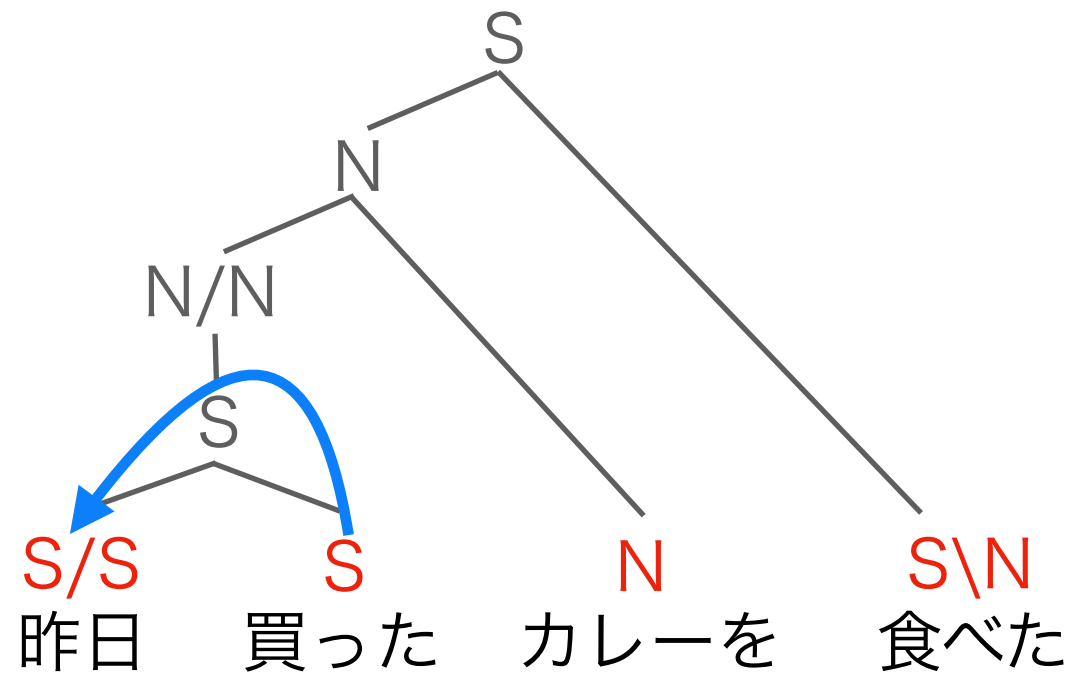
# 対応②：グローバルモデル



- Tree LSTMによる木全体を考慮したモデル(Lee+, 2016)
  - (Lewis+, 2016)を拡張
- 木構造全体を考慮したスコアで高精度を達成
- 問題：局所的な項に分解不可 → 実行速度に影響

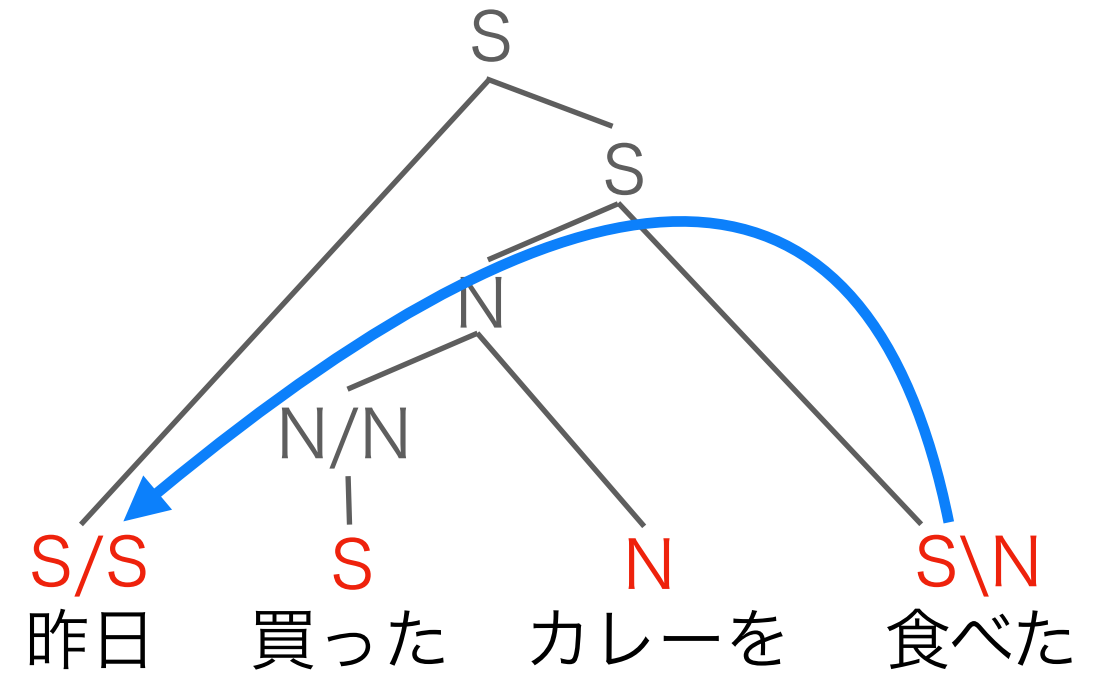
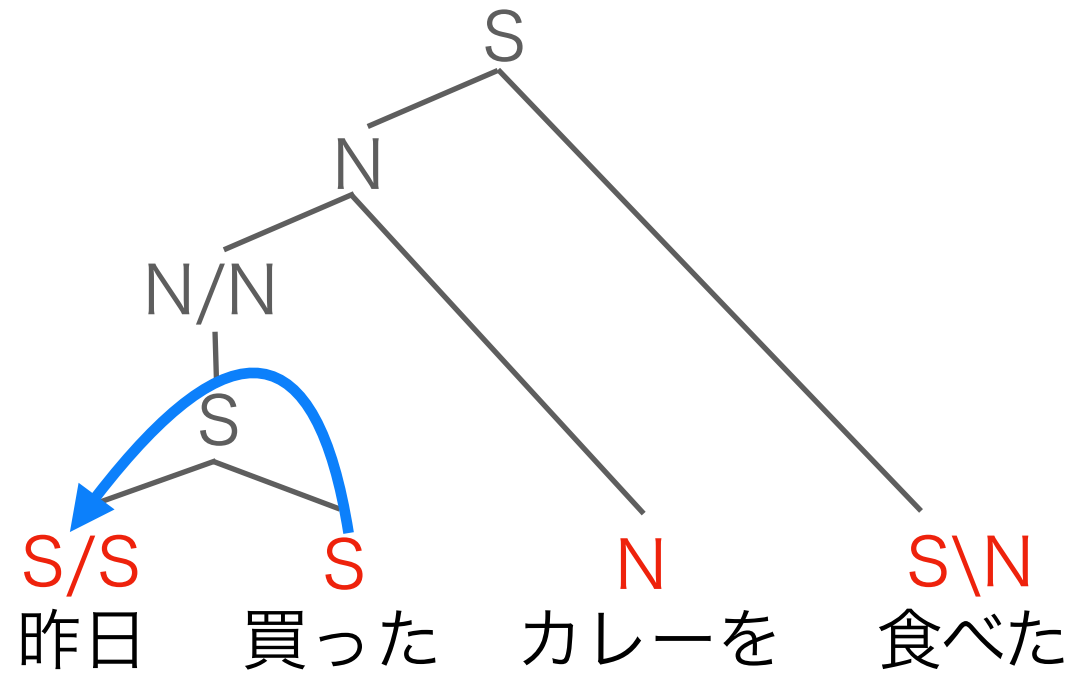
# Category-factoredモデルは日本語に不適

- 係り先が曖昧なカテゴリが存在： $S/S$  (連体修飾、副詞)



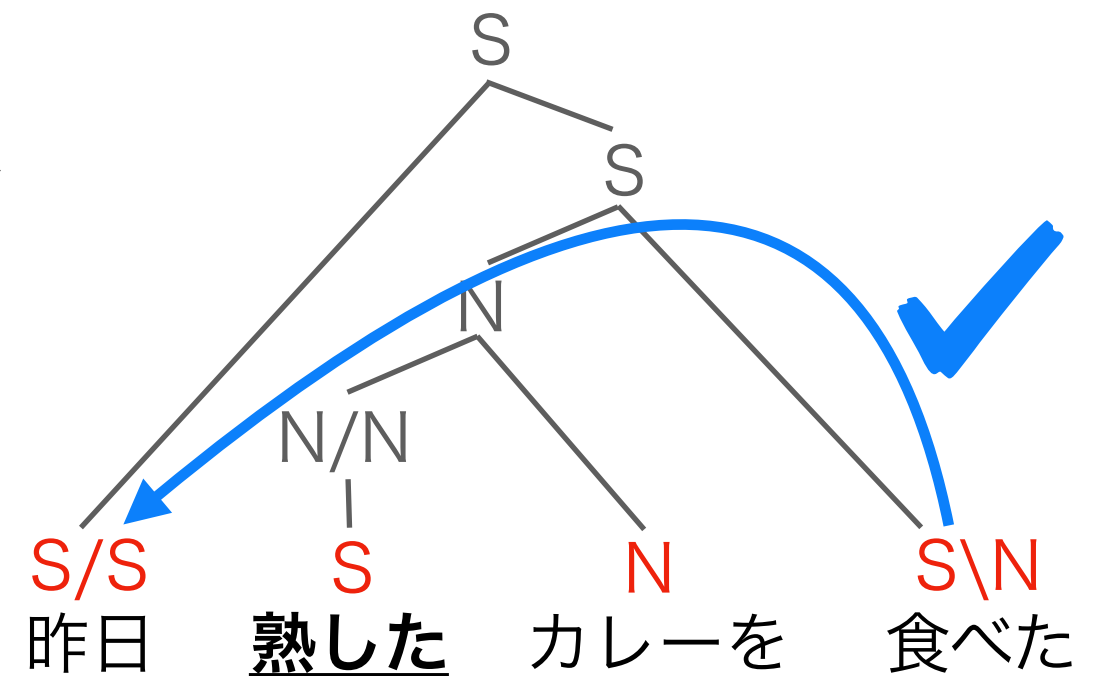
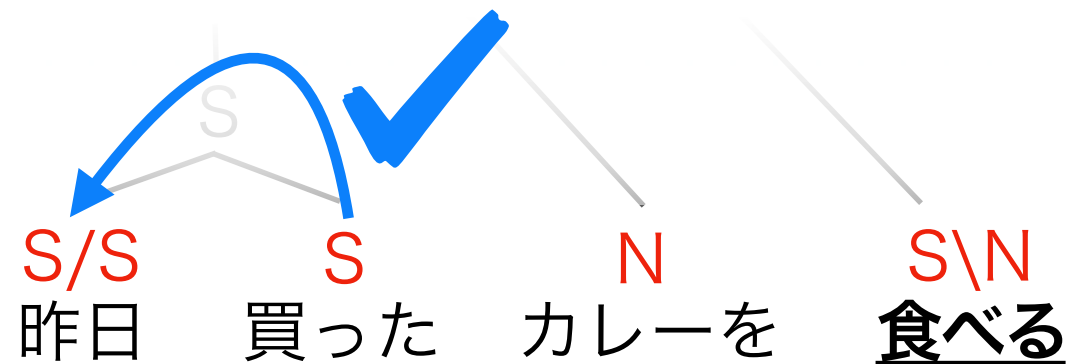
# Category-factoredモデルは日本語に不適

- 係り先が曖昧なカテゴリが存在：S/S (連体修飾、副詞)



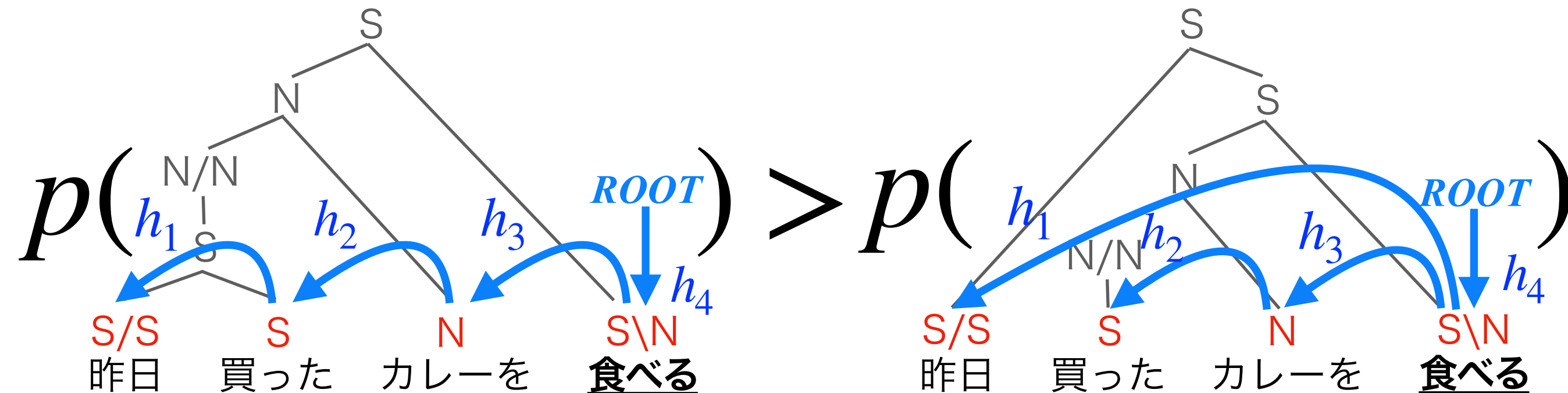
- ・人手ルールの改良では**対処困難**

- ・「昨日」と動詞の時制の一致など



# 提案手法

# 提案：係り受け構造の尤もらしさを明示的にモデル化



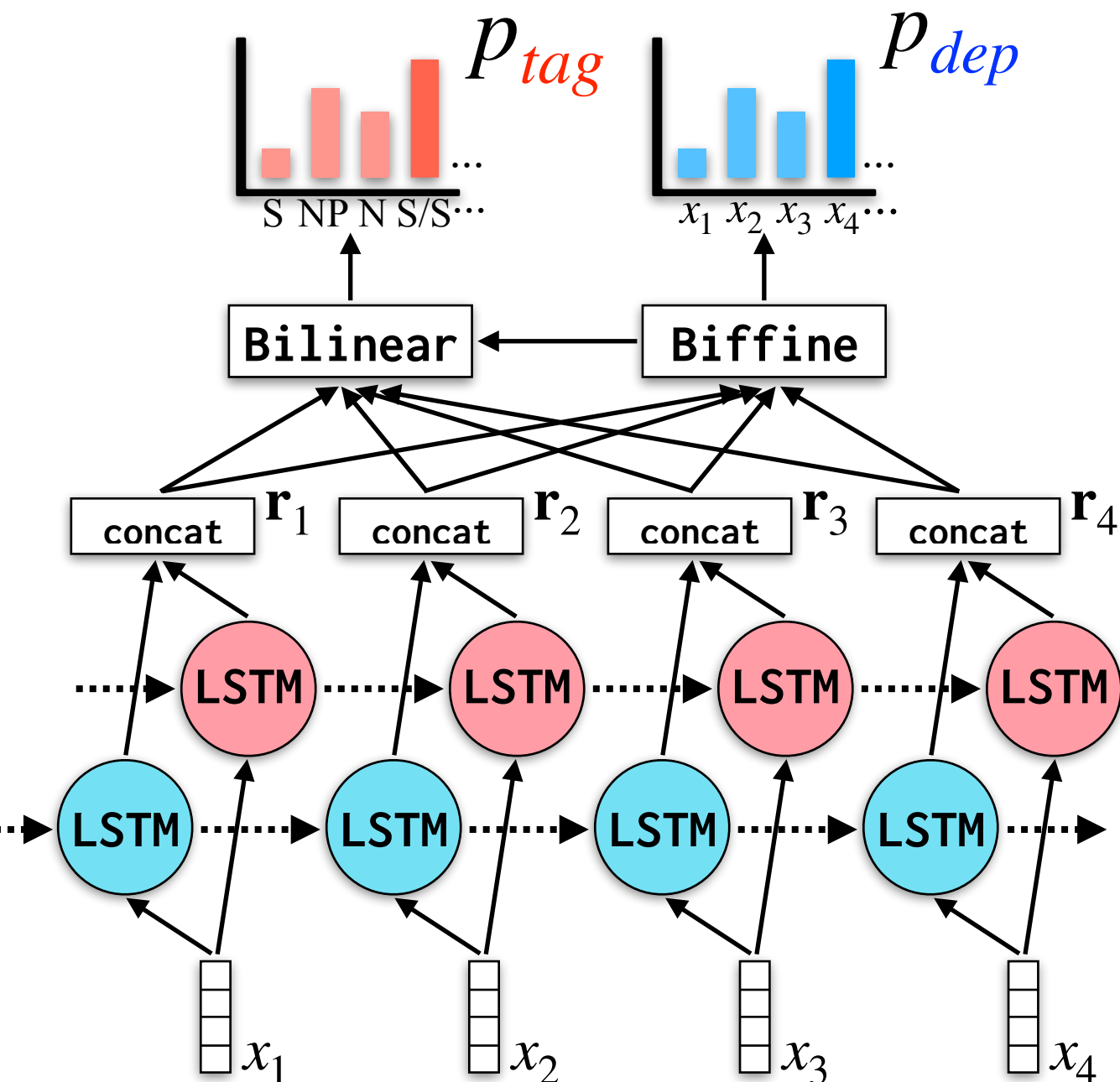
- Category & Dependency-factoredモデル

$$p(\mathbf{y} | \mathbf{x}) = \prod p_{tag}(c_i | \mathbf{x}) \times \prod p_{dep}(h_i | \mathbf{x})$$

- 係り受け構造を用いて終端以上の構造の良さを考慮
- $p_{tag}$ と $p_{dep}$ の局所的な項の積に分解可能
  - Category-factoredモデル同様にA\*構文解析が可能
  - すべての単語について $p_{tag}$ と $p_{dep}$ は事前に計算可

高速

# $p_{tag}$ と $p_{dep}$ の深層学習モデル



- 双方向LSTMの表現ベクトル： $\mathbf{r}_i$
- Biaffine(Dozat+, 2017)で係り受け予測

$$p_{dep}(x_j \rightarrow x_i) \propto \mathbf{r}_i^T W \mathbf{r}_j + \mathbf{r}_i^T \mathbf{u}$$

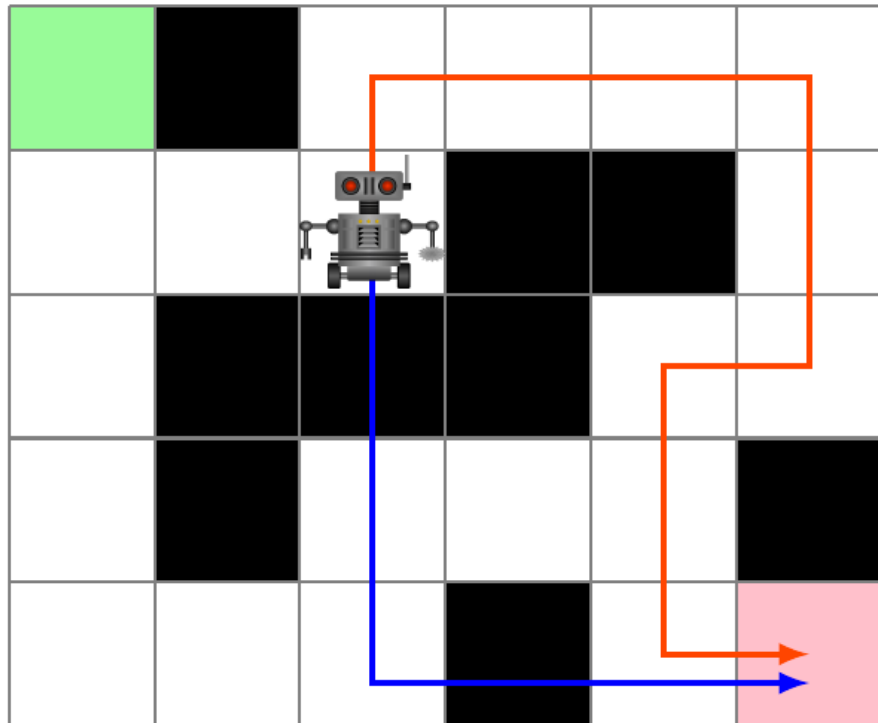
- 係り受け構造を利用しカテゴリ予測

$$p_{tag}(c_i = c) \propto \mathbf{r}_i^T W_c \mathbf{r}_{i\_head}$$

- 学習：負の対数尤度の合計を最適化
- これらはA\*解析前に計算可

# A\*構文解析

## 最短経路問題



遷移先	f
(1,1)	0.1
(2,0)	0.1
(0,1)	0.1
(0,2)	0.9
(3,0)	0.99
...	...

