単語の意味役割を用いた自己修復表現の処理

船越 孝太郎　徳永 健伸　田中 穂積
東京工業大学 大学院情報理工学研究科

e-mail: {koh, take, tanaka}@cl.cs.titech.ac.jp

1　はじめに

音声対話はシステムは，自己修復あるいは言い直しと呼ばれている現象に対処できる必要がある．しかしながら既存の研究で提案されている手法では，自己修復を捉えるモデルに不足があり，我々の作成した疑似対話コーパス [6] に見られるような表現をカバーできない．また，自己修復を検知した後の不要語の除去処理に関しても十分な手法を与えていない．

本論文では，日本語の自己修復に対処するための新しい手法を提案する．この手法では，従来の手法では捉えられなかった自己修復表現を扱うために自己修復表現のモデルを拡張する．そして，表層のマッチングと意味レベルでのマッチングを融合した，自己修復表現の処理手法を提案する．

2　パーザ

提案する自己修復表現の処理は，構文解析と平行してパーザの上で行う．以下，本論文で用いるパーザとパーザの使用する辞書について説明する．

構文解析手法　構文解析は係り受け解析を基本として行う．我々の解析手法では内容語を重視し，機能語は内容語に付属するものと捉える．構文解析はスタックを用いて漸進的に行なう．

パーザは，解析の途中に生成される複数の構文仮説を，別々のスタックに保持する．スタックの各要素には，依存関係で表現された部分構文木が納められる (**図 1**)．各スタックの要素である部分構文木のルートになっている語を「ルート語」と呼ぶ[1]．

スタックに新しい単語がプッシュされると，**図 2** の手順で構文解析が行われる．図 2 の手順は，まずスタックのトップに文節を作り，その後係り受け解析を行う．例えば，　[(君が)|(玉を)>　という仮説スタック[2]に「押せ」という単語がプッシュされると，その後に助詞が続かなければ，

[(君が)|(玉を)|(押せ)>

[(君が)|((玉を) 押せ)>

[((君が)(玉を) 押せ)>

という 3 つの仮説スタックが生成される．

文法表現と辞書　本手法では，文節構造以外の文法に相当するものは，全て単語辞書の中に単語毎に用意する．

ある内容語 c1 がある内容語 c2 に係る時，c1 は c2 に対して特定の役割を担っていると考える．内容語 c1 が内容語 c2 に係ることができるかどうかは，c1 が c2 の辞書エントリに示された役割の内のどれかを満たすことができるかどうかによって決まる．

辞書の各エントリは**図 3** のように記述する．図 3 の第一行目は,「押して」という語が，左から順に,

・動詞である
・命令 (IMP+) である
・PUSH+という動作を表す素性を持つ

ということを表している．第二行目以降は,「押して」に係ることのできる語の制約を役割毎に示している．例えば第二行目の，<OBJECT>(目的格) という役割は，左から順に,

・必須格 (!が示す) である
・「押して」に対して 1 つしか存在しない
・「押して」に係るときは前方 (F) ／後方 (B) 依存のどちらでも良い (*はワイルドカード)
・名詞しかこの役割は取れない
・機能語は「は」「を」「も」「-(無標)」のどれか
・INSTANCE+素性を持っていなければならない

という事を表している．

パーザはこの辞書を用いて解析を行い，全ての係り受けに意味的な役割を割り当てる．

3　自己修復表現の処理

3.1　過去の研究の問題点

我々の疑似対話コーパス [6] の中に現れる自己修復表現に過去の研究で提案されていた手法を適用したところ，対処できない例が見られた．

対処できない理由は大きく分けて 2 つある．1 つは，自己修復のモデルの不備，もう 1 つは自己修復を検知した後の修正処理の不備である．

[1] 図 1 において四角で囲まれた語.
[2] [がスタックの底, | が要素間の区切り, > がスタックのトップ, () が係り受け関係を表す.

