自 然 言 語 処 理 101−10
（1994. 5. 27）

前向き DP 後向き A^* アルゴリズムを用いた確率的日本語形態素解析システム

永田 昌明
NTT 情報通信網研究所

本報告では、入力文を単語に分割し、各単語に品詞を割り当てるための新しい手法について述べる。この手法は、統計的言語モデルと 2 パス N-best 探索アルゴリズムから構成される。この方法は文が分かち書きされていることを前提としないので、日本語の書き言葉の形態素解析に適している。ATR コーパスを用いて学習およびテストを行なった結果、本手法を用いた日本語形態素解析システムは、クローズドテキストに対して、再現率 97.5% および適合率 97.8% を達成した。

A Stochastic Japanese Morphological Analyzer Using a Forward-DP Backward-A^* N-Best Search Algorithm

Masaaki NAGATA
NTT Network Information Systems Laboratories
1-2356 Take, Yokosuka-Shi, Kanagawa, 238-03 Japan
nagata@nttnly.ntt.jp

We present a novel method for segmenting the input sentence into words and assigning parts of speech to the words. It consists of a statistical language model and an efficient two-pass N-best search algorithm. The algorithm does not require delimiters between words. Thus it is suitable for written Japanese. The proposed Japanese morphological analyzer achieved 97.5% recall and 97.8% precision for open text when it was trained and tested on the ATR Corpus.

1 はじめに

近年、統計的な言語モデルを用いて 95% 以上の精度を達成する英語の形態素解析システム (part of speech tagger) が相次いで報告されている [2–4, 10, 11]。これに比べると、日本では、確率的手法を用いる形態素解析システム [9, 12, 14] は少数派であり、依然として、規則に基づく方法 [16] が主流である。

本報告では、統計的言語モデルと 2 パス N-best 探索アルゴリズムを用いて、95% 以上の精度を持つ確率的日本語形態素解析システムを構築できることを示す。

本手法では、日本語の品詞付けモデル (tagging model) として tri-POS モデルを用いる。このモデルのパラメタは ATR 対話データベース (ADD) から学習した。ATR 対話データベースは、報告者が知る限り、人手により形態素解析 (単語分割と品詞付与) が行なわれ、一般に公開されている唯一の日本語コーパスである。

本報告では、入力文に対する形態素解析候補を、最も尤もらしい順に任意の個数 N だけ (N-Best) 求めることができる新しい探索法を提案する。これは、前向きの動的計画法 (DP: Dynamic Programming) と後向きの A^* 探索から構成される。このアルゴリズムは、日本語形態素解析における接続コスト最小法 [7]、文字認識における拡張ビタビアルゴリズム [6]、音声認識における木-トレリス探索 N-Best アルゴリズムという、異なる分野で各々良く知られているアルゴリズムを融合かつ拡張したものである。

さらに、本報告では、形態素解析の性能を評価する新しい手法を提案する。英語と違って、日本語は単語の間に空白を入れる習慣がない (分かち書きしない)。日本語は膠着語なので、単語境界を一貫性を保ちながら決定するのは日本人にとっても難しい。このため、従来、日本語の形態素解析の標準的な評価尺度は存在しなかった。本報告では、英語の構文解析の性能評価に用いられている括弧付け精度 (bracketing accuracy)[1] を日本語の形態素解析に適用し、さらに、これを N-Best 候補の精度を記述できるように拡張する。