PriorityQueue(f)

$f = g + h$ に基づき探索

$g$  = これまでの経路のコストの合計

$h$  = ゴールまでのコストの推定値(下界)

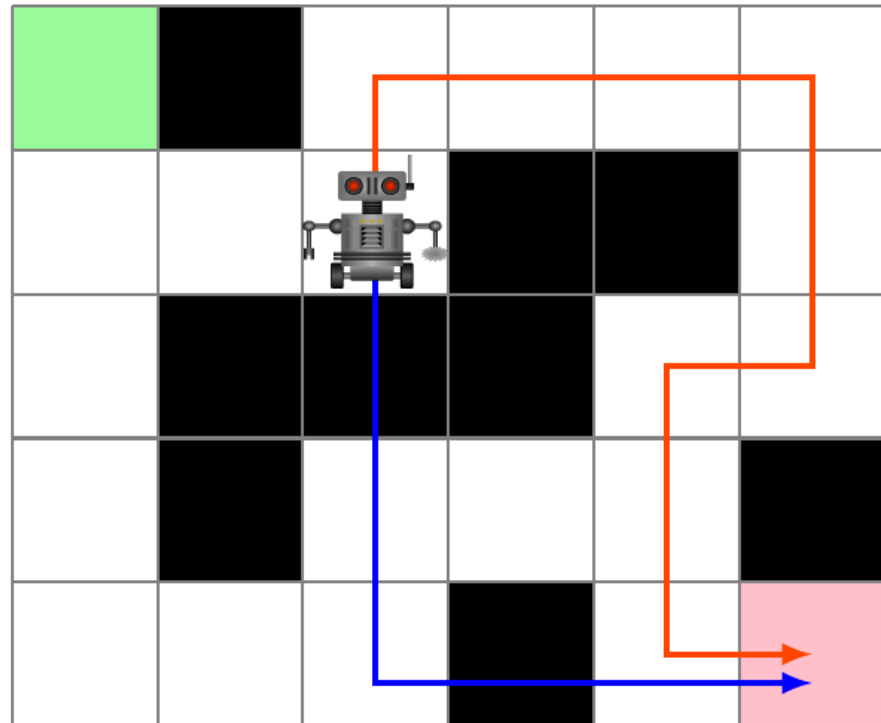
- e.g. マンハッタン距離

## チャート構文解析



# A\*構文解析

## 最短経路問題



遷移先	f
(1,1)	0.1
(2,0)	0.1
(0,1)	0.1
(0,2)	0.9
(3,0)	0.99
...	...

PriorityQueue(f)

$f = g + h$ に基づき探索

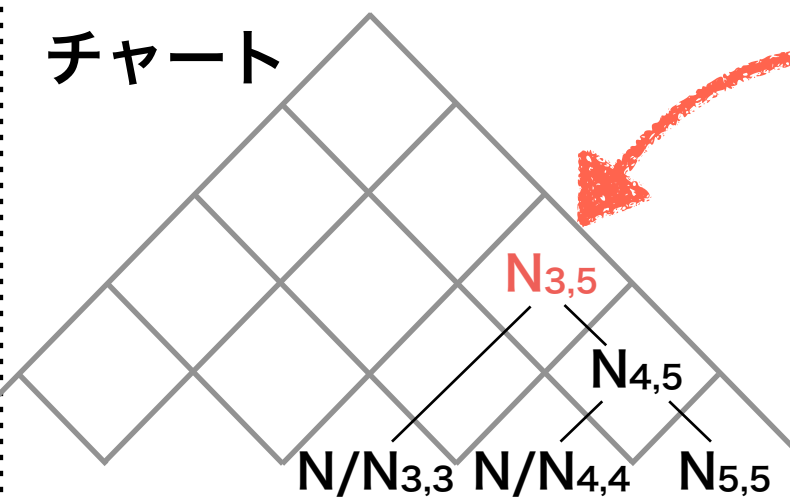
$g$  = これまでの経路のコストの合計

$h$  = ゴールまでのコストの推定値(下界)

• e.g. マンハッタン距離

## チャート構文解析

チャート



エッジ	f
<b>N<sub>3,5</sub></b>	0.1
N <sub>1,1</sub>	0.1
S\N/N <sub>2,2</sub>	0.1
N <sub>4,4</sub>	0.9
S\N <sub>2,2</sub>	0.99
...	...

PriorityQueue(f)

$f = g + h$ に基づき探索

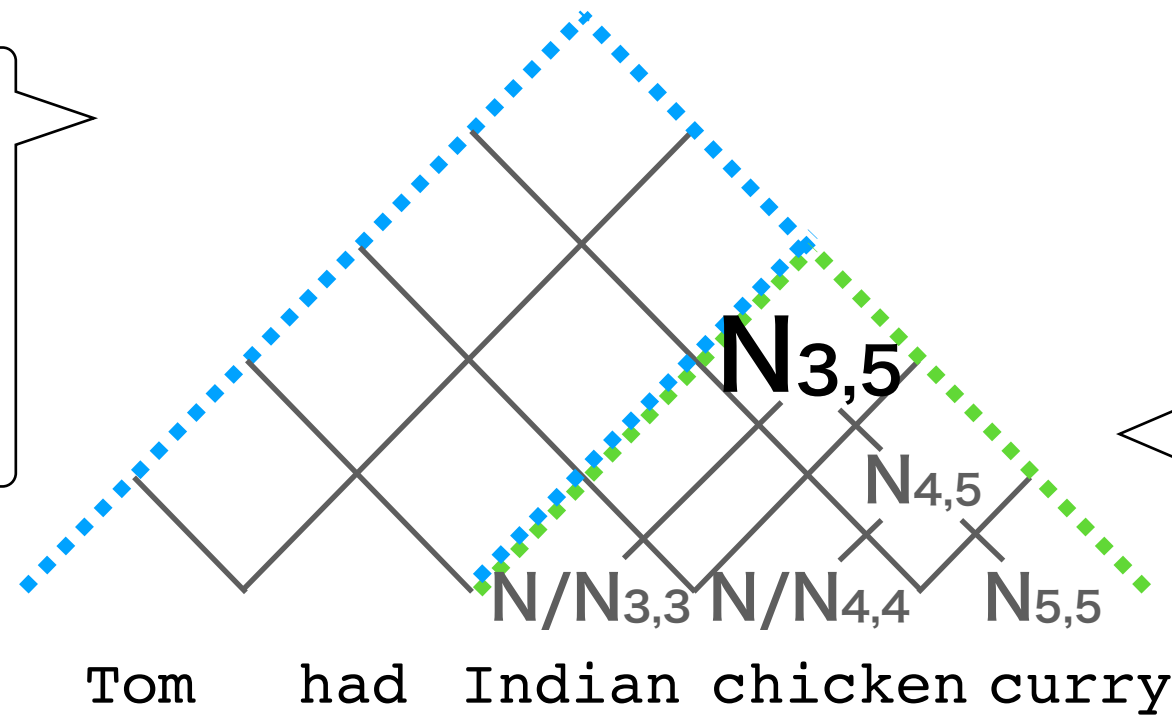
$g$  = エッジのコスト (= 負の対数確率)

エッジ: 部分木を表すデータ構造

$h$  = ゴール (= 構文木) までのコストの下界

# チャート構文解析における $g$ と $h$

$h$ : 外側確率  
の推定値  
s.t.  $h \leq h^*$



$g$ : 部分木の  
負の対数確率  
 $-\log p(\mathbf{y}_{3,5} | \mathbf{x})$

$f = g + h$  に基づき探索

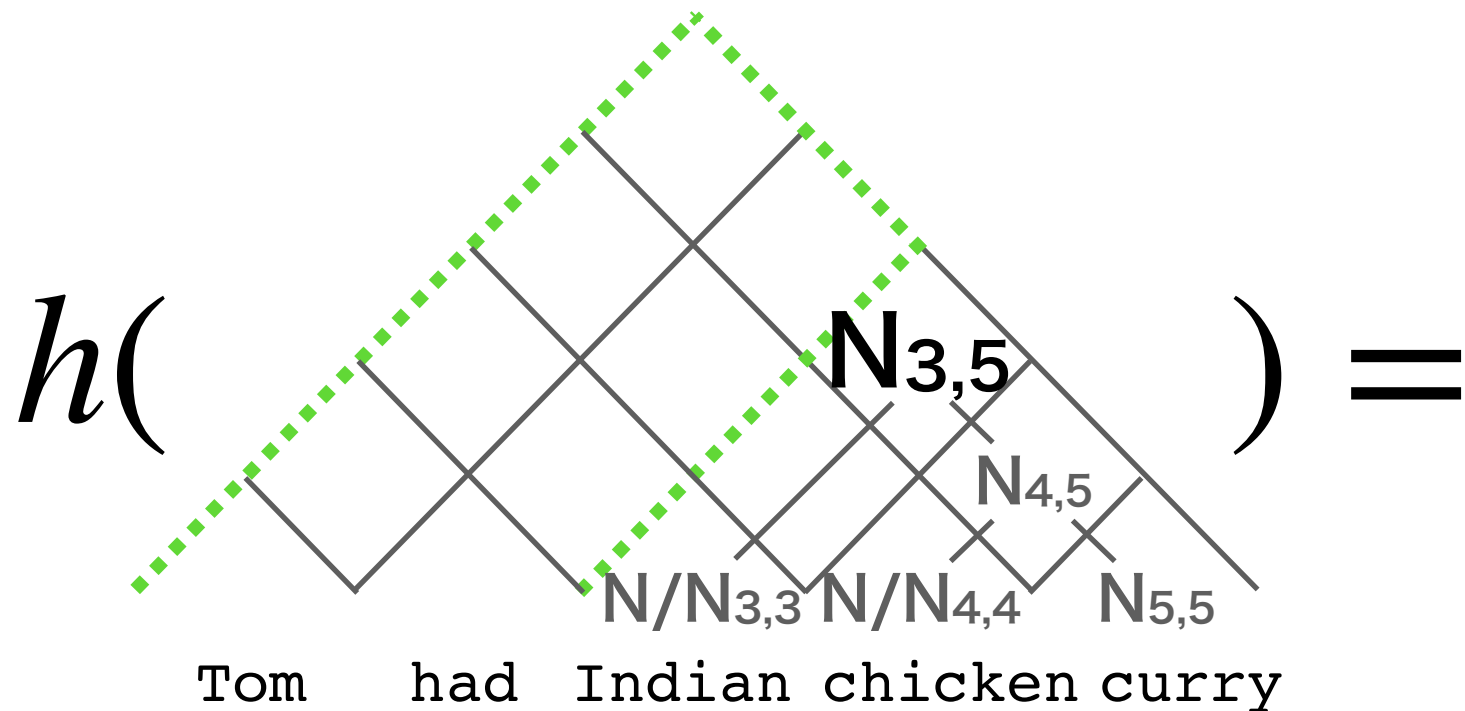
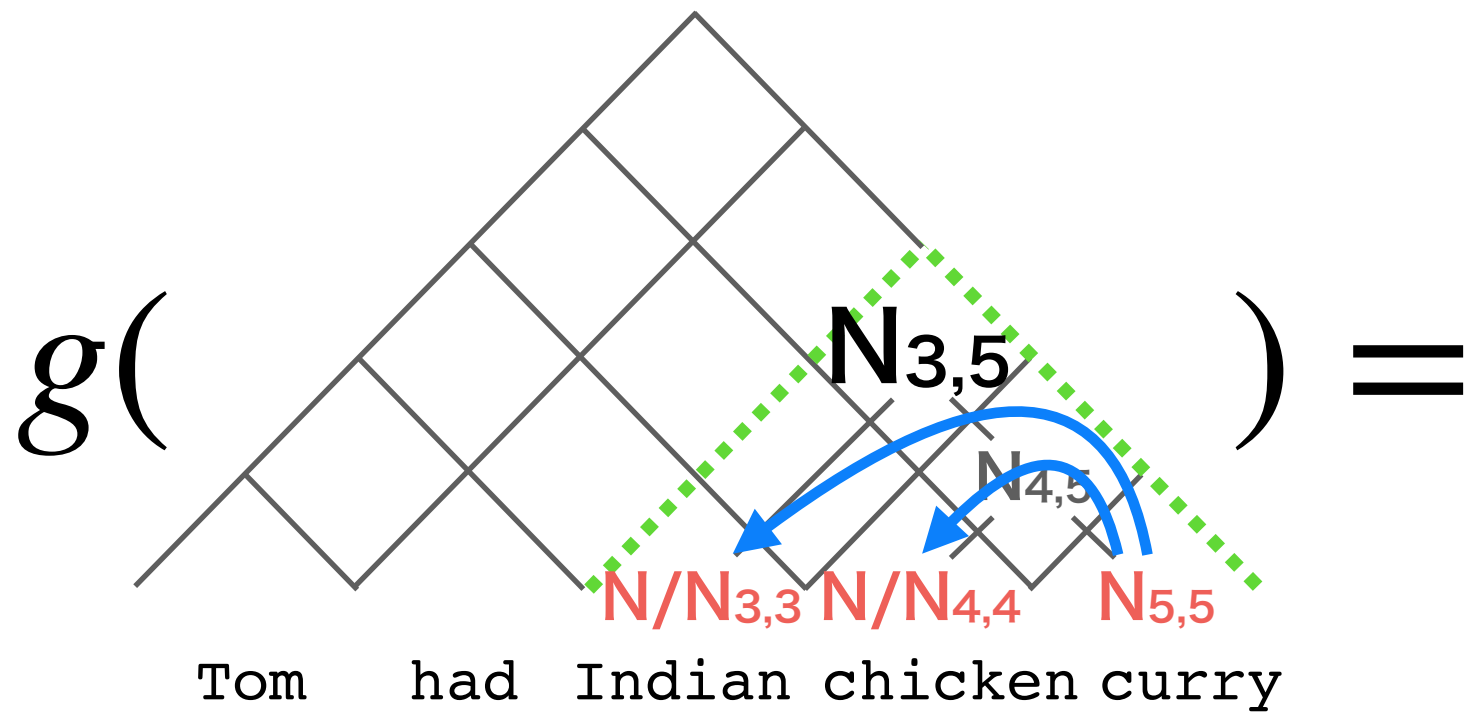
$g$  = 部分木の**コスト**(=負の対数確率)

$h$  = ゴール(=構文木)までの**コストの推定値(下界)**

s.t.  $h \leq h^*$  (真のコスト)

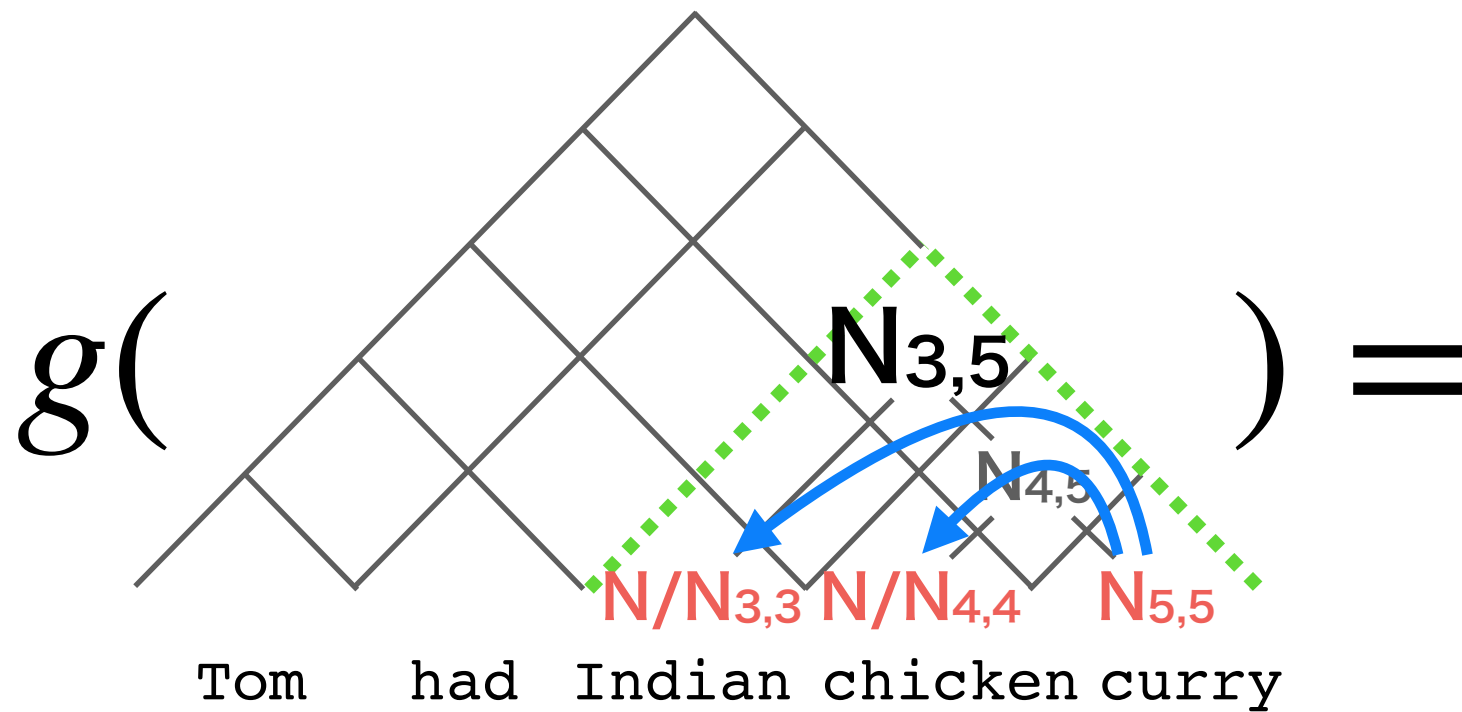
# チャート構文解析における $g$ と $h$

- エッジ  $N_{3,5}$  について...

