統計的かな漢字変換システム Mozc

工藤拓, 小松弘幸, 花岡俊行, 向井淳, 田畑悠介
Google 株式会社
{taku,komatsu,toshiyuki,mukai,tabata}@google.com

1　はじめに

かな漢字変換は自然言語処理の代表的な応用として知られているにもかかわらず, それを題材とした研究は構文解析といった他の分野と比べると少ない. また, これまでのかな漢字変換の研究はひらがなをかな漢字混じり文に変換するタスクのみに着目するものが多く, かな漢字変換に必要な機能を総合的に議論した研究は我々の知る限り無い. かな漢字変換は日本語固有の処理であり, 国際的に認知されにくいというのがその要因のひとつとして考察できるが, 同タスクそのものが自然言語処技術がサポートできる余地のないほど十分に完成された応用であるという認識が広まっているのも否定できない. 我々もそのような甘い見積もりのまま Mozc[1] (製品名 Google 日本語入力) プロジェクトを開始したが, いざプロジェクトを始めると, ひらがなをかな漢字混じり文に変換するという単純なタスクだけでは割り切れない様々な処理が実環境のかな漢字変換システムに要求されることを認識した. 例えば, 文節分割, 文節切りの変更, 候補の N-best 表示, 副作用の少ない学習機能, ユーザの利用シーンに即した評価方法といった技術的な課題が挙げられる.

本論文では, Web コーパスを用いた統計的かな漢字変換システム Mozc の設計と実装を紹介するとともに, 実環境で必要な個々の技術的課題を我々がどのように解決し, Mozc の中でどのように実装されているかを紹介する.

2　Mozc の変換エンジン

2.1　統計的かな漢字変換

統計的かな漢字変換では, 入力ひらがな列 $\mathbf{x}$ に対し, 出力文 $\mathbf{y}$ を出力する条件付き確率 $P(\mathbf{y}|\mathbf{x})$ を考える. $\mathbf{x}$ に対する変換結果は, $P(\mathbf{y}|\mathbf{x})$ を最大化する $\hat{\mathbf{y}}$ を導出する問題として定式化できる. この最大化問題はベイスの定理により, 言語モデル $P(\mathbf{x})$ とかな漢字モデル $P(\mathbf{x}|\mathbf{y})$ の積の最大化として表現できる. この定式化は文献 [6, 5] のそれと同一である.

$$\begin{aligned}\hat{\mathbf{y}} &= \operatorname{argmax} P(\mathbf{y}|\mathbf{x}) \\ &= \operatorname{argmax} P(\mathbf{y})\, P(\mathbf{x}|\mathbf{y})\end{aligned}$$

言語モデルとかな漢字モデルの実装には自由度がある. Mozc では出力候補 $\mathbf{y}$ を単語列 $\mathbf{y} = w_1 \dots w_n$ とみなし, 言語モデルにクラスバイグラムモデル, かな漢字モデルは単語-読みユニグラムモデルを用いている.

$$\begin{aligned}P(\mathbf{y}) &= \prod_{i=1}^{n} P(w_i|c_i)P(c_i|c_{i-1}) \\ P(\mathbf{x}|\mathbf{y}) &= \prod_{i=1}^{n} P(r_i|w_i)\end{aligned}$$

ただし, c_i は単語 w_i の言語クラス (一般的には品詞や活用形など), $P(w_i|c_i)$ は単語生起確率, $P(c_i|c_{i-1})$ はクラス遷移確率, r_i は単語 w_i の読み候補, $P(r_i|w_i)$ は単語 w_i を r_i と読む確率となる. 最尤出力 $\hat{\mathbf{y}}$ は, 動的計画法の一種である Viterbi アルゴリズムを用いて効率よく計算することが可能である.

言語クラスには様々な設計方法が考えられるが, Mozc では, 形態素解析器によって得られた形態素情報をベースに設計している. 具体的には, 以下のルールによりクラスを決定している.

- IPA 品詞体系[2]の最も深い品詞階層を使用
- 活用する単語は, 活用形, 活用型も全部展開
- 助詞, 助動詞, 非自立の内容語はすべて語彙化
- 頻出する動詞, 固有名詞以外の名詞, 形容詞, 接尾辞, 接頭辞は語彙化

語彙化単語が多いため, クラス数は 3000 程度になる[3].