以下では、まず本報告で提案する確率的形態素解析アルゴリズムについて述べ、次に解析精度の評価尺度について説明し、最後に実験結果を述べる。

2 品詞付けモデル

2.1 Tri-POS モデルと相対頻度法

我々は、日本語の品詞付けモデルとして tri-POS (tri-class, tri-tag, tri-Ggram, etc.) モデルを用いる。入力文が単語系列 $W = w_1 w_2 \ldots w_n$ に分割され、各単語に品詞系列 $T = t_1 t_2 \ldots t_n$ が付与されるとすると、形態素解析は、単語系列と品詞系列の同時確率 $P(W,T)$ を最大化する単語分割と品詞付与の組を求める問題として定義できる。Tri-POS モデルでは、この同時確率を品詞三つ組確率 $P(t_i|t_{i-2}, t_{i-1})$ と品詞別単語出力確率 $P(w_i|t_i)$ の積で近似する。

$$P(W,T) = \prod_{i=1}^{n} P(t_i|t_{i-2}, t_{i-1}) P(w_i|t_i) \qquad (1)$$

実際には、文境界も特別な記号と考えた方が都合が良いので、次式を用いる。

$$P(W,T) = P(t_1|\#)P(w_1|t_1)P(t_2|\#, t_1)P(w_2|t_2) \prod_{i=3}^{n} P(t_i|t_{i-2}, t_{i-1})P(w_i|t_i)P(\#|t_{n-1}, t_n) \qquad (2)$$

ここで "#" は文境界を表す特別な記号である。もしタグ付きコーパスが利用可能であれば、$P(w_i|t_i)$ と $P(w_i|t_i)$ は、着目する事象の相対頻度を数えることにより推定できる。

$$P(t_i|t_{i-2}, t_{i-1}) = f(t_i|t_{i-2}, t_{i-1}) = \frac{N(t_{i-2}, t_{i-1}, t_i)}{N(t_{i-2}, t_{i-1})} \qquad (3)$$

$$P(w_i|t_i) = f(w_i|t_i) = \frac{N(w,t)}{N(t)} \qquad (4)$$

ここで f は相対頻度を表し、$N(w,t)$ は単語 w がタグ t を持っていた回数、$N(t_{i-2}, t_{i-1}, t_i)$ は品詞列 $t_{i-2}t_{i-1}t_i$ がテキストに現れた回数を表す[1]。

2.2 次元の縮退と再帰的な追跡

次節で述べる探索アルゴリズムを理解するために、ここでは二次隠れマルコフモデル (second order HMM) と拡張ビタビアルゴリズム (extended Viterbi algorithm)

[1] 我々は 120 種類の品詞タグを用いた。ATR コーパスには、26 種類の品詞、13 種類の活用型、7 種類の活用形が定義されている。26 種類の品詞の中で 5 種類が活用する。ここで、品詞・活用型・活用形のリストを品詞タグとして用いると、ATR コーパスには 119 種類のタグがあり、これに文境界マーカを加えて 120 種類となった。

を説明する。$u_1 = t_1$ かつ $u_i = t_{i-1}t_i$ として、複合状態 (combined state) 系列 $U = u_1u_2 \ldots u_n$ を考えると、次の関係が成り立つ。

$$P(u_i|u_{i-1}) = P(t_i|t_{i-2}, t_{i-1}) \quad (5)$$

式 (5) を式 (1) へ代入すると、次式が得られる。

$$P(W,T) = \prod_{i=1}^{n} P(u_i|u_{i-1})P(w_i|t_i) \quad (6)$$

式 (6) は、通常の (一次) HMM と同じ形である。ここで、部分単語系列 $W_i = w_1 \ldots w_i$ と部分品詞系列 $T_i = t_1 \ldots t_i$ を考えると、次式が得られる。

$$P(W_i,T_i) = P(W_{i-1},T_{i-1})P(u_i|u_{i-1})P(w_i|t_i) \quad (7)$$

式 (7) より、各 u_i に関する $P(W_i,T_i)$ の最大値を求めるには、$P(W_{i-1},T_{i-1})$ の最大値を記録し、この値から式 (7) を用いて全ての u_i について $P(W_i,T_i)$ を計算し、各 u_i に関する $P(W_i,T_i)$ の最大値を選べば良いことが分かる。i を 1 から n まで増やし、$P(W_n,T_n)$ を最大にする u_n を選び、この最大確率を導いた状態系列を後向きに辿ることにより、最適な品詞系列を得ることができる。