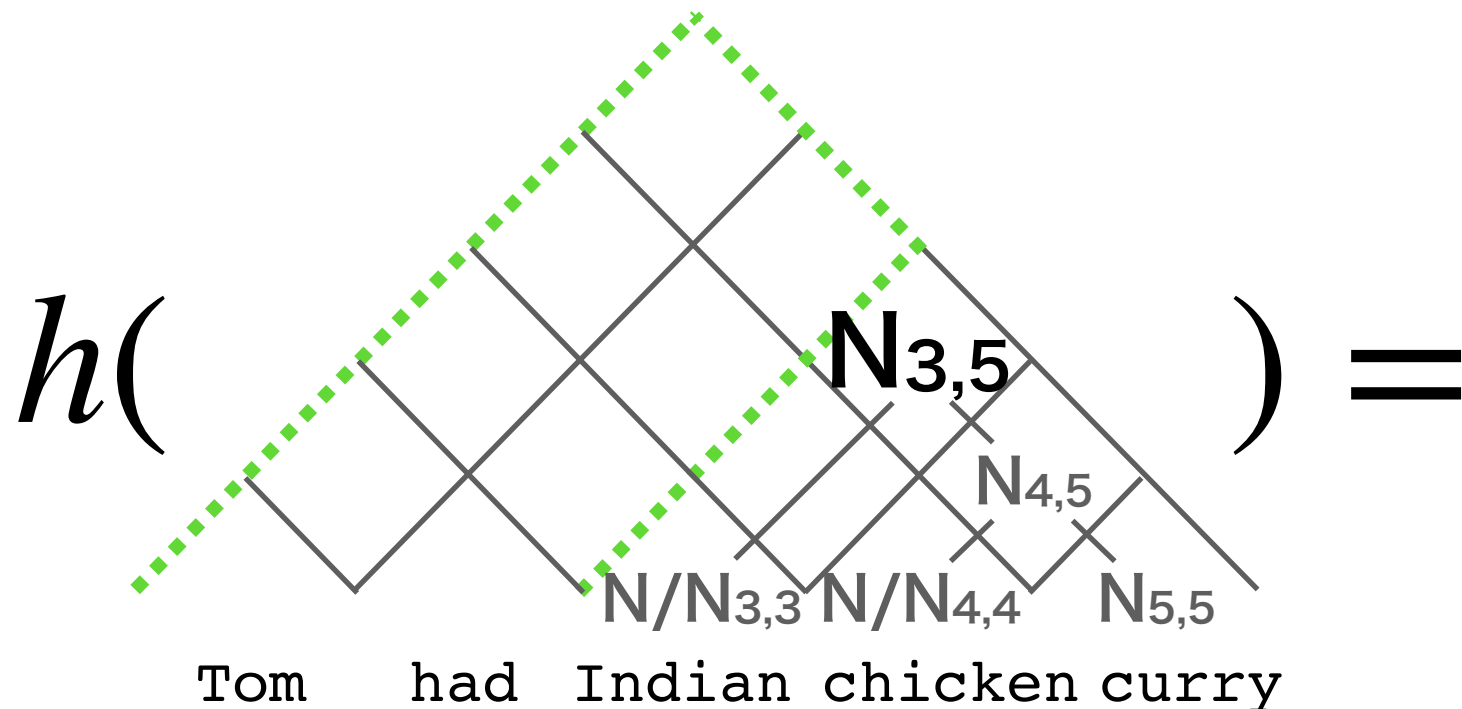


# チャート構文解析における $g$ と $h$

- エッジ  $N_{3,5}$  について...

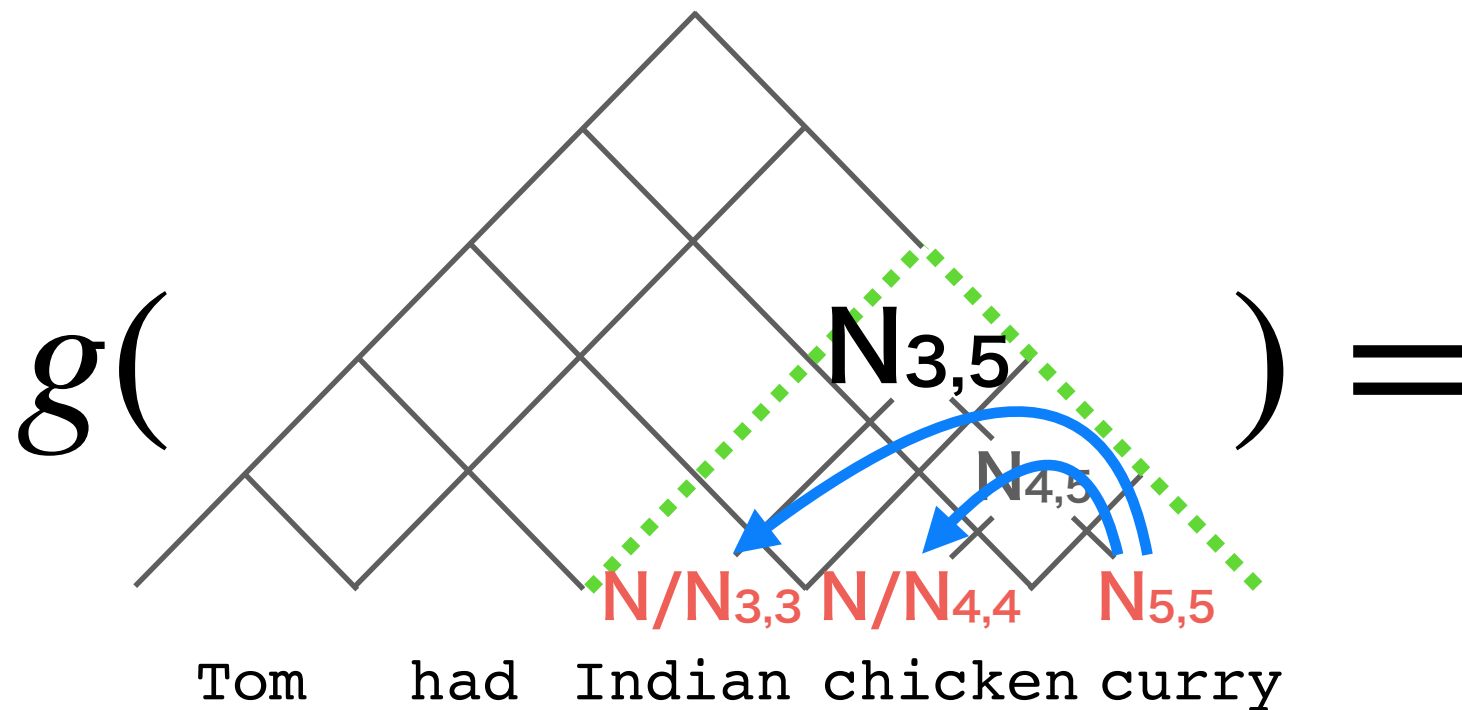


$$\begin{aligned}
 & -\log P_{tag}^{tag}(N/N_{3,3}) \\
 & -\log P_{tag}^{tag}(N/N_{4,4}) \\
 & -\log P_{tag}^{tag}(N_{5,5}) \\
 & -\log P_{dep}^{dep}(\text{chicken curry}) \\
 & -\log P_{dep}^{dep}(\text{Indian curry})
 \end{aligned}$$

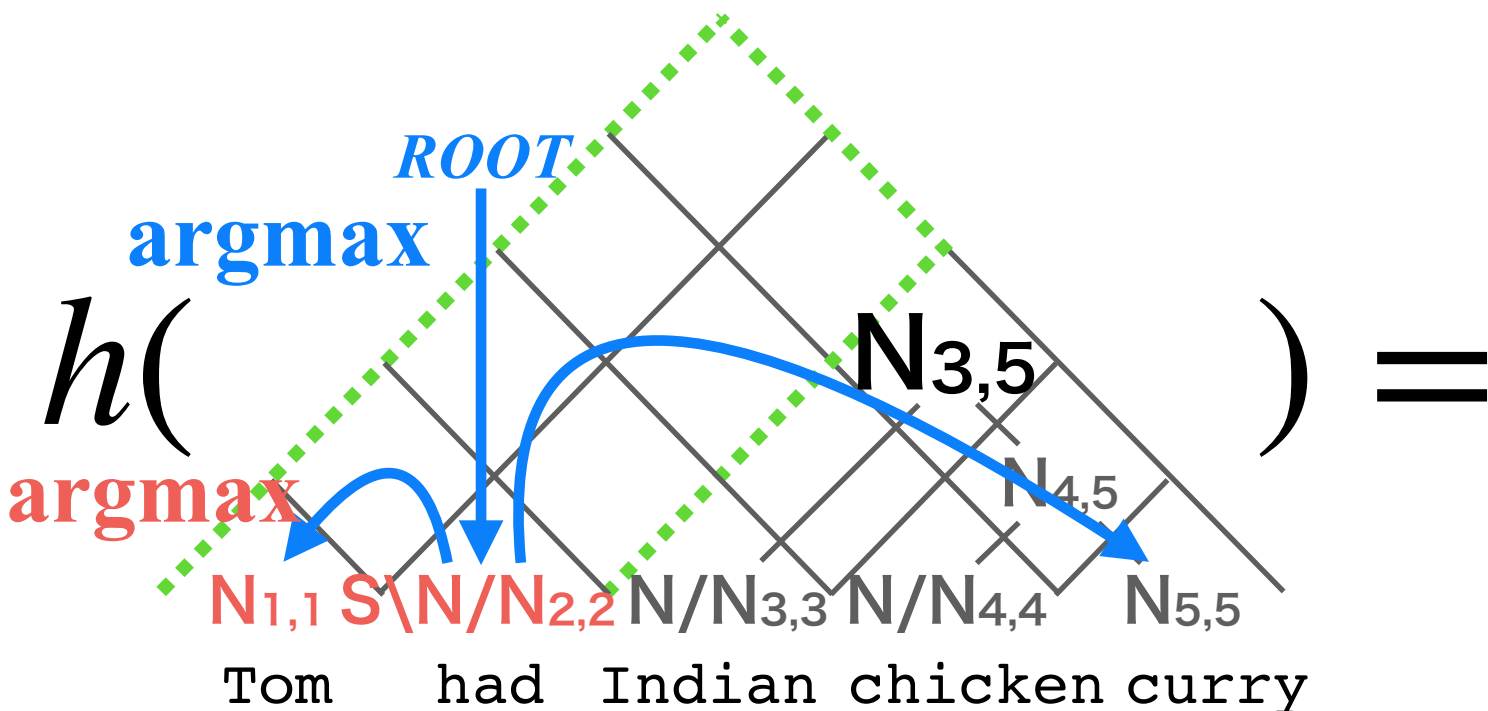


# チャート構文解析における $g$ と $h$

- エッジ  $N_{3,5}$  について...

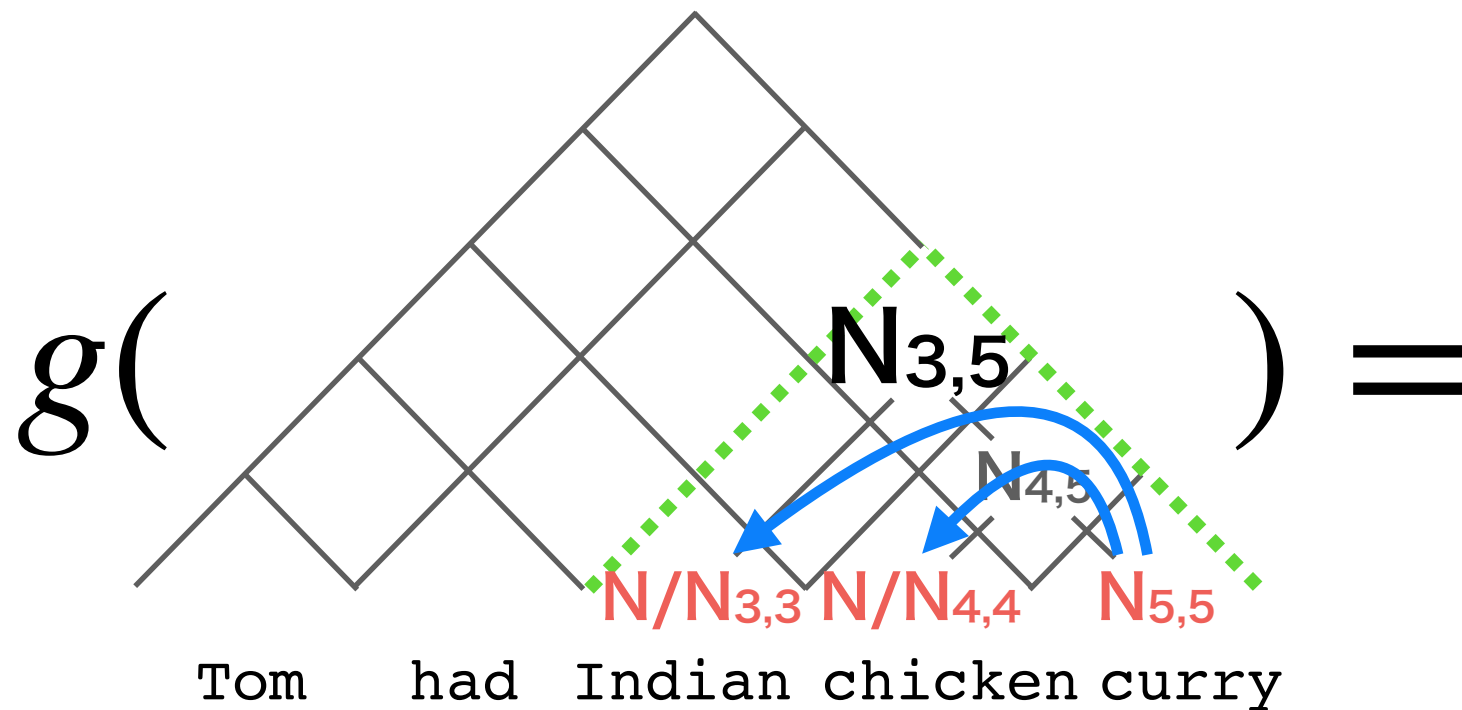


$$\begin{aligned}
 & -\log P_{tag}^{red}(N/N_{3,3}) \\
 & -\log P_{tag}^{red}(N/N_{4,4}) \\
 & -\log P_{tag}^{red}(N_{5,5}) \\
 & -\log P_{dep}^{blue}(\text{chicken} \rightarrow \text{curry}) \\
 & -\log P_{dep}^{blue}(\text{Indian} \rightarrow \text{curry})
 \end{aligned}$$

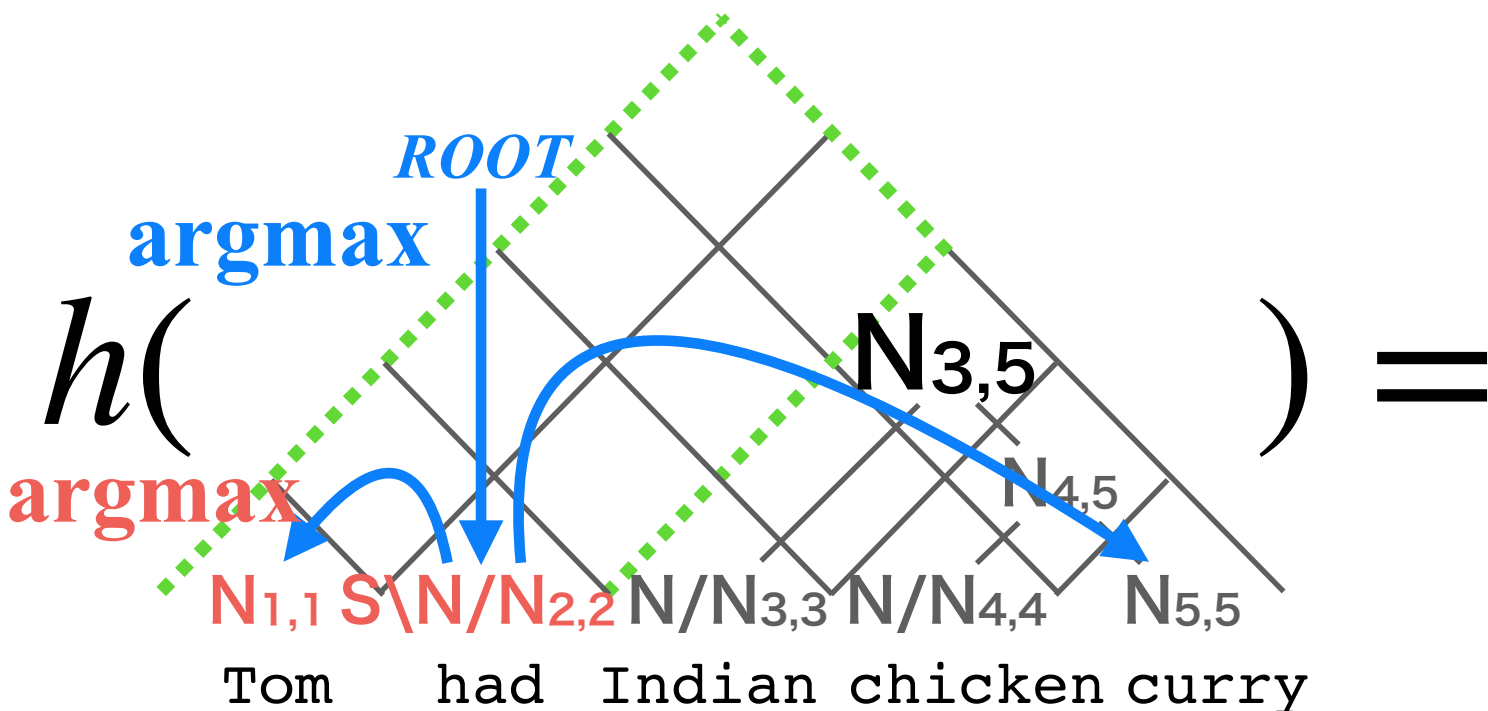


# チャート構文解析における $g$ と $h$

- エッジ  $N_{3,5}$  について...