2.2　モデルの推定方法

確率 $P(w_i|c_i)$, および $P(c_i|c_{i-1})$ は, 形態素解析済みのコーパスから最尤推定によって計算する. かな漢字変換モデルは, 読みと単語のペアと推測されるデータを Web 等からマイニングし, そのデータから最尤推定法により計算する[4].

言語モデルの計算には, 約 130 億文から構成される大規模 Web コーパス, 形態素解析器 MeCab[5], 並列分散処理システム MapReduce[1] を用いている. コーパスの最終的な内容は異なるが大規模 Web コーパスの構築方法は基本的に文献 [4] と同一である.

2.3　複合語化

クラスバイグラムモデルには, 長い文脈を考慮できない問題点がある. この解決法として, トライグラム以上の長い文脈に拡張することが考えられるが, デコードの時間, デコーダの複雑化, 計算機リソースの制限など実環境においては考慮すべき課題が多い.

Mozc では, 連続する k 個の単語を改めて一つの単語 (複合語) とすることで長距離文脈を擬似的に考慮している. この方法の利点は, 既存のクラスバイグラムデコーダを変更する必要がないことと, サジェスト機能 (ユーザの入力から候補を補完する機能) にそのまま複合語辞書が適用できる点にある.

具体的には, 複合語化したい単語列を, 形態素列の正規表現パターン (複合語化ルール) を使って Web コーパスから抽出し, 頻度が一定のしきい値以上のものを複合語として辞書に登録する. 複合語化ルールとして姓と名の連続, 名詞と接尾辞の連続などがある. 複合語化された単語は, あたかも形態素解析レベルで 1 つの単語となったとみなして言語モデルの確率値を再計

[1] http://code.google.com/p/mozc/

[2] http://sourceforge.jp/projects/ipadic/

[3] 初期の実装では, 連接表の実装の制約からクラス数が 1000 程度になるように, 複雑なルールでクラスを設計していたが, 現在はメンテナンス性を向上させるためにルールを単純化している.

[4] たとえばアンカーテキストなど. 単語に対する読みは既存の辞書に掲載されているものを用い, 読みと単語のデータはそのペアの頻度を算出するためだけに用いる. そのため, 十分な量のペアがマイニングできれば, 少々のノイズがあっても実用に耐えうる.

[5] http://mecab.sourceforge.net/

算する．ただし，複合語化された単語は，一般に複合語中の最左の単語と最右の単語が異なるクラスを持つことがあるため，単語を左から見た場合と右から見た場合とで異なる言語クラスを用いるように拡張している[6]．また複合語 w の単語生起確率は，構成語の言語クラス $c_1, \ldots, c_k$ それぞれを複合語のクラスとみなした k 個の単語生起確率値を計算し，それらの幾何平均で近似している．

$$P(w|c_1, \ldots, c_k) \approx \left(\prod_{i=1}^{k} P(w|c_i)\right)^{1/k}$$

2.4 大規模 Web 辞書との統合

Google 日本語入力では，Web から自動的にマイニングされた Web 辞書を内蔵することで芸能人の名前といった単語の変換精度を向上している[7]．これまでのかな漢字変換システムでは，医療辞書やトレンド辞書といった拡張辞書はデフォルトで組み込まれておらず，ユーザが必要に応じて有効にする必要があった．ただやみくもに語彙を増やすだけでは副作用によって平均的な変換精度が低下するためである．Web 辞書は，Mozc の言語モデル学習プロセスとは独立した手法で生成されており，副作用の影響が考慮されていない．そのため，同辞書をそのままユーザ辞書に登録するだけでは副作用による精度低下が避けられない．この問題を解決するために以下の方法で既存の言語モデルと Web 辞書を統合している．

1. Web 辞書のみを使い最長一致法に基づく単語分割で単語生起確率を推定する．
2. Web 辞書から作られた生起確率と既定の確率値を内挿し新しい確率値を得る．
3. Web 辞書の各エントリーと同一の読みを持つ単語列を Web コーパスからマイニングし，その単語列の頻度が十分大きければ複合語化する．

Web 辞書を形態素解析器の辞書に登録し一つの言語モデルを構築する方法も考えられるが，Web 辞書の追加が形態素解析の結果に悪影響を及ぼす可能性が否定できないため，Web 辞書のみの独立した言語モデルを構築している．3. の処理は，「花より男子 (だんご)」が Web 辞書にあるために，「花より団子」が変換できなくなるといった典型的な副作用を低減させるための処理である[8]．