3 探索戦略

探索アルゴリズムは、前向きの動的計画法と後向きの A^* 探索から構成される。まず、線形時間の動的計画法を用いて、文頭からの全ての部分経路のスコアをテーブルに記録する [2]。次に、A^* アルゴリズムを用いて、文末から部分経路を展開する。後向き探索において、部分経路は完全な経路のスコアを用いて順位付けされる。完全な経路のスコアは、後向き部分経路のスコアと、前向き探索で記録された残りの経路に対する最適なスコアの和として求められる。残りの経路のスコアは既知なので、後向き探索は admissible である。すなわち、N 個の最尤候補が正確に得られる。

3.1 前向き DP 探索

N-Best アルゴリズムで用いる三つのデータ構造を表 1に示す。Parse 構造は、単語およびその単語へ至る最適部分経路の情報を記録する。Parse.start と parse.end は、入力文における単語の開始位置と終了位置を表すインデックスである。Parse.pos は、品詞・活用型・活用形のリストからなる品詞タグである。Parse.nth-order-state は、最後の二つの品詞タグのリストである。このスロットは二次 HMM の複合状態に対応する。Parse.prob-so-far は、文頭から現在の単語へ至るまでの最適部分経路のスコアである。Parse.previous は、従来のビタビアルゴリズムで使われるような、直前の parse 構造へのポインタである。もし後向き N-Best 探索を行なうのならば、このスロットは必要ない。Word 構造は、表記、品詞、品詞別単語出力確率など、辞書中の単語情報を記録する。なお、path 構造については後で説明する。

表 1: N best アルゴリズムで用いるデータ構造

parse 構造	
start	入力文における単語の開始位置
end	入力文における単語の終了位置
pos	単語の品詞
nth-order-state	最後の二つの品詞のリスト
prob-so-far	文頭からの最適部分経路スコア
previous	直前の parse 構造

word 構造	
form	単語の表記
pos	単語の品詞
prob	品詞別の単語出力確率

path 構造	
parse	parse 構造
previous	直前の path 構造
cost-so-far	初期状態からのコスト
total-cost	初期状態から最終状態までのコスト

前向き探索を説明する前に、幾つかの関数とテーブルを定義する。前向き探索では parse-list というテーブルを用いる。このテーブルは parse 構造の終了位置をキーとし、その位置において、各複合状態について最適部分経路スコアを持つ parse 構造のリストを値とする。関数 register-to-paarse-list は parse 構造を parse-list に登録し、最適部分経路スコアを持つものだけをテーブルに残す。関数 get-parse-list は、与えられた位置における parse 構造のリストを返す。関数 leftmost-substrings は、入力文の与えられた位置から始まる部分列と照合する表記を持つ辞書の中の word 構造のリストを返す。

[2] 実際には、後述するように parse-list と path-map という二つのテーブルを用いる。

前向き探索の中心部分を付録 A に示す。前向き探索は、入力文の先頭から始まり、一文字ずつ進む。入力文の各位置において、この位置で終わる最適部分解析候補 (best partial parse) と、この位置から始まる単語仮説 (word hypothesis) の組合せを調べる。もし、品詞付けモデルが部分解析候補と単語仮説の接続を許せば、新しい解析候補が作られ parse-list に登録される。新しい解析候補のスコアは、この位置までの最適部分解析候補のスコア、品詞三つ組確率、および、品詞別単語出力確率の積である。

3.2 後向き A^* 探索

後向き探索では path-map というテーブルを用いる。このテーブルのキーは parse 構造の終了位置であり、その位置において、開始位置と複合状態の異なる組合せについて、最適な部分経路スコアを持つ parse 構造のリストを値とする。Path-map は、複合状態と最後の単語の開始位置の組に関して場合分けされている点が parse-list と異なる。

この区別は本報告で提案する N-Best アルゴリズムにおいて非常に重要である。式 (1) を最大化する候補を求める前向き探索では、品詞系列だけが重要であった。しかし、後向き探索では単語分割と品詞付与の組を最も尤もらしい順に求めたい。Parse-list は、最適候補と同じ品詞系列を持つが最後の単語区切りが異なる (より確率の小さい) 候補を捨ててしまう。