$$\begin{aligned}
 & -\log P_{tag}^{tag}(N/N_{3,3}) \\
 & -\log P_{tag}^{tag}(N/N_{4,4}) \\
 & -\log P_{tag}^{tag}(N_{5,5}) \\
 & -\log P_{dep}^{dep}(\text{chicken curry}) \\
 & -\log P_{dep}^{dep}(\text{Indian curry})
 \end{aligned}$$



$$\begin{aligned}
 & -\log P_{tag}^{tag}(N_{1,1}) \\
 & -\log P_{tag}^{tag}(S\N/N_{2,2}) \\
 & -\log P_{dep}^{dep}(\text{Tom had}) \\
 & -\log P_{dep}^{dep}(\text{had ROOT}) \\
 & -\log P_{dep}^{dep}(\text{curry had})
 \end{aligned}$$

# 計算は簡単

$$g(\text{Tom had Indian chicken curry}) =$$

Tom had Indian chicken curry

$$= g(\text{Indian chicken curry})$$

Indian chicken curry

$$+ g(\text{Indian chicken curry}) - \log P_{dep}(\text{Indian} \rightarrow \text{curry})$$

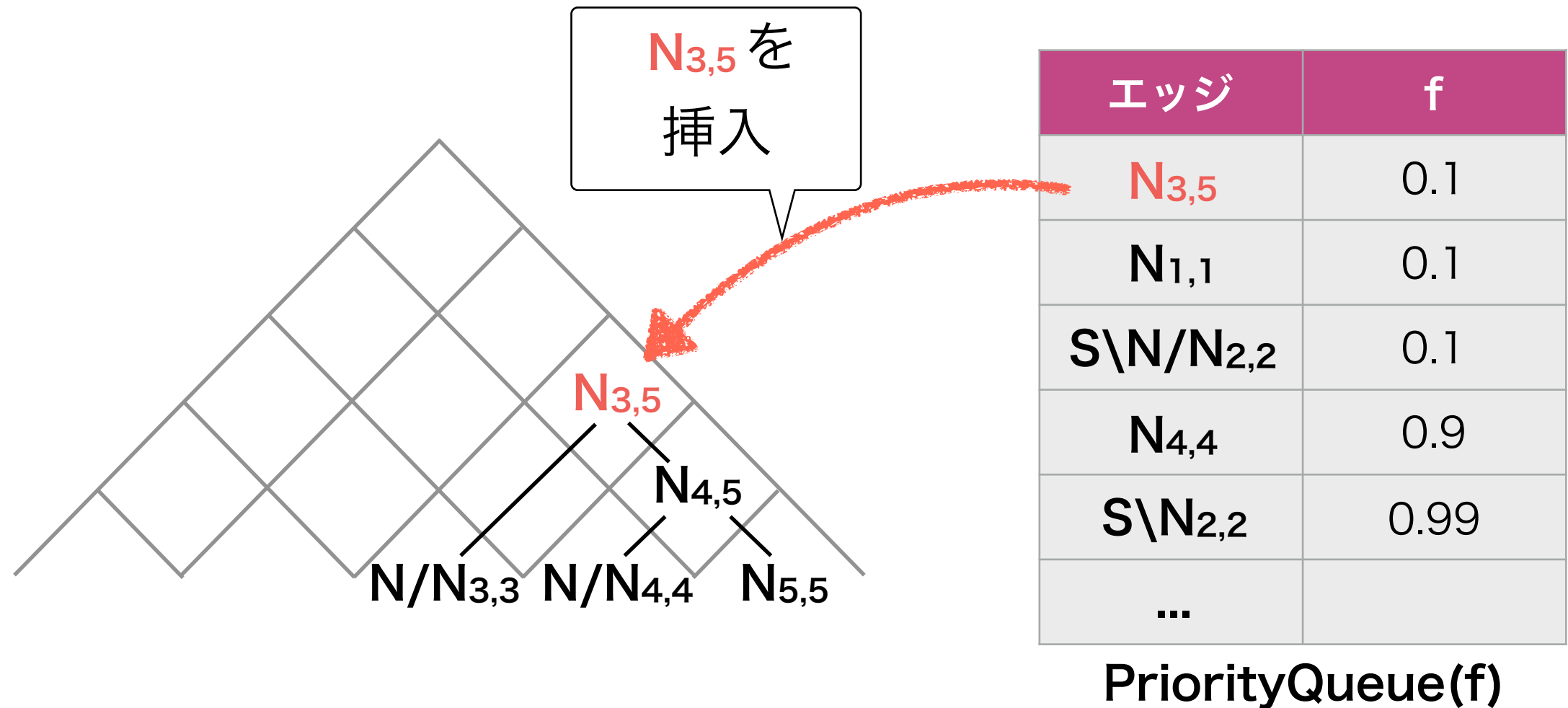
Indian chicken curry

# 計算は簡単

$$\begin{aligned}
 &g( \text{Tom had Indian chicken curry} ) = \\
 &\quad -\log P_{tag}(N/N_{3,3}) \\
 &\quad -\log P_{tag}(N/N_{4,4}) \\
 &\quad -\log P_{tag}(N_{5,5}) \\
 &\quad -\log P_{dep}(\text{chicken} \rightarrow \text{curry}) \\
 &\quad -\log P_{dep}(\text{Indian} \rightarrow \text{curry}) \\
 \\
 &= g( \text{Indian chicken curry} ) \\
 &\quad + g( \text{Indian chicken curry} ) - \log P_{dep}(\text{Indian} \rightarrow \text{curry})
 \end{aligned}$$

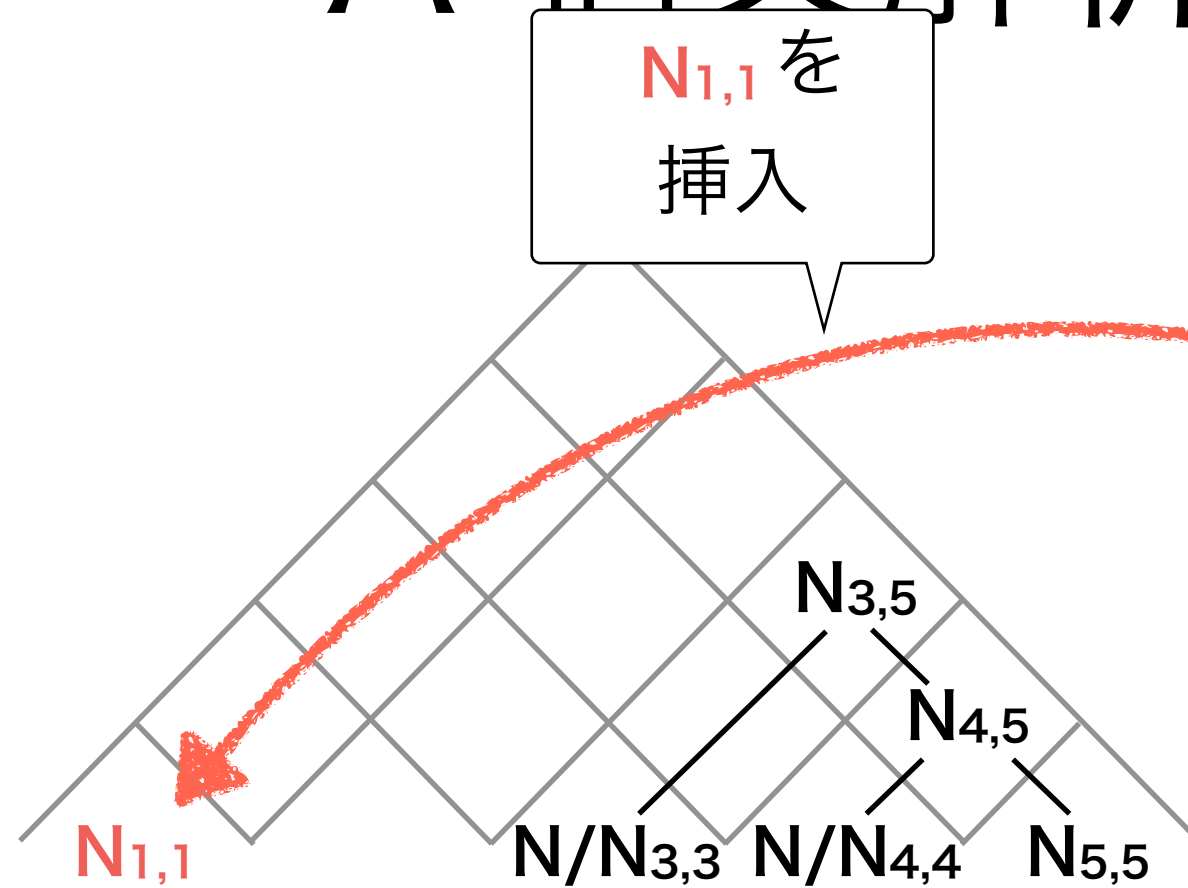


# A\*構文解析の動作



- f値が最も低いエッジをキューからポップしチャートに挿入
- 新しいエッジを構築できればキューに追加

# A\*構文解析の動作

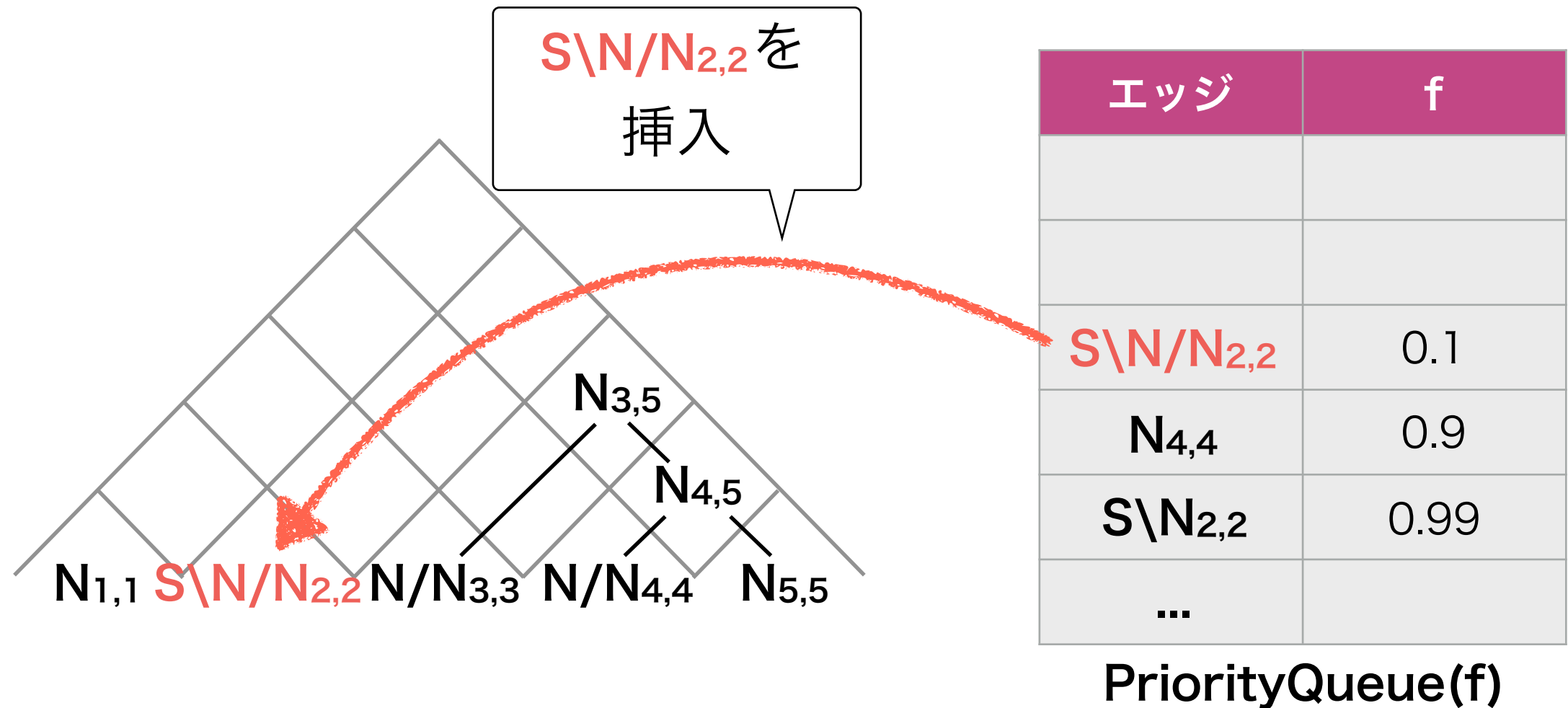


エッジ	f
$N_{1,1}$	0.1
$S \setminus N / N_{2,2}$	0.1
$N_{4,4}$	0.9
$S \setminus N_{2,2}$	0.99
...	

PriorityQueue(f)

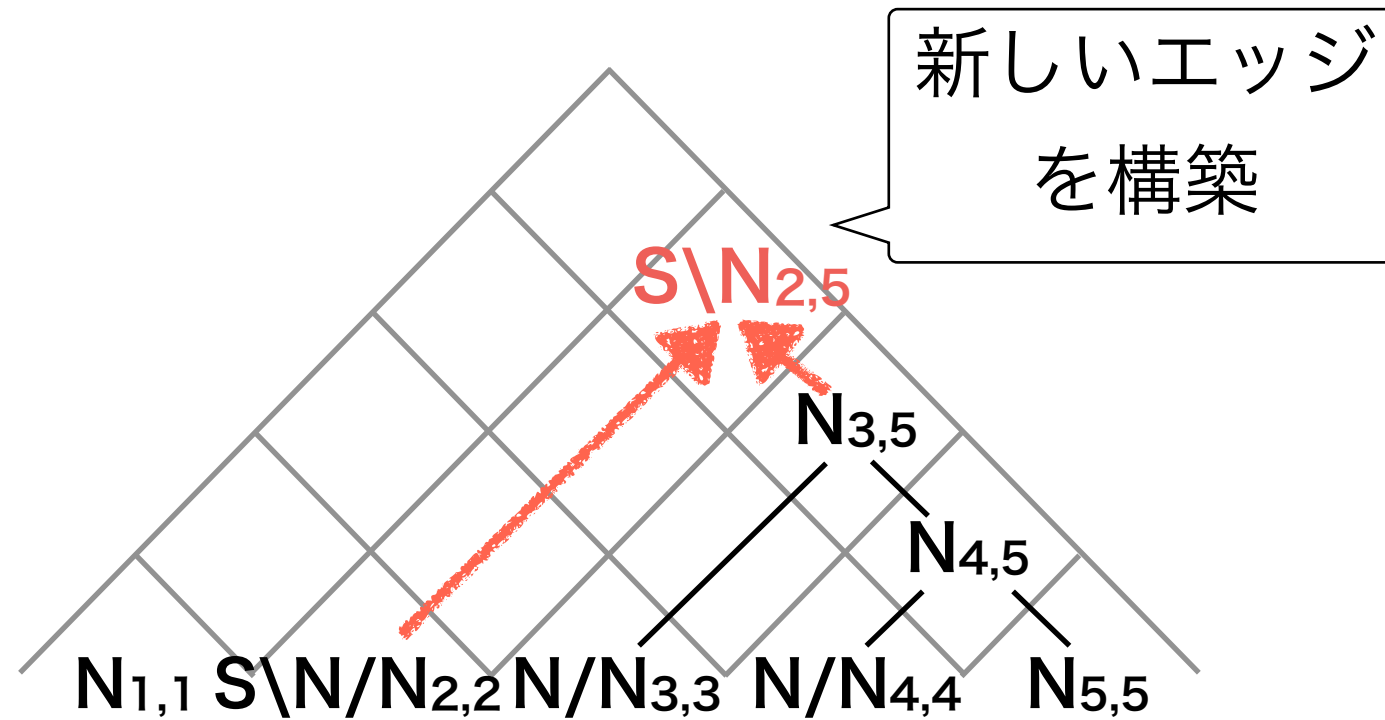
- f値が最も低いエッジをキューからポップしチャートに挿入
- 新しいエッジを構築できればキューに追加

# A\*構文解析の動作



- f値が最も低いエッジをキューからポップしチャートに挿入
- 新しいエッジを構築できればキューに追加

# A\*構文解析の動作

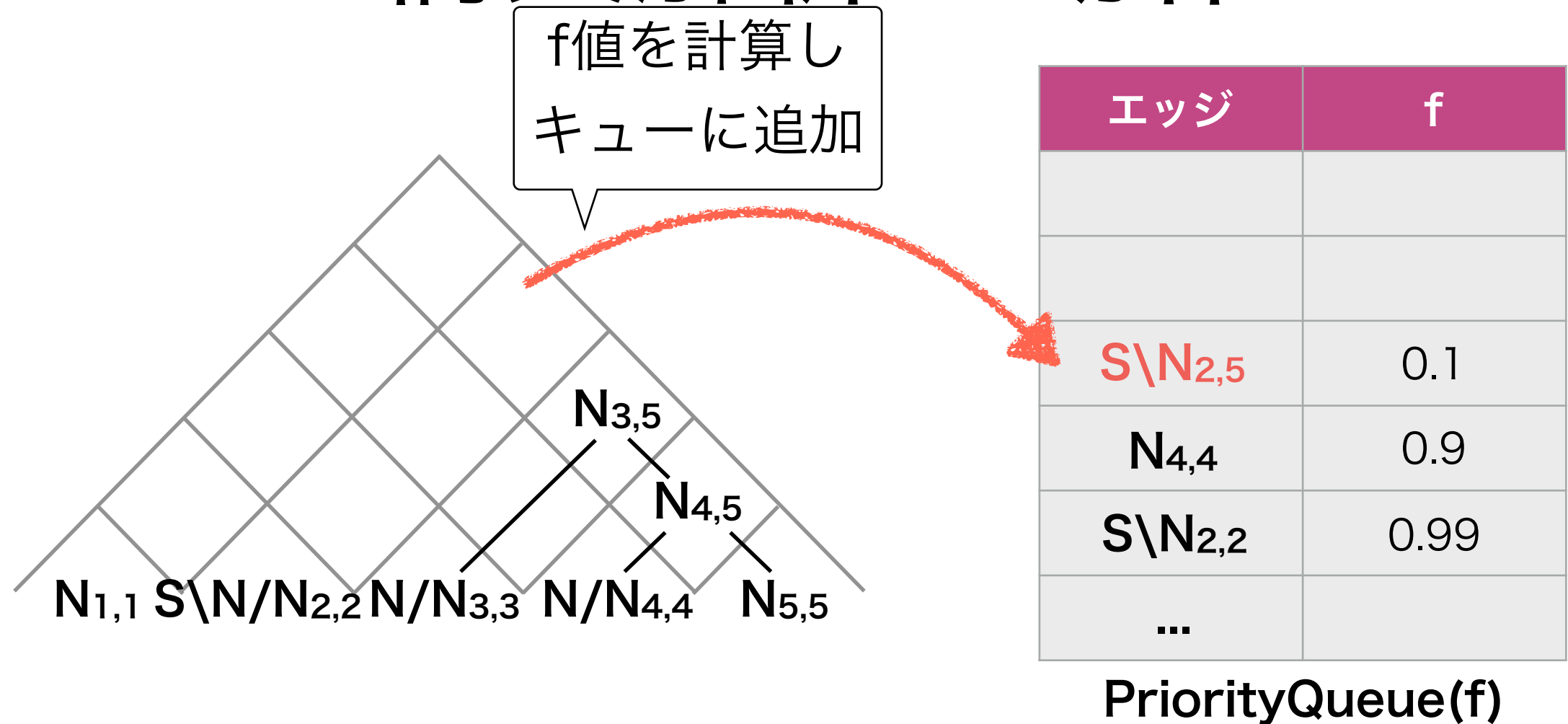


エッジ	f
$N_{4,4}$	0.9
$S \backslash N_{2,2}$	0.99
...	