2.5 実験

複合語化と大規模 Web 辞書の効果を調べるために，4 つの異なる条件のもとで，変換精度を比較した．結果を表 1 に示す．評価には Anthy[9]で使用されている教師データ[10]，評価基準には BLEU[2] を用いた．

注目すべきは，Web 辞書をユーザ辞書に登録する手法は，Web 辞書を全く用いない手法に比べて精度が低下している点である．確率値を正しく推定することなしに，単純に語彙を増やすだけでは副作用が避けられない裏付けになっている[11]．複合語化の効果は，Web 辞書のそれよりも高いことが分かる．

[6] このような拡張は ChaSen や MeCab にも実装されている．

[7] Web 辞書はオープンソース版 Mozc には組み込まれていない．

[8] 「花より団子」は 2 文節から構築されるため，「花より男子」が辞書に登録されるだけで「花より団子」が候補に現れなくなる．

[9] http://anthy.sourceforge.jp/

[10] このデータはユーザのフィードバックに基づくデータを多く含んでおり，実環境に近い評価データとなっている．

[11] プロジェクトの初期の段階では，既存のかな漢字変換システムのための拡張辞書を提供しようというアイデアもあったが，副作用の影響を予想していたため，その選択にはいたらなかった．詳しくは http://www.google.co.jp/intl/ja/ime/comic/ を参照．

表 1: 複合語化と Web 辞書の効果

システム	BLEU
複合語化 + Web 辞書 (言語モデル推定)	0.885
複合語化 + Web 辞書 (ユーザ辞書に追加)	0.863
複合語化 + Web 辞書なし	0.874
複合語化なし + Web 辞書 (言語モデル推定)	0.846

2.6 なぜ識別モデルを使わないのか

形態素解析や構文解析で成功を収めている識別モデルを適用することは技術的に可能ではあるが，Mozc では採用していない．その理由に以下が挙げられる．

- スケーラビリティー
  識別モデルのスケーラビリティーが向上したとはいえ，頻度を数えるだけの生成モデルに比べれば識別モデルのスケーラビリティーは低い．識別学習のモデル構築時間を生成モデル並にするには，学習用のコーパスを小さくするしかなく，これでは Web の統計を反映しにくくなる．
- N-best の提示
  識別モデルは，与えられたひらがな列に対して最尤なただ一つの解を出力することを目的としてパラメータの学習が行われる．しかし，かな漢字変換は，候補の N-best のランキングの妥当性も重要な評価指標である．識別モデルを用いた場合は，N-best の候補が妥当であるかという根拠付けが難しい．一方，生成モデルの場合は，生成確率，すなわち Web 上での頻度順に候補が並ぶことが期待できる．
- 解釈の難しさ
  生成モデルを用いた場合，パラメータ (確率) は頻度として解釈できるが，識別モデルの場合はパラメータ (重み) の説明付けが困難である．実環境では，変換というコンテキストを使い様々な付随機能を提供する．頻度は比較的簡単に計算できる統計量であるため，付随機能を同一の枠組みで取り入れやすい．
- 再変換への対応
  かな漢字変換には，かな漢字混じり文から読みを出力する再変換機能が要求されることがある．生成モデルの場合は，同時確率をベースにしているために，辞書引きの順番を入れ替えるだけで再変換が実現できる．

3 付加機能

本節では，かな漢字変換の利便性を高めるために必要な付加機能が Mozc の中でどのように実装されているかを紹介する．

3.1 文節への分割処理

多くのかな漢字変換システムは，変換結果を文節という単位に分割して提示する．文節分割は必須処理ではないが，N-best 候補が指数的に増える問題を避けるための運用上の工夫と言える．

Mozc における文節分割は，出力単語列を導出後の後処理として行われる．具体的には，隣接する言語クラス c_{i-1}, c_i が与えられたとき，それが文節境界になるかならないかを返す関数を実装し，それを順次出力単語列に適用することで，文節区切りを実現している．Mozc の言語クラス c_i は形態素情報から構成されるため，品詞や活用といった文法知識を使って関数を構築することができる．Mozc では，多くのかな漢字変換システムと同様，「内容語+機能語の 0 個以上の連続」を文節の基本単位としている．