付録 A に示すように, path-map は前向き探索において関数 register-parse-to-path-map を用いて作られる。この関数は parse 構造を path-map に登録し、このテーブルの基準に基づいて最適なものだけを残す。「会議に申し込みたいのですが」という文を前向き探索する際の path-map の内容を付録 B に示す。

また、後向き探索では A^* 探索の状態を表現するために表 1 に示す path 構造を用いる。Path.parse は現在の単語と品詞の組を表す parse 構造である。Path.previous は直前の path 構造へのポインタである。Path.cost-so-far は初期状態から現在の状態までのコストである。Path.total-cost は初期状態から最終状態までのコストの推定値である。

さて、これより、後向き A^* 探索について説明する。ただし、ここでは読者が A^* を既に知っていると仮定し、この形態素解析の問題に対して我々が A^* アルゴリズムをどのように適用したかを述べる。

後向き探索の中心部分を付録 C に示す。後向き探索では、parse 構造を A^* 探索の状態に対応させる。そして確率の対数の絶対値をコストとして用いる。これにより、確率最大の解はコスト最小の解に対応し、確率の積はコストの和に対応する。

後向き探索では、通常の A^* 探索 と同様な意味で open と close という二つのリストを用いる。(関数 find-path, insert-and-sort-path, delete-path などを用いて) リストを操作する際には、二つの状態は parse 構造が同じ開始位置、終了位置、および、複合状態を持つ時に等しいとする。

初期状態は path-map の文末位置に記録された項目を検索することにより得られる (関数 backward-search-initial-paths)。後向き探索は文末に到達したら終了する (関数 is-beginning-of-sentence)。次状態 (successor state) は、現在の parse の開始位置における path-map の項目を検索し、これへの遷移が二次 HMM における複合状態遷移の制約を満たし、かつ、品詞付けモデルで許されているかどうかを検査した結果として得られる (関数 immediate-left-parses)。複合状態遷移の制約とは、現在の parse の parse.nth-order-state に記録された品詞系列から最後の要素を取り除いた系列が、直前の parse の parse.nth-order-state に記録された品詞系列から最初の要素を取り除いた系列と一致することである。

後向き探索の状態遷移コストは、品詞三つ組確率と品詞別単語出力確率の積である (関数 transition-and-word-cost)。残りの経路のスコアの推定値は parse 構造の parse.prob-so-far スロットから得られる (関数 cost-from-beggining-of-sentence)。

「会議に申し込みたいのですが」という文を後向き探索する際の open の内容を付録 D に示す。後向き探索は N-Best 候補を一つずつ順番に生成するので、予め候補数 N を決める必要はない。また、後向き探索の計算量は、前向き探索に比べて非常に小さい。

4 評価尺度

我々は、英語の構文解析システムの性能評価尺度 [1] を日本語の形態素解析システムへ適用する。ここでは、文の形態素解析をラベル付き括弧付け (labeled bracketing) とみなす。括弧は単語境界に対応し、ラベルは品詞に対応する。そして、システム出力に含まれた括弧の集合と、正解に含まれた括弧の集合を比較する。N-best 候

補に対しては、各候補に含まれた括弧の集合和を求め、これを正解と比較する。

比較の手順は次の通りである。まず、正解データの括弧数 (Std)、システム出力の括弧数 (Sys)、照合した括弧数 (M) を求める。次に、再現率 (recall = M/Std) と適合率 (precision = M/Sys)、および、交差数 (crossing) を求める。

次に、括弧の照合に際して、2 種類の等値性を定義する。もし二つの括弧の境界が等しいならば、二つの括弧はラベルなし括弧として等しい (unlabeled bracket equal) とする。また、もし二つの括弧の境界が等しく、さらにラベルも等しければ、二つの括弧はラベル付き括弧として等しい (labeled bracket equal) とする。