PriorityQueue(f)

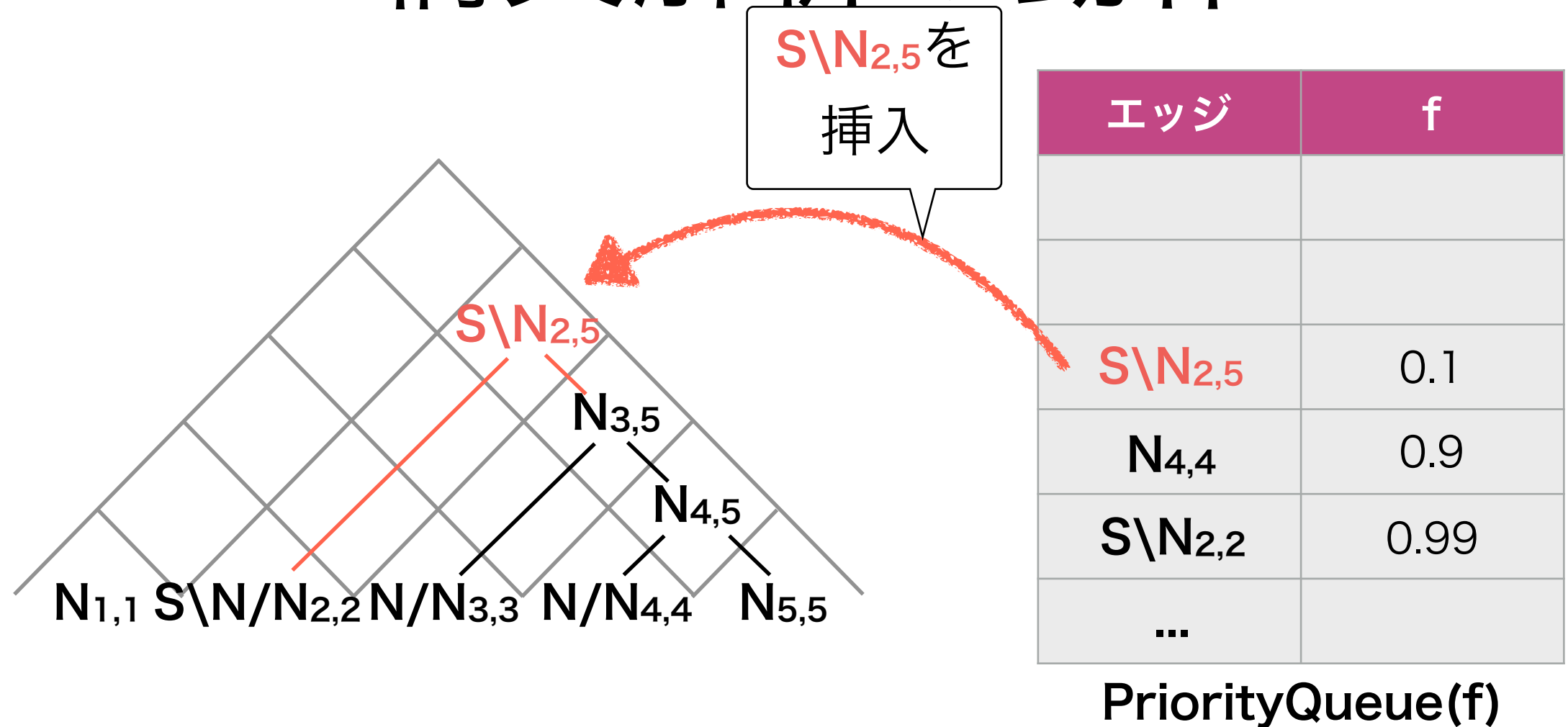
- f値が最も低いエッジをキューからポップしチャートに挿入
- 新しいエッジを構築できればキューに追加

# A\*構文解析の動作



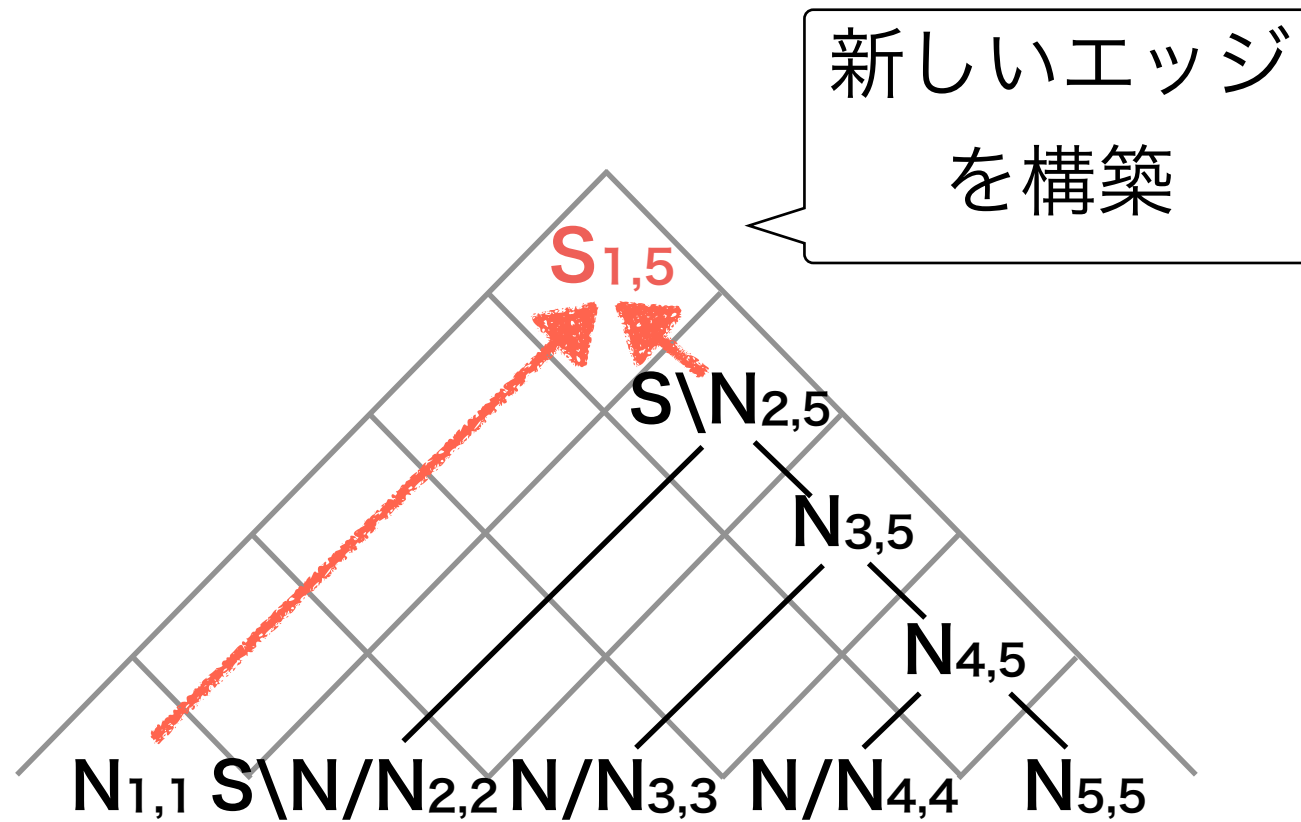
- f値が最も低いエッジをキューからポップしチャートに挿入
- 新しいエッジを構築できればキューに追加

# A\*構文解析の動作



- f値が最も低いエッジをキューからポップしチャートに挿入
- 新しいエッジを構築できればキューに追加

# A\*構文解析の動作

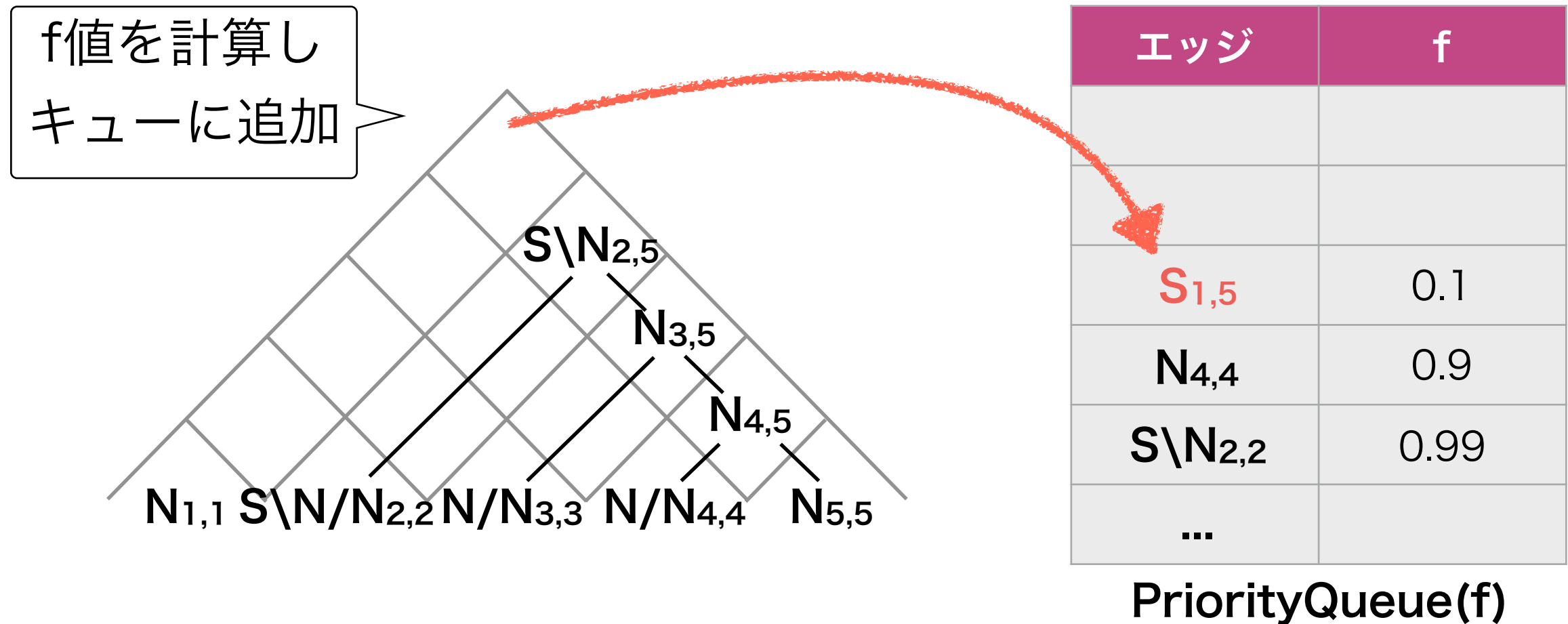


エッジ	f
N4,4	0.9
S\N2,2	0.99
...	

PriorityQueue(f)

- f値が最も低いエッジをキューからポップしチャートに挿入
- 新しいエッジを構築できればキューに追加

# A\*構文解析の動作

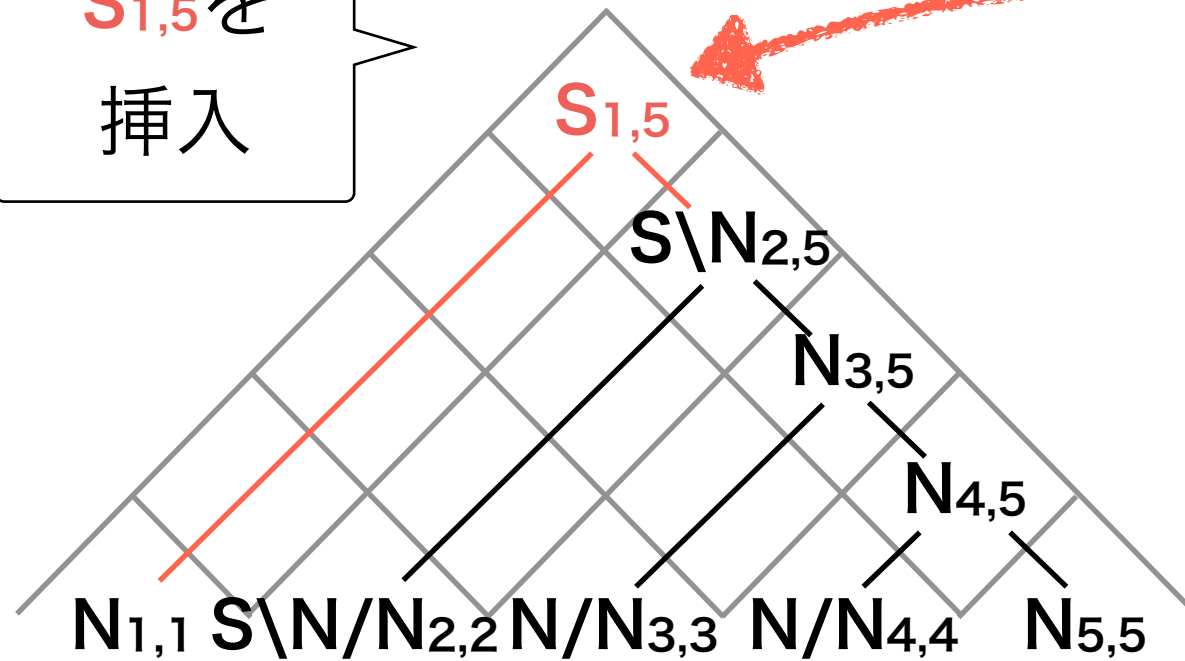


- f値が最も低いエッジをキューからポップしチャートに挿入
- 新しいエッジを構築できればキューに追加



# A\*構文解析の動作

$S_{1,5}$ を  
挿入

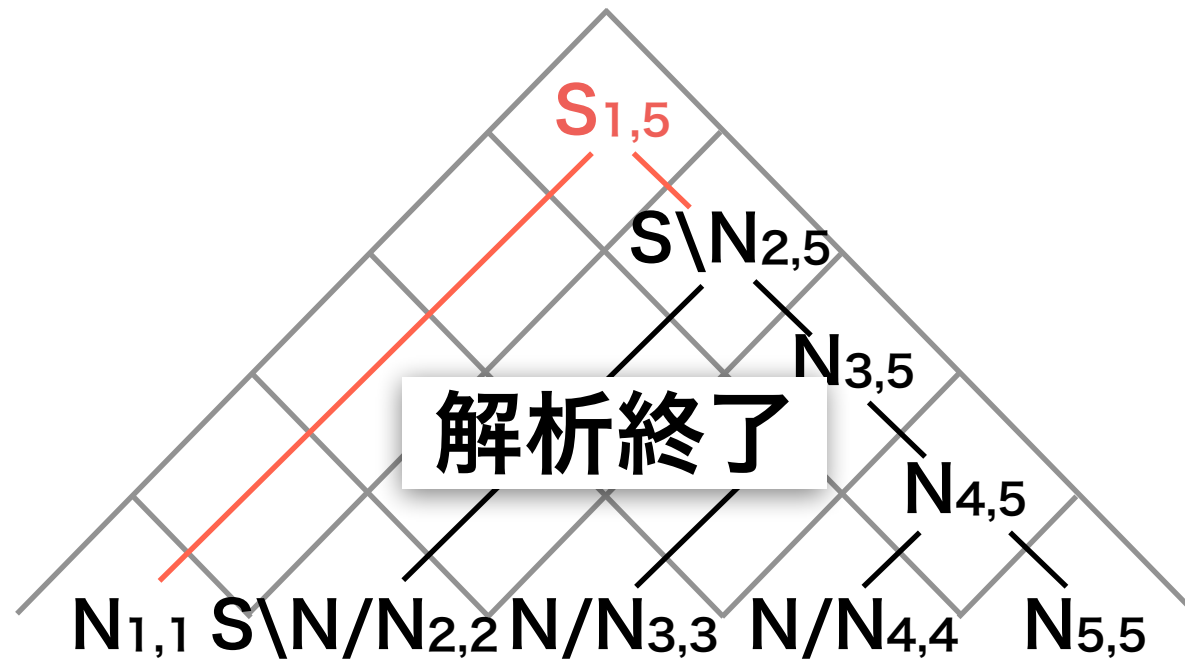


エッジ	f
$S_{1,5}$	0.1
$N_{4,4}$	0.9
$S\backslash N_{2,2}$	0.99
...	