3.2 文節境界の変更

かな漢字変換システムは常に正しい単語・文節境界の結果を返すとは限らないため, キーボード操作で単語・文節境界を変更する機能を提供している. Mozc では, 単語, 文節境界の変更機能を, 制約付きの Viterbi として実装している. 単純には, ユーザが与えた境界情報に適合しない単語を探索空間 (ラティス) 上から削除して[12] Viterbi アルゴリズムを動かせばよい. 例えば, 「ここで履物」を「ここでは着物」に変更する場合, 単語「履物」がラティスから削除される. 上記のように単語境界が変更される場合は単純であるが, 文節境界が変更される場合は少し工夫が必要である. ある単語列 a, b が一つの文節を構成しているとき, ユーザが a と b の間に文節境界を指定したケースを考えてみる. このとき, 単語 a, b を削除してしまうと, a, b それぞれが出現しなくなる. もしここで別の単語列 c, b があり, これらの間に文節境界がある場合, この単語列はユーザが与えた制約に適合するにもかかわらず候補として表示されなくなってしまう. このような場合, 削除するのではなく, a, b 間の連接確率にペナルティを与えソフトな制約としている.

3.3 N-best の提示

3.3.1 N-best の列挙・ランキング方法

文節分割の無いかな漢字変換システムの場合, 文頭から文末につながる全候補を A*アルゴリズムを使い確率順に列挙すれば N-best 候補が得られる [3]. しかし, 文節に分割されている場合, 各文節の N-best をどのように列挙しランキングすればよいかは自明ではない. 手法として以下が考えられる.

- 最適解制約
  着目文節に隣接する文節を最適解に固定して, N-best を列挙する. 両隣の文節の影響を強く受ける.
- 制約なし
  着目文節の両隣の文節の影響を考えず, 文節内だけで N-best を列挙する. 文節境界をまたぐ連接コストも考慮しない. 列挙漏れが少ない利点がある.
- 周辺化
  着目文節に隣接するすべての単語の周辺化確率をランキングに利用する. 例えば, 注目文節に左から単語 l が, 右から単語 r が隣接する場合, l の周辺確率を l の単語生起確率, r の周辺確率を r の単語生起確率とみなして, l から r につながる全ての候補を列挙する. この操作を隣接する全ての単語候補 l, r について行なう[13] 隣接する単語は l, r 以外にも複数あるため重複した候補が生成されるが, 周辺のコンテキストを考慮しながら注目文節の N-best 候補を列挙することができる.
- 擬似周辺化
  周辺確率の計算には Forward Backward アルゴリズムが必要なため, 実環境では速度上の問題がある. そこで, 周辺確率の代わりに, バックトラック確率を用いる. 単語 w のバックトラック確率とは, 文頭から単語 w まで到達するまでの最小確率であり, Viterbi アルゴリズムの過程で算出される.

Mozc の初期の実装では, 最適解制約を使っていた. 最適解制約は制約が強いため, 変換候補に適切な候補が出現しなくなるという問題が頻発した. 例えば, 「かれのいたい」という入力に対し, 「彼の痛い」が結果となり, 「遺体」が変換候補に出現しない問題などがあった. これは「彼の痛い」における「の」の品詞が「助詞-連体化」ではなく, 「助詞-格助詞」となったためである. この問題を解決するために, 制約なし戦略に切り替えた. 制約なし戦略では候補に存在しないという問題は減ったが, 文脈に即した N-best 解が得られないといった別の問題に遭遇した. 現在は, 擬似周辺化を用いている.

3.3.2 列挙の停止条件

N-best 列挙のもう一つの技術的課題に列挙の停止条件がある. 生成モデルをベースにしているため, 最適文節の確率と列挙対象の文節の確率の比を計算し, それが十分小さくなれば列挙を停止するという方法が考えられる. しかし, この方法では, 文法的には適合するが出現頻度の極めて小さい候補が列挙されない問題がある. 例えば, 表外漢字[14]や人名等はたとえ確率が低くてもユーザに提示するべきである. Mozc では上記の要求を踏まえ, 以下の二つのルールで停止条件を実装している.