単語分割の精度、すなわち、二つの括弧付けの構造無矛盾性 (structure consistency) を比較する場合には、ラベルなし括弧の等値性に基づいて再現率・適合率・交差数を計算する。また、単語分割と品詞付与の精度、すなわち、二つの括弧付けのラベル無矛盾性 (label consistency) を比較する場合には、ラベル付き括弧の等値性に基づいて再現率・適合率・交差数を計算する。

```
> (morph-n-best "それでは登録用紙をお送り致します。")
-31.90894138309038
それでは / 接続詞 登録 / 普通名詞 用紙 / 普通名詞
 を / 格助詞 お / 接頭語 送り / 本動詞・連用・五段
 致し / 補助動詞・連用・五段 ます / 助動詞・終止 。 / 記号
-38.594338366582356
それ / 代名詞 で / 格助詞 は / 係助詞 登録 / 普通名詞
 用紙 / 普通名詞 を / 格助詞 お / 接頭語 送り / 本動詞・連用・五段
 致し / 補助動詞・連用・五段 ます / 助動詞・終止 。 / 記号
-43.10567483646801
それでは / 接続詞 登録 / 普通名詞 用紙 / 普通名詞
 を / 格助詞 お / 接頭語 送り / 本動詞・連用・五段
 致し / 本動詞・連用・五段 ます / 助動詞・終止 。 / 記号
```

図 1: N-Best 形態素解析候補

図 1に、N-best 形態素候補の例を示す。この例では第 1 候補が正解である。第 2 候補について考えると、正解 (第 2 候補) には 9 個の括弧があり (Std=9)、第 2 候補には 11 個の括弧があり (Sys=11)、照合した括弧は 8 個である (M=8)。従って、ラベル無矛盾性に関する再現率と適合率は 8/9 と 8/11 である。上位 2 候補について考えると、システム出力には 12 個の括弧があり、その中で 9 個が照合しているので、ラベル無矛盾性に関する再現率と適合率は 9/9 と 9/12 である。第 3 候補について考えると、正解と第 3 候補は品詞タグが一つ違うだけなので、構造無矛盾性に関する再現率と適合率は 9/9 と 9/9 である。

5 実験

表 2: 訓練データと試験データの量

	訓練テキスト	クローズドテスト	オープンテスト
文	10945	1000	1000
単語	149059	13176	13899
文字	267422	94221	98997

本報告で提案した形態素解析法の訓練とテストは ATR 対話データベース [5] を用いて行なった。このコーパス (約 800,000 語) は人手により単語分割と品詞付与が行なわれている。実験では、ATR コーパスの中の「国際会議予約に関するキーボード会話」(全体の約 1/4) を用いた。まず、オープンテストのためにランダムに 1,000 文を選び、残りの文を訓練に用いた。次に、クローズドテストのために訓練セットの中から 1,000 文をランダムに選んだ。訓練セットと二つのテストセットの文・単語・文字の総数を表 2に示す。

訓練テキストは 6,580 種類の単語と 6,945 種類の三つ組を含んでいた。オープンテストの中には 247 種類の未知語と 213 種類の未知品詞三つ組があった。従って、オープンテストには品詞三つ組確率と単語出力確率のスムージングが必須である。

表 3: 単語分割と品詞付与が正しく行なわれた単語の割合 (二つ組と三つ組の比較)

	二つ組 (クローズドテスト)			三つ組 (クローズドテスト)		
	再現率	適合率	交差数	再現率	適合率	交差数
1	96.2%	96.6%	0.001	97.5%	97.8%	0.001
2	98.0%	89.7%	0.004	99.0%	90.7%	0.007
3	98.9%	83.5%	0.010	99.5%	84.3%	0.012
4	99.2%	78.5%	0.013	99.7%	79.6%	0.015
5	99.4%	74.2%	0.017	99.8%	76.0%	0.015