PriorityQueue(f)

- f値が最も低いエッジをキューからポップしチャートに挿入
- 新しいエッジを構築できればキューに追加

# A\*構文解析の動作



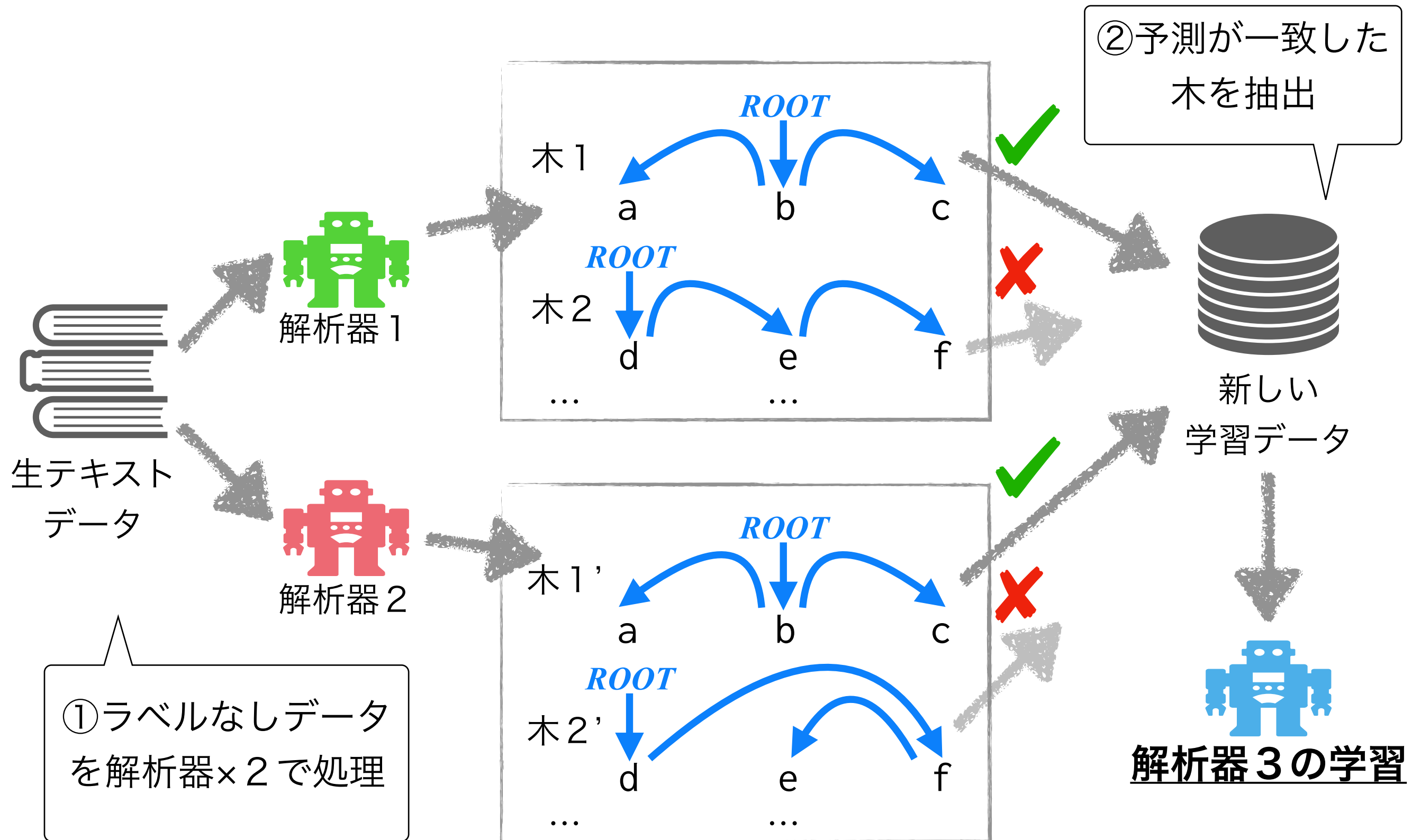
エッジ	f
N <sub>4,4</sub>	0.9
S\N <sub>2,2</sub>	0.99
...	

PriorityQueue(f)

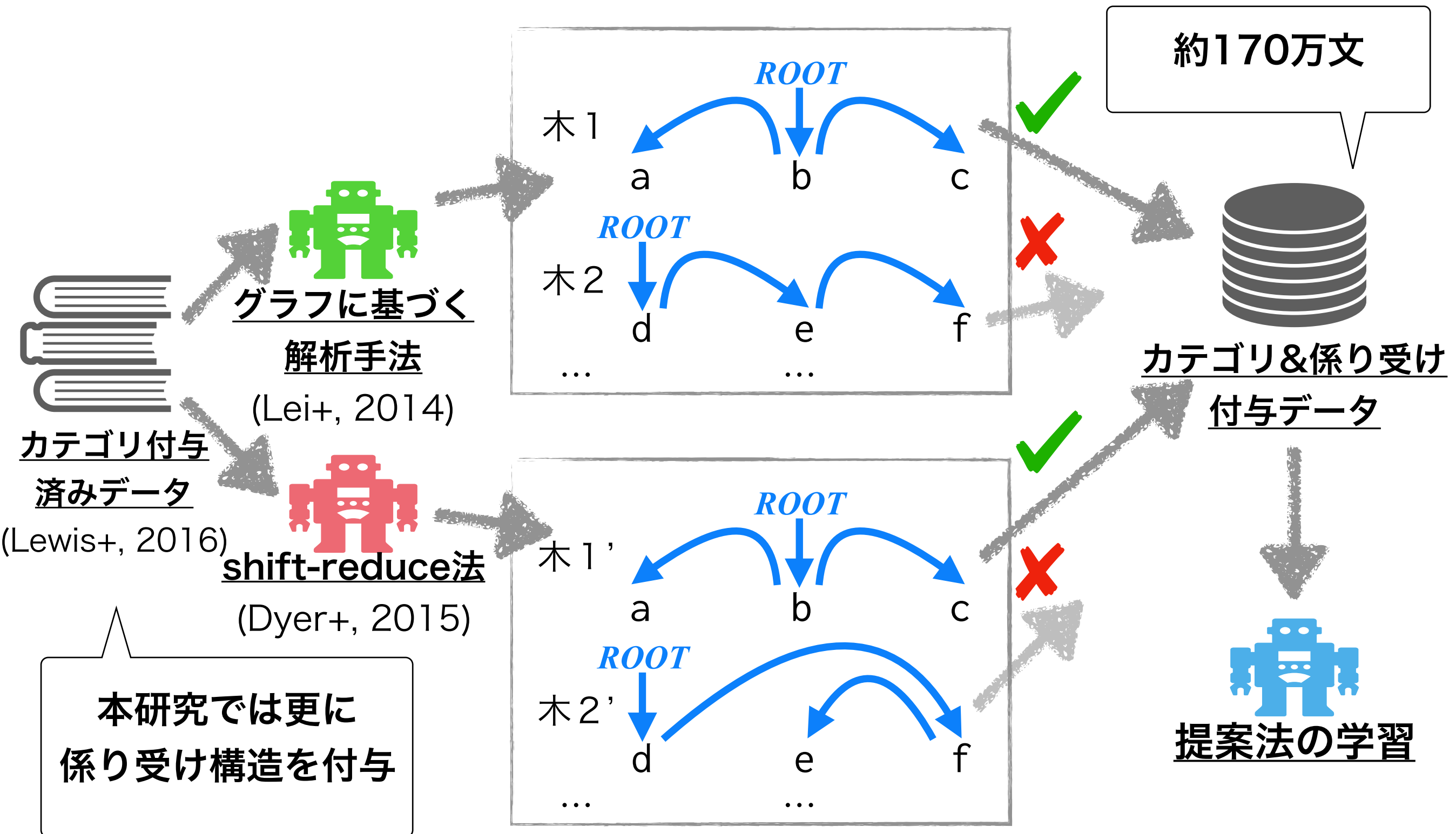
- f値が最も低いエッジをキューからポップしチャートに挿入
- 新しいエッジを構築できればキューに追加
- $P_{tag}$  と  $P_{dep}$  の予測が正確  $\Rightarrow h = h^* \Rightarrow$  無駄な探索なし
- gが単調増加 : エッジの再訪なし(CLOSEリスト不要)

高速

# Tri-trainingによる半教師あり学習 (Weiss+, 2015)



# Tri-trainingの提案法への適用

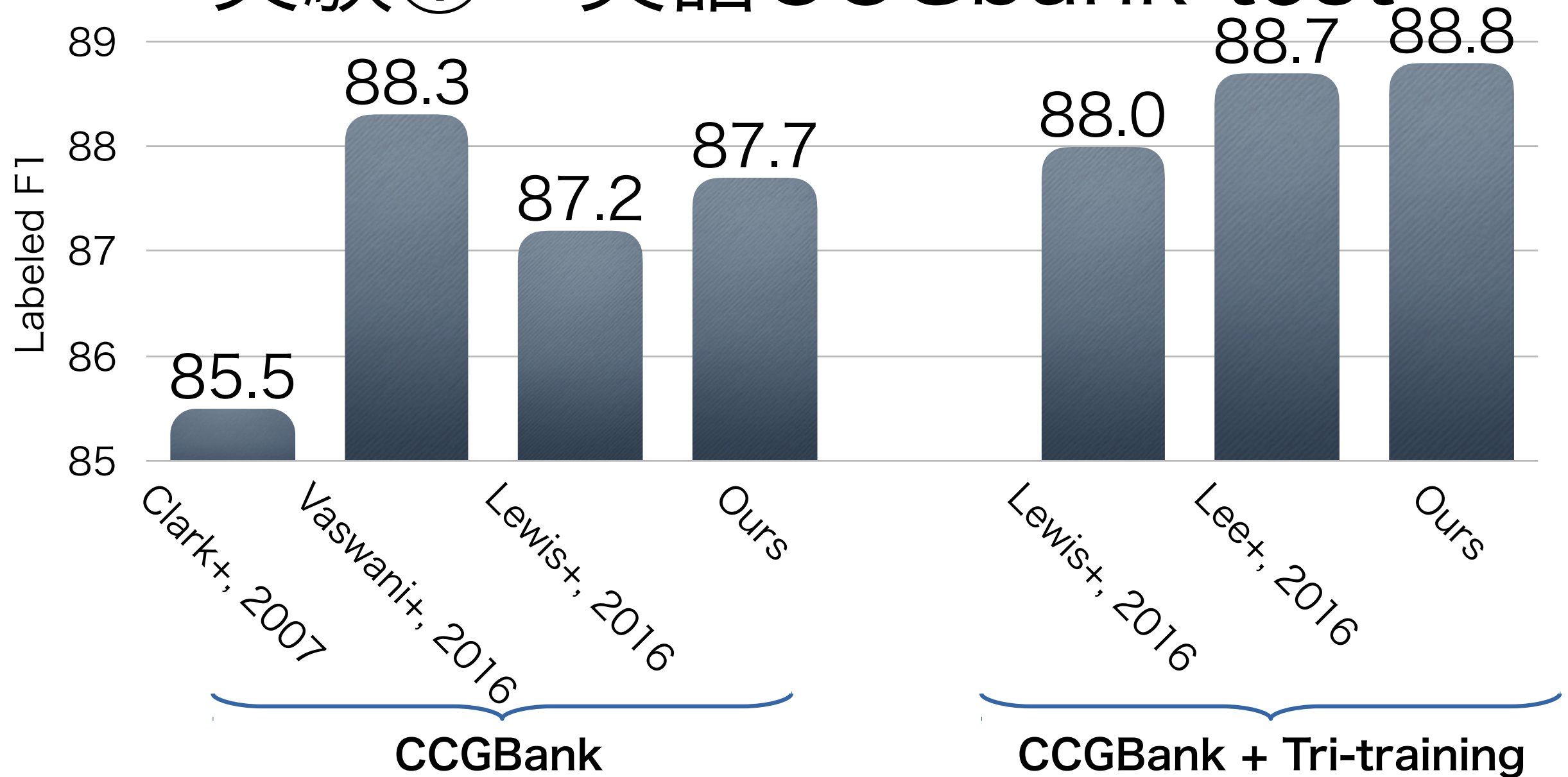


実験

# 実験設定

- モデル： 4層300次元双方向LSTM、 1層100次元MLP
- 実験①： 英語CCGBank (Hockenmaier+, 2007)
  - 100次元GLoVeベクトル、 30次元接辞ベクトル(Lewis+, 2016)
  - 評価： 意味的な依存関係による指標(Clark+, 2007)
- 実験②： 日本語CCGBank (Uematsu+, 2015)
  - 200次元日本語エンティティベクトル、 50次元文字ベクトルから畳み込み→100次元(dos Santos+, 2015)
  - 評価： 終端のカテゴリの正答率、 文節係り受けの係り受け正答率

# 実験①：英語CCGbank test



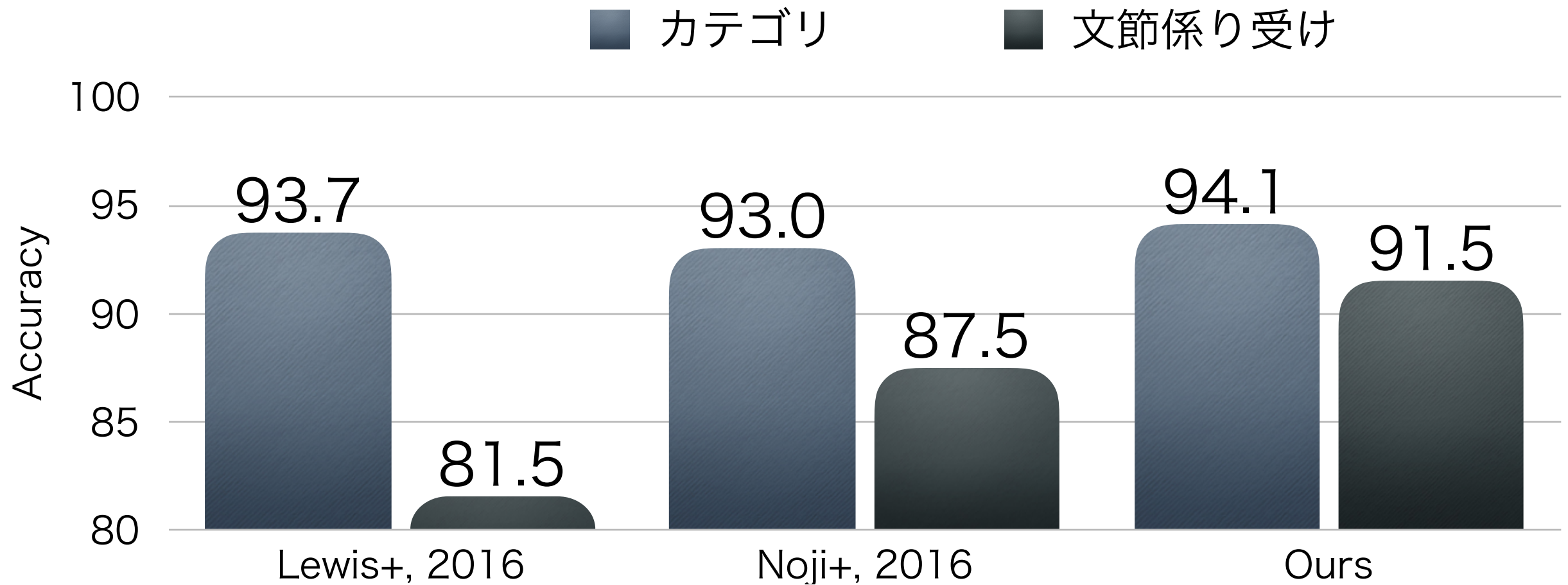
- CCGBankで実験

- Category-factoredモデル(Lewis+, 2016)より向上

- Full derivation + 深層学習モデル(Vaswani+, 2016)に劣る

- Tri-trainingを行うことで、最高精度

# 実験②：日本語CCGbank test



- (Noji+, 2016) : shift-reduce法によるCCG解析器
- LewisらのCategory-factoredモデルは日本語では低精度
- 提案法は日本語でも最高精度



# まとめと今後の予定

- **まとめ：CCG木の確率をカテゴリと係り受け構造でモデル化**
  - カテゴリ列で表現できない構造の違い：特に日本語は重要
  - 局所的な要素の積に分解可：高速なA\*解析を実現
  - 日本語最高精度、半教師あり学習(Tri-training)で英語も
- **今後の予定**
  - 今年度：CCG解析の多分野テキストへの適応
  - 今年度：CCGを用いた並列句の範囲同定
  - 来年度：CCG解析を用いた意味解析(RTE, QAなど)で有効性を検証

おまけ

# 実験③：実行速度の評価

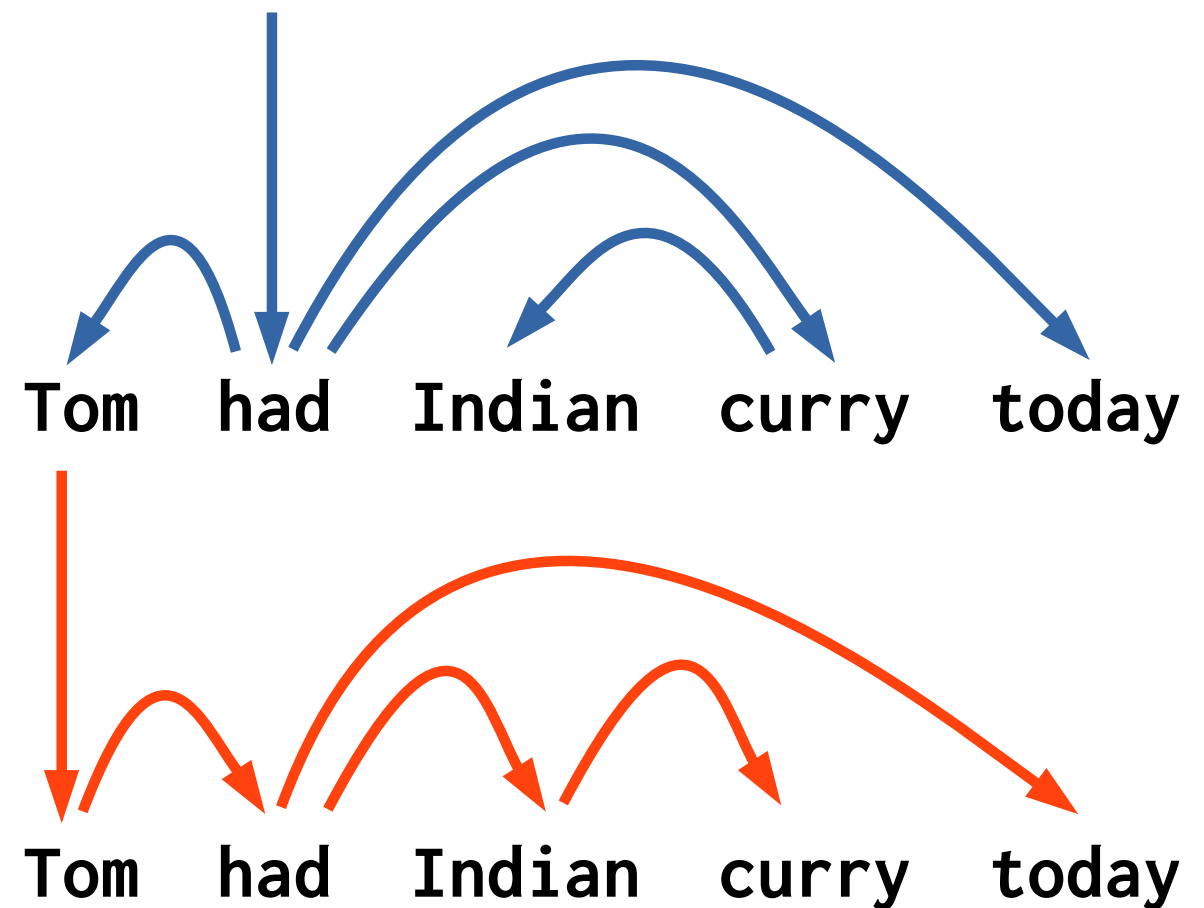
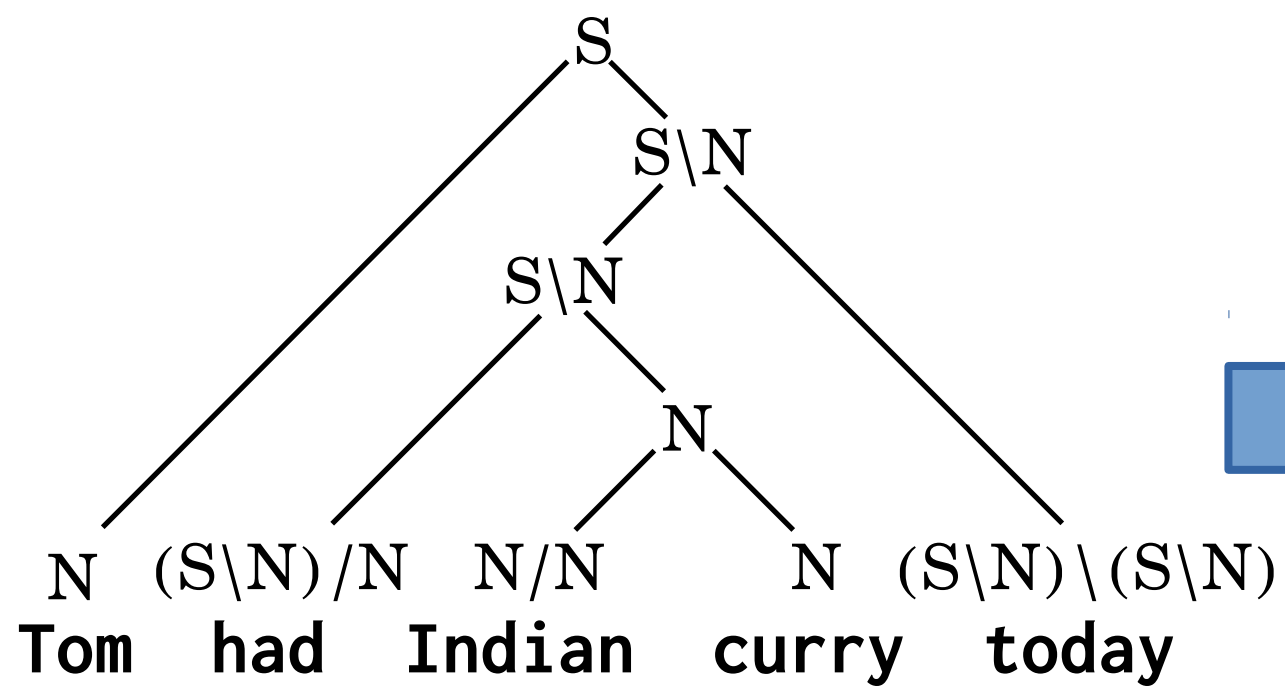
単位は文/秒