- ホワイトリスト
  絶対にフィルタリングしてはいけない品詞列を事前にルール化しておく. 例えば, 人名を含む文節, 単漢字, 一単語から構成される文節などが該当する.
- 構造の生成確率
  構造の生成確率とは, 単語生起確率をすべて 1.0 として計算した生成確率である. 同じクラス列 (品詞列) を持つ単語列は, 同じ構造の生成確率を持つ. 構造の生成確率が最適解のそれと比べて十分小さくなった時に列挙を停止する. 同じような文法的構造を持った候補が誤ってフィルタリングされてしまう問題を抑制する効果がある.

3.4 学習機能

3.4.1 学習機能の実装

学習機能とは, ユーザが既定とは異なる候補を選択したときに, システムがその動作を学習し 2 回目以降は過去に選択された候補を自動的に提示する機能である. 実装には以下の二つの方法が考えられる.

- パラメータのオンライン更新
  パラメータ (重み) をオンライン更新の手法を用いて逐次変更する.
- 学習用言語モデルとの統合
  学習の結果から言語モデル (一般にはキャッシュモデル) を構築し, デフォルトの言語モデルと統合する.

Mozc の辞書および言語モデルは実行時に発生する様々な問題を回避するため, 書き込み不可のデータとしている. そのためパラメータの動的な更新は困難である. さらに, 学習結果は, かな漢字変換システム自身がバージョンアップしても引き継がれる必要がある. オンライン更新の場合は, 既定のモデルと学習用のモデルが同じパラメータ空間を共有するため, 後方互換性を常に気にしなければならない. 上記の運用上の制約から, Mozc では, 学習用言語モデルと統合するアプローチを採用している.

Mozc の学習機能は, 文節分割の後に個々の文節に対して順次適用される. 具体的には, 4 個の変数を記

[12] 単語生起確率を 0 に設定してもよい.

[13] A*を用いる場合, 右から連接するすべての単語をその周辺確率を初期値として Priority Queue に登録する. 左に連接する単語のいずれか一つに到達すると一つの候補が列挙される.

[14] にくむ→悪む など

憶する連想配列 d を使い擬似的なキャッシュモデルを構築している. ユーザがコンテキスト H のもとで, 読み R の候補を W に変更した場合, $d(W, R, q, H)$ を 1 にセットする. ただし, $q \in \{0, 1\}$ は, 学習の粒度を表す変数であり, 0 のときは文節全体の完全マッチのみ学習し, 1 のときは, 完全一致および内容語のみの学習も対象に含める[15]. q の値は, 学習の条件によって適宜変更する. 変換時には, 同連想配列を参照し同一の学習履歴が存在する時に限りランキングを変更する. 周辺コンテキスト H には, 文節のトライグラム, バイグラム, 単文節ユニグラム, ユニグラムの 4 パターンがあり, 文脈の深い順に学習の優先順位を決定する. ここで, 単文節ユニグラムとはユーザが単文節で入力したという状況を特別な文脈とみなしたユニグラムである[16]. 複数の文節が同じ優先順位を持つ場合は, キャッシュモデルと同様, 直近の学習を優先させる.

3.4.2 副作用の抑制

学習機能には, 再現性と適合性のトレードオフが存在する. ユーザの入力に完全一致した場合にのみ学習内容を適用すると, 再現性が低い. 一方, 学習履歴を過剰に適用するとコンテキストに則さない候補が提示されてしまう. コンテキスト H がユニグラム, もしくは内容語を含めて学習する ($q = 1$) 条件は副作用を生じやすい. そこで, 同条件での学習は, 既定の文節 W と学習対象の文節 W' が**交換可能**の時のみに限定している. 交換可能とは, 既定の文節 W を W' に置き換えても文脈上問題ないことを定義した制約である. 現状では, 以下の二つの条件の連言で近似している.

1. W と W' の内容語の品詞大分類が同一である.
2. W と W' の機能語の表層形が同一である.

この二つの制約を例を交えて説明する. ユーザが「きょうと」に対し「京都」という既定の候補ではなく, 「京と」を選択したとしよう. この場合, 内容語はそれぞれ名詞であり, 1 の制約は満たされている. 一方, 機能語は異なるため, 2 の制約に違反する. この結果, ユニグラム及び内容語のみの学習は行われない. ユーザが次に「きょう」や「きょうも」を入力しても「京」や「京も」が提示されることはない.