我々はクローズドテストセットを用いて品詞二つ組と品詞三つ組の性能を比較した。正しい単語分割と品詞付与が行なわれた単語の割合を表 3に示す。三つ組モデルが第 1 候補で再現率 97.5% および適合率 97.8% を達成したのに対して、二つ組モデルは再現率 96.2% および適合率 96.6% であった。どちらの品詞付けモデルも非

常に高い精度を持っているが、全ての評価尺度に関して三つ組モデルの方が二つ組モデルより優れている。

6 考察

形態素解析は、かな漢字変換・音声認識・文字認識・音声合成・推敲支援・情報検索・機械翻訳など多くのアプリケーションの基盤技術である。

従来の日本語形態素解析システムは、多くの場合、品詞接続表 (品詞対文法) を言語モデルとし、最長一致や文節最小などの発見的規則を用いて候補に順位を与える [16]。また、従来の確率的な日本語形態素解析システムは、言語モデルとして品詞二つ組 [9, 14] または文字 HMM [12] を用いていた。発見的規則に基づくアプローチも統計に基づくアプローチも、線形時間の動的計画法である接続コスト最小法 [7] を探索に用いることが多い。

本報告で提案した日本語形態素解析アルゴリズムの特徴は、タグ付きコーパスから品詞三つ組に基づいた統計的言語モデルを学習し、これに基づいて入力文に対する最も尤もらしい N 個の形態素解析候補を求めることである。このアルゴリズムは、高次のマルコフモデルを扱うように自然に拡張できる。また、音声認識や文字認識の出力となるラティス形式の入力を扱うように拡張することも容易である。

本報告では、言語モデルのパラメタを相対頻度法で求めているが、現在の学習データの量では品詞三つ組がスパース (sparse) になるのは避けられない。また本報告では、全ての単語が辞書に登録されていることを仮定しているが、これもあまり現実的ではない。従って、未知構文 (未知品詞列) および未知語が扱えるように、品詞三つ組確率および単語出力確率をスムージングすることが必要である。

なお、品詞三つ組確率と単語出力確率のスムージング方法、および、オープンテキストでの解析精度については、また改めて報告する予定である [13]。

7 おわりに

我々は確率的日本語解析システムを開発した。このシステムは、統計的な品詞付けモデルと入力文の最尤な N 個の形態素解析候補を求めるための効率的な 2 パスアルゴリズムを用いる。このシステムの単語分割および品詞付与の精度は 95% 以上であり、これは現在の最先端の確率的英語形態素解析システムと同程度の性能である。

参考文献

[1] Black, E. et al.: "A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars", DARPA Speech and Natural Language Workshop, pp.306-311, Morgan Kaufmann, 1991.

[2] Charniak, E., Hendrickson, C., Jacobson, N., and Perkowitz, M.: "Equations for Part-of-Speech Tagging", AAAI-93, pp.784-789, 1993.

[3] Church, K.: "A Stochastic Part of Speech Tagger and Noun Phrase Parser for English", ANLP-88, pp.136-143, 1988.

[4] Cutting, D., Kupiec, J., Pedersen, J., and Sibun, P.: "A Practical Part-of-Speech Tagger", ANLP-92, pp.133-140, 1992.

[5] Ehara, T., Ogura, K. and Morimoto, T.: "ATR Dialogue Database," ICSLP-90, pp.1093-1096, 1990.

[6] He, Y.: "Extended Viterbi Algorithm for Second Order Hidden Markov Process", ICPR-88, pp.718-720, 1988.

[7] 久光徹, 新田義彦: "接続コスト最小法による形態素解析の提案と計算量の評価について", 言語理解とコミュニケーション研究会 NLC90-8, pp.17-24, 電子情報通信学会, 1990.

[8] Jelinek, F.: "Self-organized language modeling for speech recognition", IBM Report, 1985 (Reprinted in *Readings in Speech Recognition*, pp.450-506).

[9] 松延栄治, 日高達, 吉田将: "確率文節文法による構文解析", 自然言語処理研究会 NL 56-3, 情報処理学会, 1986.

[10] Merialdo, B.: "Tagging Text with a Probabilistic Model", ICASSP-91, pp.809-812, 1991.