	タグ付け	A*解析	全体
Lewis+, 2016 の再実装	24.8	185.2	21.9
Lee+, 2016	21.7	16.7	9.33
提案法	16.6	114.6	14.5

- (Lewis+, 2016)に実行速度は劣る
  - タグ付けは深層学習モデルが少し複雑になったため
  - A\*解析はなぜか少し遅い(提案法のhのほうがtightな推定値のはず?)
    - 行列の前処理など本質的でないところで遅くなっている可能性
- これまでの最高精度の(Lee+, 2016)と比べるとかなり高速

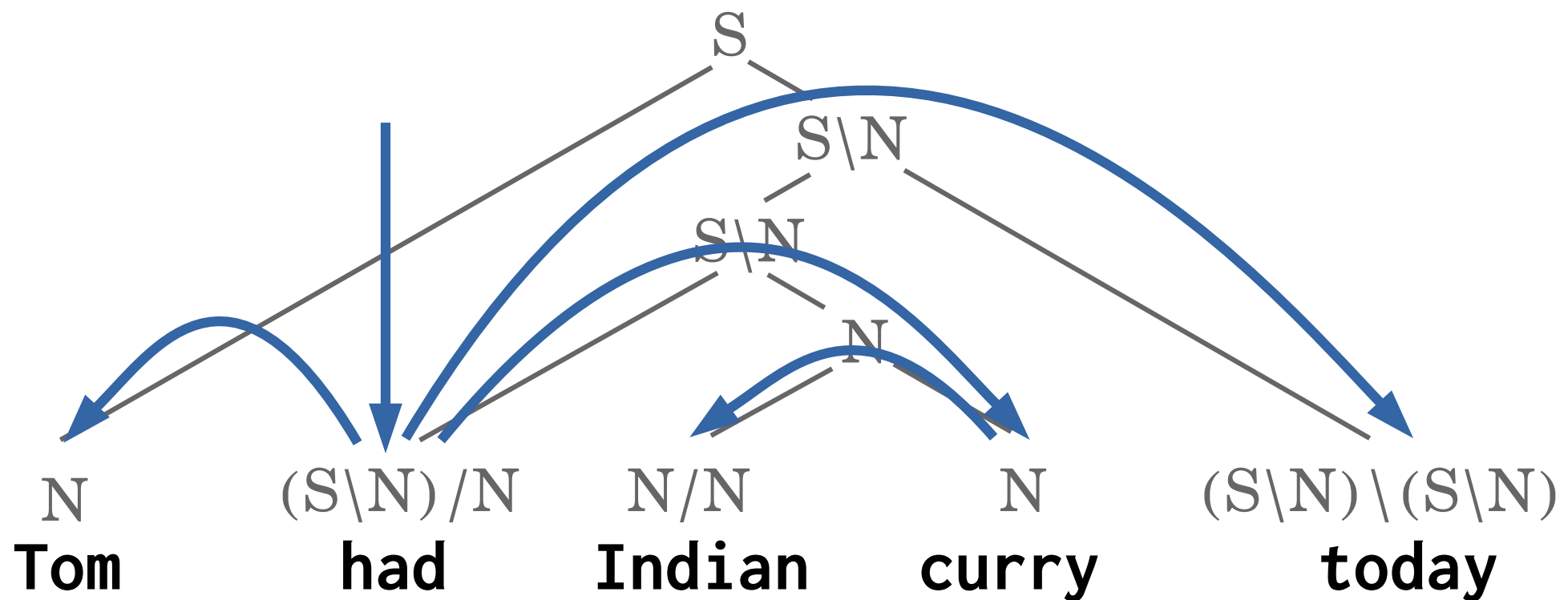
# CCG木から係り受け木への変換

- 変換方法は自明ではなく、自分でデザイン
- 本研究では2種類の変換方法を検討



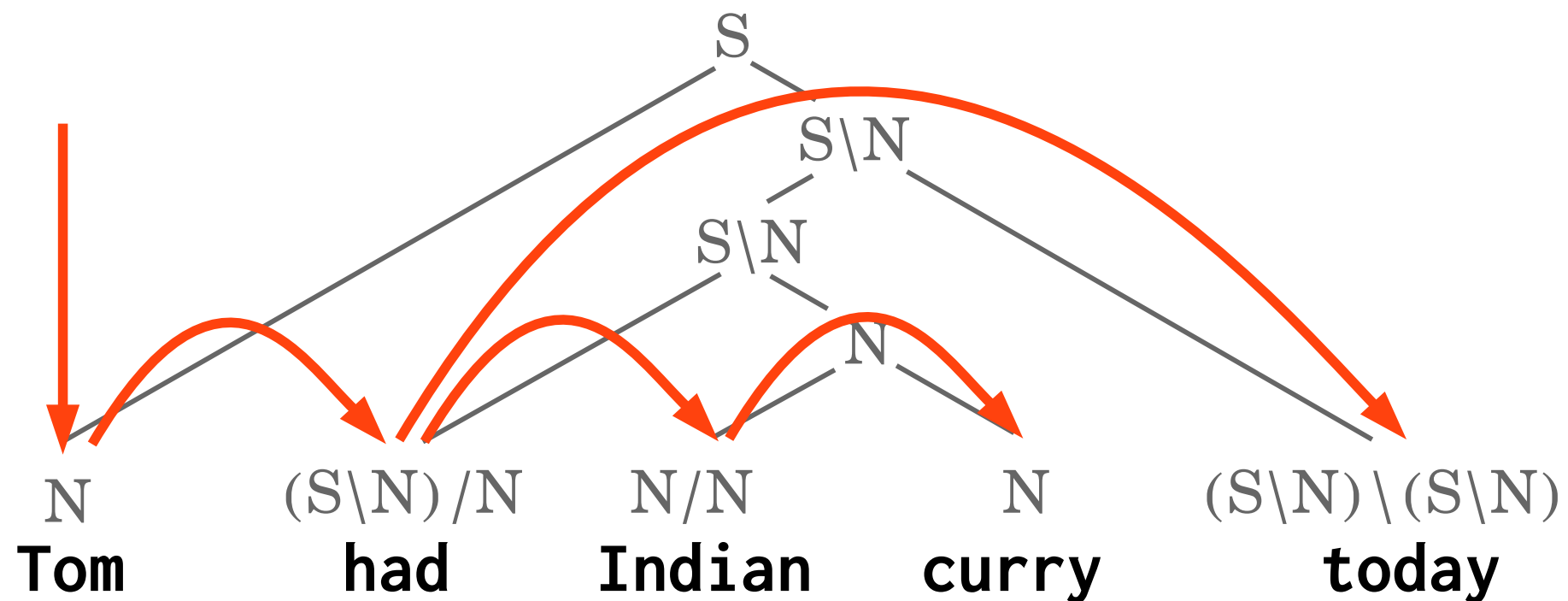
# Lewisらによる変換規則

- それぞれの組み合わせ規則について規則を定義
- 直感的にもっともな係り受け木



# HeadFirst変換規則

- 常に左の子を親として選ぶ
  - 単純だが理論的には変
  - 予測は簡単
- 94.5% vs. 92.5% (UAS, WSJの開発データ, 解析器は(Dyer+, 2015))

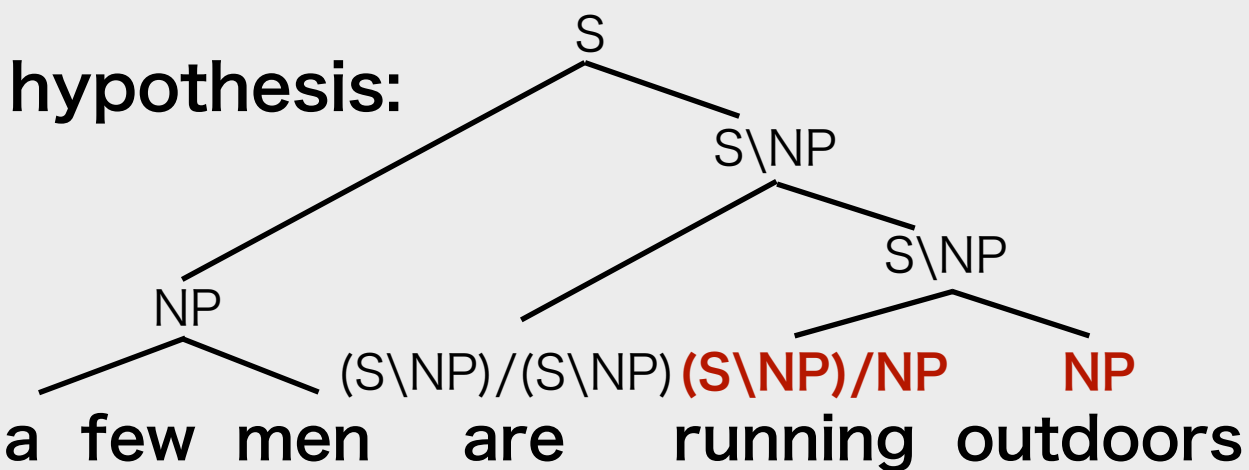
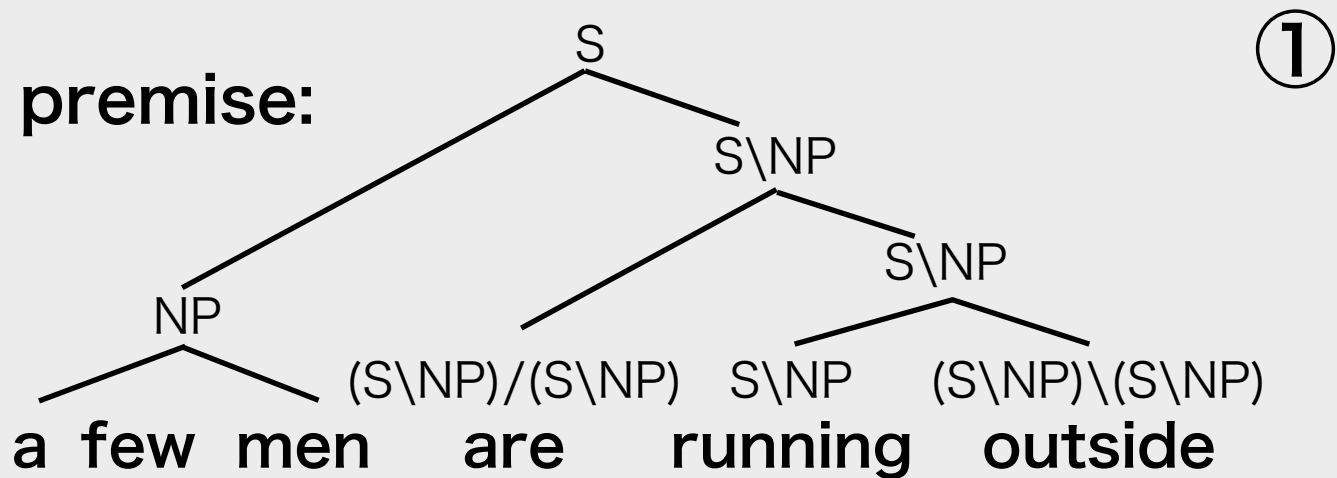


# 実験④：係り受けの種類による精度

Method	Labeled	Unlabeled
<i>CCGbank</i>		
LEWISRULE w/o dep	85.8	91.7
LEWISRULE	86.0	92.5
HEADFIRST w/o dep	85.6	91.6
HEADFIRST	<b>86.6</b>	<b>92.8</b>
<i>Tri-training</i>		
LEWISRULE	86.9	93.0
HEADFIRST	<b>87.6</b>	<b>93.3</b>

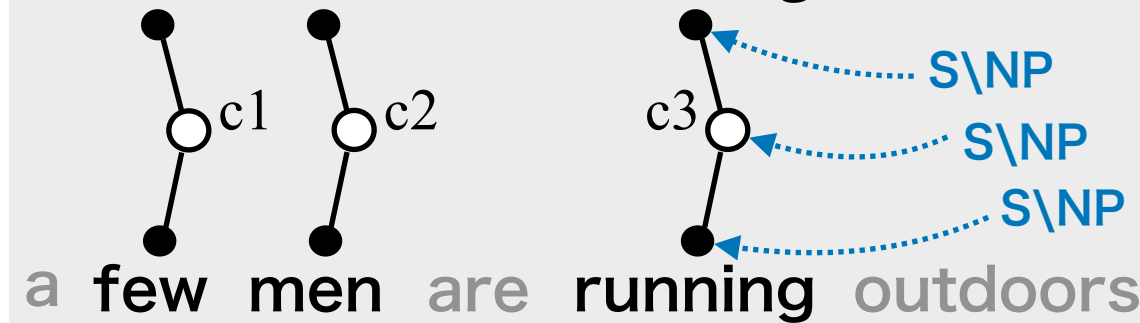
- PTB devセットでの評価
- w/o depは係り受けスコアを捨ててCategory-factoredモデルで解析
  - 係り受けを使うことが有効
- Headfirstのほうが精度が高い

# Consistent CCG Parsing over Multiple Sentences



CCG parsing:  $P(Y|X) = p(\mathbf{y}_p|\mathbf{x}_p)p(\mathbf{y}_h|\mathbf{x}_h)$

a few men are running outside ②

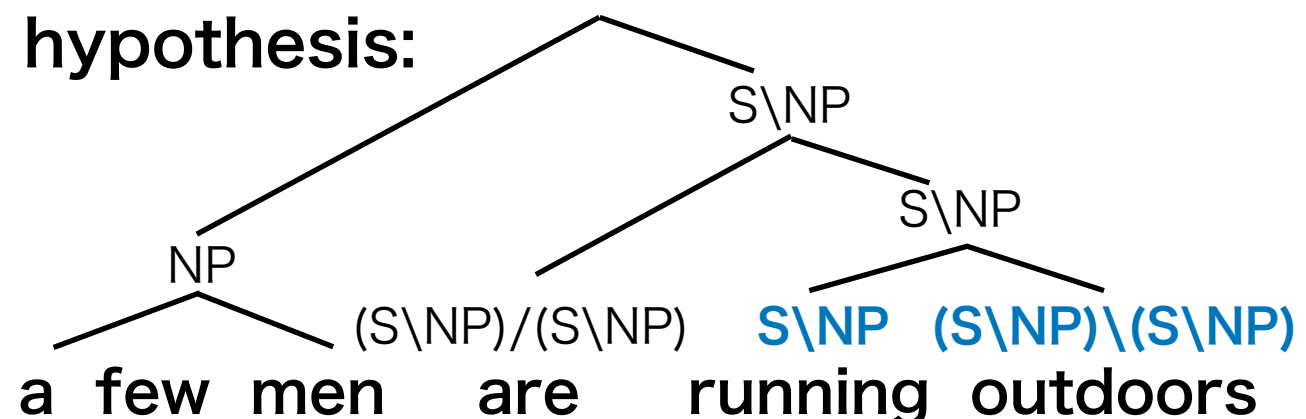
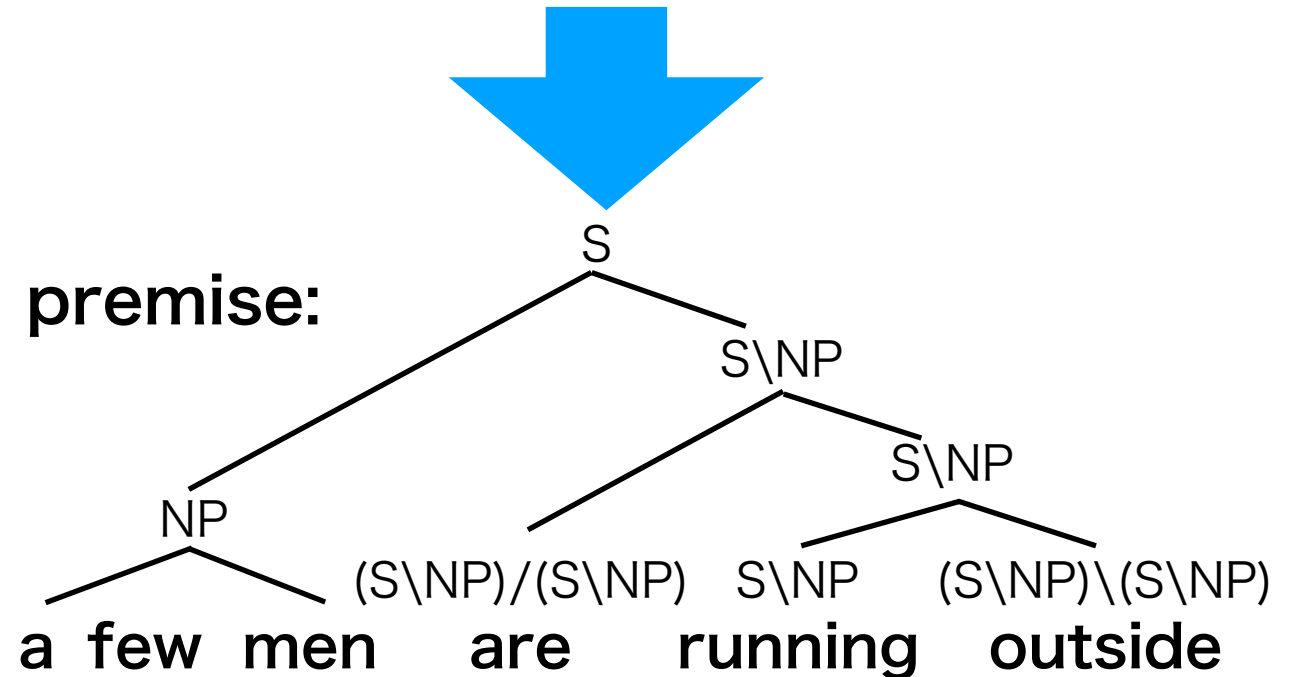