Mozc の初期の実装は, 交換可能制約が無かったため, 使えば使うほど意図しない候補が出てくる問題が生じた. 交換可能制約を設けることで同現象は改善された. 交換可能制約は自由度の高い方法である. 実装方法をより細かくチューニングしたり, 機械学習の手法を取り入れるなどすることで, 学習の精度の向上が見込めるであろう.

4 かな漢字変換の評価

かな漢字変換の評価には, 読みとそれに対応する正解かな漢字混じり文のペアからなる評価コーパスを用いて, システムの出力と正解を比較する方法が一般的である. Mozc でも, Web, 新聞記事, 検索クエリなどから評価コーパスを作成し, 上記のような**平均評価**を行っている. 評価基準として, 森らは, 最長共通部分列と完全一致率を用いている [6]. Mozc では, 機械翻訳の評価に使用される BLEU を採用している [2].

平均評価は, システムの平均的な振る舞いを調べたり, 他のシステムと比較を行ったり, 実験的な機能を評価する目的には長けているが, 実環境でのかな漢字変換システムの振る舞いを評価するには不十分なことが多い. 平均評価がユーザの利用シーンに即した評価にはなっていないからである. 例えば,1000 個の固有名詞を正しく変換できたとしても, 単純な変換, 例えば「明日」が変換できないだけでユーザはこのシステムは使いものにならないという評価を下してしまう. 一般にユーザはかな漢字変換の振る舞いを逐次予想しながら日本語を入力している. その予想が大幅にずれてしまうとユーザが感じる心理的負担が大きくなる. このずれを反映したコストセンシティブな評価が求められる. コストの定義は一般には困難である. そこで Mozc では, 絶対に誤ってはいけない変換例を収集し, その全ての事例にパスしないと出荷しないという**回帰評価**を行っている. 同事例は, ユーザからの誤変換レポートやパターンを用いて半自動的に構築している[17].

Mozc の初期のバージョンの出荷基準は, 平均評価のみに頼っていた. しかし, 言語モデル生成プロセスのバグ等が原因で, 「昨日 (さくじつ)」や「午後 (ごご)」が変換できないといった問題が連続した. その後, 出荷基準に回帰テストを取り入れることで, そのような問題は減少した.

5 おわりに

本論文では, かな漢字変換システム Mozc の設計と実装を紹介するとともに, 実環境で必要な個々の技術的課題を我々がどのように解決し, Mozc の中でどのように実装されているかを紹介した.

Mozc(Google の日本語入力) の最初のリリースから約 1 年 3ヶ月経つ. この間, 語彙の拡充など変換精度に関する更新を行ってきたが, 実際には, 最初のリリースと BLEU スコアに大きな変化はない. 本稿で紹介した付加機能における問題の修正, 回帰評価に基づく修正が主な変更点であり, これにより, ユーザが感じる違和感を軽減させることに成功した. かな漢字変換システムの難しさは, ひらがなを漢字に変換するだけではなく, 個々のコンポーネントを総合的に設計・評価しなければならないことにある. 本論文が, 今後のかな漢字変換研究のベースラインとなり, 変換以外のサブタスクにも目を向ける研究が増えることを切望する.

参考文献

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proc. of OSDI*, 2004.

[2] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proc. of ACL*, pp. 311–318, 2002.

[3] 永田昌明. 統計的言語モデルと n-best 探索を用いた日本語形態素解析法. 情報処理学会論文誌, Vol. 40, No. 9, 1999.

[4] 工藤拓, 賀沢秀人. Web 日本語 n グラム第 1 版. 言語資源協会発行, 2007.

[5] 小町守, 森信介, 徳永拓之. あいまいな日本語のかな漢字変換. 情報処理学会夏のプログラミング・シンポジウム, 2008.

[6] 森信介, 土屋雅稔, 山地治, 長尾真. 確率的モデルによる仮名漢字変換. 情報処理学会論文誌, Vol. 40, No. 7, 1999.

[15] 文節「彼は」の内容語は「彼」となる.

[16] 例えば, ユーザが「きょうと」を単独で入力し, 「教と」を選択した場合, 次に「きょうと」と単独で入力した場合は, 以前の入力が再現されるのが自然である. そのため, 単独で入力したという情報を特別な文脈としている.

[17] 「頻出動詞の基本形全て」等がパターンに相当する.