[11] Meteer, M. W., Schwartz, R. and Weischedel, R.: "POST: Using Probabilities in Language Processing", IJCAI-91, pp.960-965, 1991.

[12] 村上仁一, 嵯峨山茂樹: "HMM を用いた形態素解析", 情報処理学会第 45 回全国大会, Vol.3, pp.161-162, 1992.

[13] Nagata, M.: "A Stochastic Japanese Morphological Analyzer Using a Forward-DP Backward-A^* N-Best Search Algorithm", to appear in COLING-94, 1994.

[14] Sakai, S.: "Morphological Category Bigram: A Single Language Model for both Spoken Language and Text", ISSD-93, pp.87-90, 1993.

[15] Soong, F. K. and Huang E.: "A Tree-Trellis Based Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition", ICASSP-91, pp.705-708, 1991.

[16] 吉村賢治, 日高達, 吉田将: "文節数最小法を用いたべた書き日本語文の形態素解析", 情報処理学会論文誌, Vol.24, No.1, pp.40-46, 1983.

A 前向き DP 探索アルゴリズム

```
function forward-pass(string)
begin
initial-step(); # Pads special symbols at both ends.
for i=1 to length(string) do
  foreach parse in get-parse-list(i) do
    foreach word in leftmost-substrings(string,i) do
      pos-ngram := append(parse.nth-order-state, list(word.pos))
      if (transprob(pos-ngram) > 0) then
        new-parse := make-parse();
        new-parse.start := i;
        new-parse.end := i + length(word.form);
        new-parse.pos := word.pos;
        new-parse.nth-order-state := rest(pos-ngram);
        new-parse.prob-so-far := parse.prob-so-far * transprob(pos-ngram) * word.prob;
        new-parse.previous := parse;
        register-parse-to-parse-list(new-parse);
        register-parse-to-path-map(new-parse);
      endif
    end
  end
end
final-step(); # Handles transition to the end symbol.
end
```

B 前向き DP 探索における path-map の内容

「会議に申し込みたいのですが。」という文の前向き探索がが、部分文字列「会議に」(文字位置 3) までが進んだ段階における最適経路スコアテーブル (path-map) の内容を以下に示す。ここで SBM は文の境界を表す特別なマーカ (sentence boundary marker) である。

```
((-1 0 (SBM SBM))
 (#<Parse NIL -1 0 SBM (SBM SBM) 0.0>))
((0 1 (SBM 接尾語))
 (#<Parse 会 0 1 接尾語 (SBM 接尾語) -16.8>))
((0 1 (SBM 普通名詞))
 (#<Parse 会 0 1 普通名詞 (SBM 普通名詞) -7.0>))
((0 2 (SBM 固有名詞))
 (#<Parse 会議 0 2 固有名詞 (SBM 固有名詞) -11.0>))
((0 2 (SBM 普通名詞))
 (#<Parse 会議 0 2 普通名詞 (SBM 普通名詞) -4.9>))
((2 3 (普通名詞 格助詞))
 (#<Parse に 2 3 格助詞 (普通名詞 格助詞) -7.3>))
((2 3 (固有名詞 格助詞))
 (#<Parse に 2 3 格助詞 (固有名詞 格助詞) -13.3>))
((2 3 (普通名詞 助動詞・連用))
 (#<Parse に 2 3 助動詞・連用 (普通名詞 助動詞・連用) -13.2>))
((2 3 (固有名詞 助動詞・連用))
 (#<Parse に 2 3 助動詞・連用 (固有名詞 助動詞・連用) -17.2>))
```

この例では、一つのリストが path-map の一つの項目を表す。各リストは、第一要素がキーの値であり、第二要素がそのキーの値を持つ parse 構造のリストである。例えば、最後のリストは、単語の開始位置が 2 で、終了位置が 3 で、最後の二つの品詞が 固有名詞と助動詞・連用であるような parse 構造を格納している。