MRF:  $g(\mathbf{z}) = \sum_{v \in V} f_v(z_v) + \sum_{(w,c) \in E} f_{w,c}(z_w, z'_c)$

Joint decoding via Dual Decomposition:

$$(Y^*, \mathbf{z}^*) = \arg \max_{Y \in \mathcal{Y}(X), \mathbf{z} \in \mathcal{Z}(X)} P(Y|X) + g(\mathbf{z})$$

③  $\text{s.t. } \forall \langle s, t \rangle \in W \quad z_{s,t} = c_{s,t}$





# Experiments

- The RTE results of ccg2lambda and LangPro (Abzianidze, 2017)
- Inter-sentence constraints: あり (✓), なし (✗)

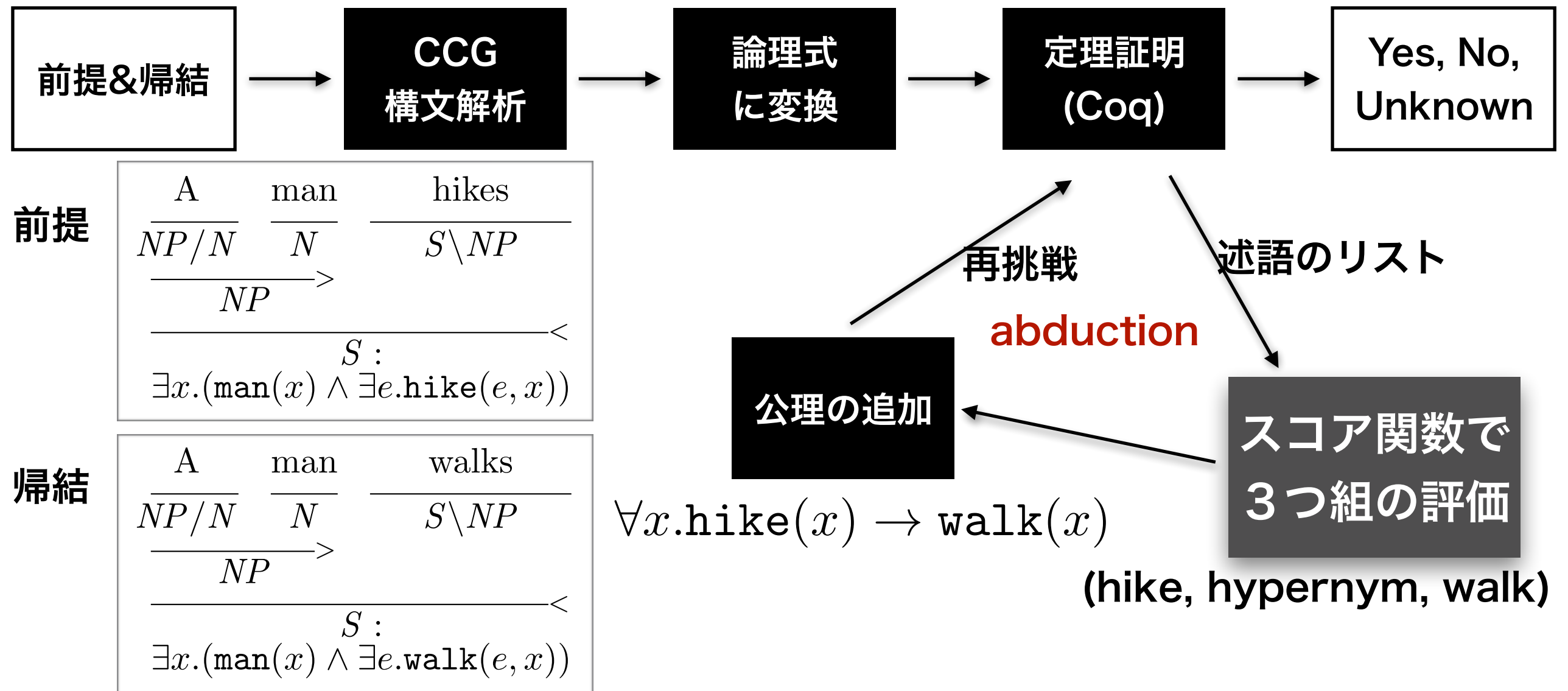
Method	MRF	Accuracy	Precision	Recall
LangPro (EasyCCG)	✗	79.05	<b>98.00</b>	52.67
LangPro	✗	78.85	97.48	52.48
LangPro	✓	<b>79.20</b>	97.60	<b>53.23</b>
ccg2lambda (EasyCCG)	✗	81.59	<b>97.73</b>	58.48
ccg2lambda	✗	81.95	97.19	59.98
ccg2lambda	✓	<b>82.86</b>	97.14	<b>62.18</b>

**Table:** RTE results on the test section of the SICK dataset

Method	MRF	Accuracy	Precision	Recall
ccg2lambda (jigg)	✗	75.0	92.7	65.4
ccg2lambda	✗	67.87	88.34	56.77
ccg2lambda	✓	71.31	88.88	62.24

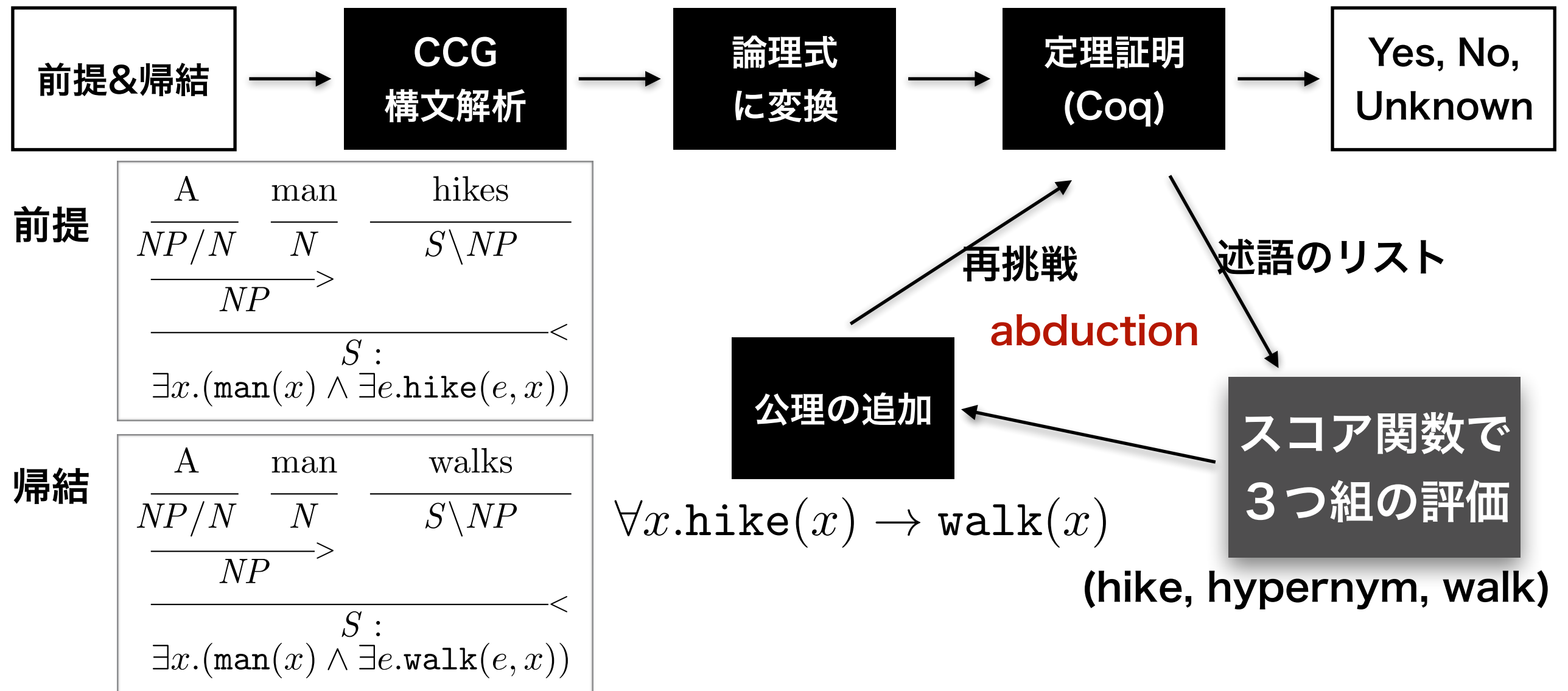
**Table:** RTE results on Japanese Semantics Test Suite (JSeM)

# ccg2lambda (Mineshima+, 2015)



- 問題①: 外部知識(Wordnet等)の拡充と効率性の緊張 ✓
- 問題②: 一度Coqを終了しないといけない

# ccg2lambda (Mineshima+, 2015)



- 問題①: 外部知識(Wordnet等)の拡充と効率性の緊張 ✓
- 問題②: 一度Coqを終了しないといけない ←

# Coqプラグイン(abduction) による更なる高速化

# Coqプラグイン(abduction)

## による更なる高速化

```
1 subgoal
```

```
=====
```

```
(exists x : Entity, man x /\ hike x) ->  
exists x : Entity, man x /\ walk x
```

```
t < intro.
```

```
1 subgoal
```

```
H : exists x : Entity, man x /\ hike x
```

```
=====
```

```
exists x : Entity, man x /\ walk x
```

```
t < abduction.
```

```
1 subgoal
```

```
H : exists x : Entity, man x /\ hike x
```

```
NLax1 : forall x : Entity, hike x -> walk x
```

```
=====
```

```
exists x : Entity, man x /\ walk x
```

# Coqプラグイン(abduction)

## による更なる高速化

```
1 subgoal
```

```
=====
```

```
(exists x : Entity, man x /\ hike x) ->  
exists x : Entity, man x /\ walk x
```

```
t < intro.
```

```
1 subgoal
```

```
H : exists x : Entity, man x /\ hike x
```

```
=====
```

```
exists x : Entity, man x /\ walk x
```

```
t < abduction.
```

```
1 subgoal
```

```
H : exists x : Entity, man x /\ hike x
```

```
NLax1 : forall x : Entity, hike x -> walk x
```

```
=====
```

```
exists x : Entity, man x /\ walk x
```

コンテキストとゴールから  
述語のペアリストを構築

(man, walk)  
(man, hike)  
(hike, walk)

# Coqプラグイン(abduction)

## による更なる高速化

```
1 subgoal
```

```
=====
```

```
(exists x : Entity, man x /\ hike x) ->  
exists x : Entity, man x /\ walk x
```

```
t < intro.
```

```
1 subgoal
```

```
H : exists x : Entity, man x /\ hike x
```

```
=====
```

```
exists x : Entity, man x /\ walk x
```

```
t < abduction.
```

```
1 subgoal
```

```
H : exists x : Entity, man x /\ hike x
```

```
NLax1 : forall x : Entity, hike x -> walk x
```

```
=====
```

```
exists x : Entity, man x /\ walk x
```

コンテキストとゴールから  
述語のペアリストを構築

Pythonサーバに送信

ComplExモデルで述語の  
関係进行评估

(man, walk)  
(man, hike)  
(hike, walk)

# Coqプラグイン(abduction)

## による更なる高速化

```
1 subgoal
```

```
=====
(exists x : Entity, man x /\ hike x) ->
exists x : Entity, man x /\ walk x
```

```
t < intro.
```

```
1 subgoal
```

```
H : exists x : Entity, man x /\ hike x
=====
exists x : Entity, man x /\ walk x
```

```
t < abduction.
```

```
1 subgoal
```

```
H : exists x : Entity, man x /\ hike x
NLax1 : forall x : Entity, hike x -> walk x
=====
exists x : Entity, man x /\ walk x
```

コンテキストとゴールから  
述語のペアリストを構築

Pythonサーバに送信

CompExモデルで述語の  
関係性を評価

スコアでフィルタ

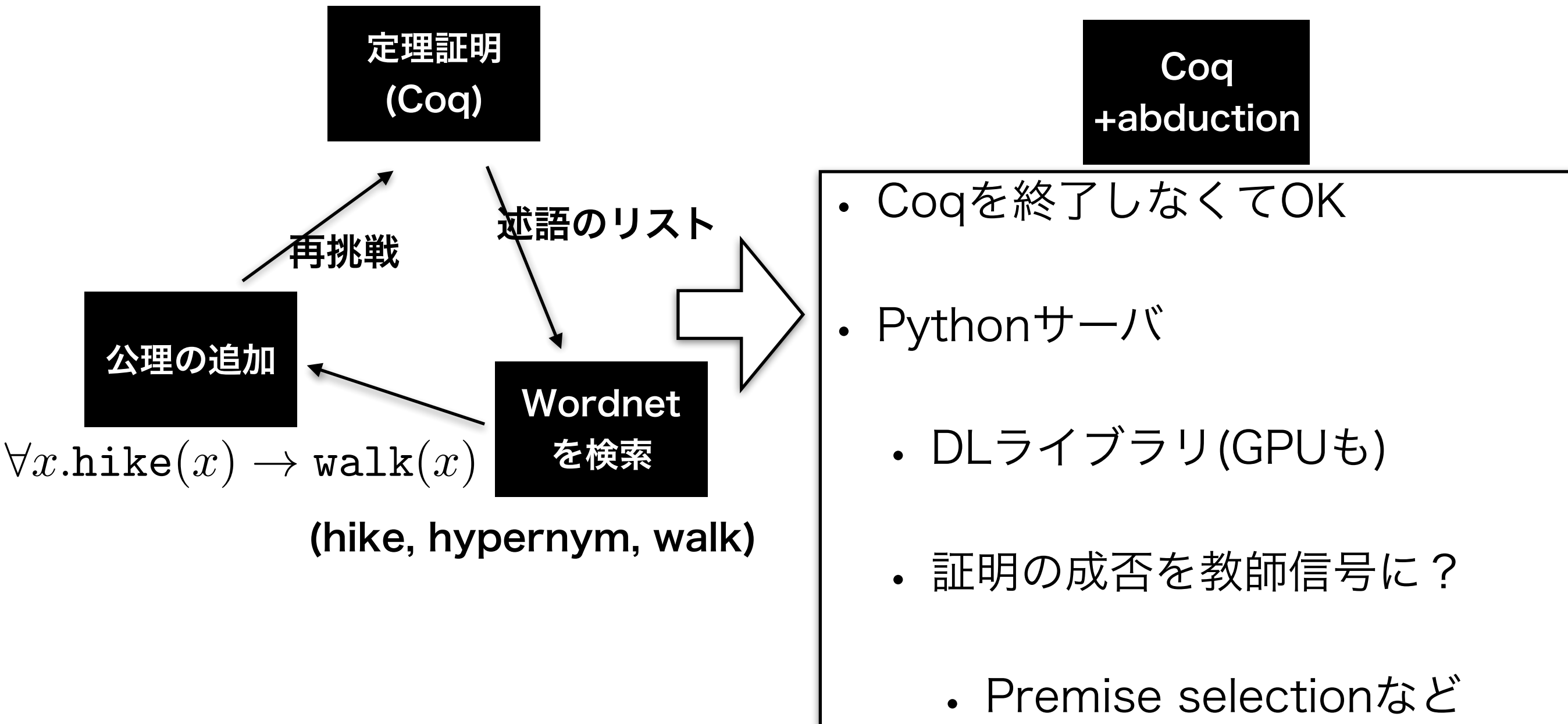
公理として追加

(man, walk)  
(man, hike)  
(hike, walk)

(hike, hyper, walk)



# Coqプラグイン(abduction) による更なる高速化



# 実験

- SICK(Marelli+, 2014)testデータで評価
  - 精度と証明時間
    - (trialデータで5回実験のマクロ平均、タイムアウト100秒)
- 知識ベース補完モデル (ComplEx, Trouillon+ 2016)
  - ロジスティック損失:  $\sum_{((s,r,o),t) \in \mathcal{D}} t \log f(s,r,o) + (1-t) \log(1-f(s,r,o))$
- 学習データ
  - WordNetから3つ組抽出: (同義、反義、上位、下位)
    - 頻度等によりフィルタ(論文詳述)
  - VerbOcean(Chklovski+, 2004): 動詞間の関係

# 実験結果

	正答率	処理速度(秒/問題)
公理生成なし	77.38	4.21
ベースライン(WordNet)	83.55	10.92
+VerbOcean	83.84	11.06
知識グラフ補完(WordNet)	82.82	4.63
+VerbOcean	82.95	4.63

- ベースライン:検索による公理生成(Martínez-Gómez+, 2017)

# 実験結果


	正答率	処理速度(秒/問題)
公理生成なし	77.38	4.21
ベースライン(WordNet)	83.55	10.92
+VerbOcean	83.84	11.06
知識グラフ補完(WordNet)	82.82	4.63
+VerbOcean	82.95	4.63



- ベースライン:検索による公理生成(Martínez-Gómez+, 2017)
- **正答率:** ベースラインにすこし劣る

# 実験結果

	正答率	処理速度(秒/問題)
公理生成なし	77.38	4.21
ベースライン(WordNet)	83.55	10.92
+VerbOcean	83.84	11.06
知識グラフ補完(WordNet)	82.82	4.63
+VerbOcean	82.95	4.63

A red arrow points from the baseline accuracy of 83.55 down to the proposed method's accuracy of 82.82. A green arrow points from the baseline processing speed of 10.92 down to the proposed method's processing speed of 4.63.

- ベースライン:検索による公理生成(Martínez-Gómez+, 2017)
- **正答率:** ベースラインにすこし劣る
- **処理速度:** 公理生成なしの場合と同等に高速化

# 実験結果

	正答率	処理速度(秒/問題)
公理生成なし	77.38	4.21
ベースライン(WordNet)	83.55	10.92
+VerbOcean	83.84	11.06
知識グラフ補完(WordNet)	82.82	4.63
+VerbOcean	82.95	4.63

- ベースライン:検索による公理生成(Martínez-Gómez+, 2017)
- **正答率**: ベースラインにすこし劣る
- **処理速度**: 公理生成なしの場合と同等に高速化
- **VerbOceanでデータ拡張**: 処理速度同じ、正答率は向上