ここで、一つの parse 構造は #<Parse 表記 開始位置 (parse.start) 終了位置 (parse.end) 品詞 (parse.pos) 複合状態 (parse.nth-order-state) 最適部分経路スコア (parse.prob-so-far)> という形式で表記している。単語表記は開始位置と終了位置から求められるので、本来、parse 構造には格納されていないが、分かり易さを考慮してここに加えた。また、最適部分経路スコアは確率の対数を用いている。

C 後向き A^* 探索アルゴリズム

```
function backward-pass()
begin
open := backward-search-initial-paths();
closed := nil;
LOOP: if open == nil then exit(fail);
  bestpath := first(open);
  if is-beginning-of-sentence(bestpath.parse) then exit(success);
  open := rest(open);
  insert-and-sort-path(bestpath,closed);
  foreach successorparse in immediate-left-parses(bestpath.parse) do
    newpath := make-path();
    newpath.parse := successorparse;
    newpath.previous := bestpath;
    newpath.cost-so-far := bestpath.cost-so-far + transition-and-word-cost(bestpath.parse,newpath.parse);
    newpath.total-cost := newpath.cost-so-far + cost-from-beginning-of-sentence(newpath.parse);
    if oldpath := find-path(successorparse,open) then
      if newpath.total-cost < oldpath.total-cost then
        delete-path(oldpath,open);
        insert-and-sort-path(newpath,open);
      endif
    elseif oldpath := find-path(successorparse,closed) then
      if newpath.total-cost < oldpath.total-cost then
        delete-path(oldpath,closed);
        insert-and-sort-path(newpath,open);
      endif
    else
      insert-and-sort-path(newpath,open);
    endif
  end
goto LOOP;
```

D 後向き A^* 探索における open の内容

「会議に申し込みたいのですが。」という文の後向き探索において、一番外側のループに入った直後の open の内容の変化を以下に示す。Parse 構造と path 構造の previous スロットは表示していない。

```
初期状態
(#<Path #<Parse NIL 14 15 SBM (記号 SBM) -25.8> 0.0 25.8>)
2周目
(#<Path #<Parse 。 13 14 記号 (接続助詞 記号) -24.5> 1.4 25.8>
 #<Path #<Parse 。 13 14 記号 (格助詞 記号) -34.6> 1.6 36.2>)
3周目
(#<Path #<Parse が 12 13 接続助詞 (助動詞・終止 接続助詞) -23.7> 2.2 25.8>
 #<Path #<Parse が 12 13 接続助詞 (助動詞・連体 接続助詞) -33.2> 2.2 35.4>
 #<Path #<Parse 。 13 14 記号 (格助詞 記号) -34.6> 1.6 36.2>)
4周目
(#<Path #<Parse です 10 12 助動詞・終止 (準体助詞 助動詞・終止) -21.6> 4.3 25.8>
 #<Path #<Parse が 12 13 接続助詞 (助動詞・連体 接続助詞) -33.2> 2.2 35.4>
 #<Path #<Parse 。 13 14 記号 (格助詞 記号) -34.6> 1.6 36.2>
 #<Path #<Parse す 11 12 助動詞・終止 (接続助詞 助動詞・終止) -39.1> 5.4 44.5>
 #<Path #<Parse す 11 12 助動詞・終止 (助動詞・連用 助動詞・終止) -38.5> 6.3 44.8>)
5周目
(#<Path #<Parse の 9 10 準体助詞 (助動詞・連体 準体助詞) -20.3> 5.5 25.8>
 #<Path #<Parse が 12 13 接続助詞 (助動詞・連体 接続助詞) -33.2> 2.2 35.4>
 #<Path #<Parse 。 13 14 記号 (格助詞 記号) -34.6> 1.6 36.2>
 #<Path #<Parse す 11 12 助動詞・終止 (接続助詞 助動詞・終止) -39.1> 5.4 44.5>
 #<Path #<Parse す 11 12 助動詞・終止 (助動詞・連用 助動詞・終止) -38.5> 6.3 44.8>)
```