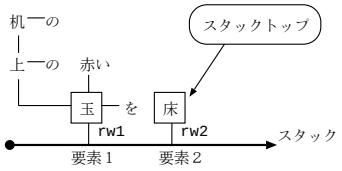

図 1: 「机の上の赤い玉を床」という入力を解析した場合のスタックの一例

スタックトップの要素を要素 2，スタックの上から 2 番目の要素を要素 1 とする (図 1 参照)．要素 1 のルート語を rw1, 要素 2 のルート語を rw2 とする．

1. プッシュされた語が機能語の場合
   (a) プッシュされた機能語である rw2 を rw1 に付属させる．既に機能語が付いている場合には,「に・も」のように連接が可能な場合を除いて，機能語が言い直されたと考えて新しい機能語に置換する．
   (b) 次にプッシュされる予定の語が内容語なら 2b へ．そうでなければ，終了．
2. プッシュされた語が内容語の場合
   (a) 次にプッシュされる予定の語が内容語なら次のステップへ．そうでなければ，終了．
   (b) スタックの上 2 つの要素を取り出す．
   (c) $rw2$ と $rw1$ が係り受け関係を作れるならば，スタックを複製して，次のステップへ．そうでなければ，取り出した 2 つの要素を元に戻して終了．
   (d) 片方の仮説 ($H1$) では，要素 1 と要素 2 をスタックに戻して処理を終える．もう片方の仮説 ($H2$) では，$rw1$ と $rw2$ の間に係り受け関係を作り，$rw1$ が $rw2$ に係るなら要素 2 を，$rw2$ が $rw1$ に係るなら要素 1 を，スタックに戻す．$H2$ には再び，2b からの手順を適用する．

図 2: 文節ベースの係り受け解析の手順

```
押して VERB IMP+ PUSH+
  !<OBJECT> 1 *  NOUN は|を|も|-  INSTANCE+
  !<AGENT>  1 *  NOUN は|が|も|-  ANIMATE+ INSTANCE+
   <TO>     1 *  NOUN に|へ|-     LOCATION+
   <FROM>   1 *  NOUN から        LOCATION+
```

図 3: 命令動詞「押して」の辞書エントリ

1. 言い足し
   (a) 構造隣接 「赤い玉を 大きい玉を 押して」
   (b) 構造非隣接 「赤い玉を 押して 大きい玉を」
2. 言い直し
   (a) 明示的
      i. 構造隣接 「赤い玉を ごめん 青い玉を 押して」
      ii. 構造非隣接 「赤い玉を 押して ごめん 青い玉を」
   (b) 非明示的
      i. 構造隣接 「赤い玉を 青い玉を 押して」
      ii. 構造非隣接 *「赤い玉を 押して 青い玉を」
3. リスタート
   (a) 明示的 「赤い玉を ごめん 馬は 前に 行って」
   (b) 非明示的 「赤い玉を 馬は 前に 行って」

図 4: 自己修復表現の分類

・P1 = P2
・N1 = N2 and P2 ≠ nil
・N1 ∼ N2 and P1 = nil

図 5: 名詞の置き換え条件． P1，P2 はそれぞれ名詞 N1，N2 に付いている機能語である．= は全く同じ語であること，∼ は N1 と N2 が同じ意味クラスに入ることを意味する．P1 = nil は，N1 には機能語が付いていないことを示す．

まず，自己修復のモデルの不備を説明する．既存の研究では，どれも Nakatani らによって提案された Repair Interval Model(RIM) に類するモデルを用いて自己修復を捉えている [5]．RIM は，入力文の上で，修復を受けるものを含む区間を REPARANDUM(以下 RPD)，修復するものを含む区間を REPAIR(以下 RP)，RPD と RP の間に現れるフィラーや休止，手がかり句 (cue phrase) [3]を含む区間を DISFLUENCY(以下 DF) としたときに，“ ... RPD DF RP ... ” の関係になるというモデルである．このモデルでは，手がかり句やフィラー以外の語が RPD にも RP にも含まれずに RPD と RP の間に現れることはないと仮定している．従って，RIM は「赤い玉を 押して 君の前の玉を」というような自己修復を扱えない．この例では動詞「押して」が，RPD である「赤い玉を」と RP である「君の前の玉を」の間に入っており，モデルの仮定を破っている．

次に，自己修復を検知した後の修正処理についての不備について説明する．従来の手法では，検知した RPD の部分を全て発話から削除する方法が用いられている．実際，英語，日本語に関わらずこの処理が正しく機能する場合は多い．しかし，「さっき押した赤い玉を 遠くに押したやつをもってきて」のような発話では，この方法は重要な情報を落としてしまう．この自己修復表現の修正処理は単なる削除だけでは不十分で，もっと複雑な処理が必要である．

3.2 自己修復表現の分類

本論文では，自己修復表現を図 4 のように分類する．

言い足しは，RPD の中に間違った情報は含まれていないものである．言い足しの中で，構造隣接に含まれるものが今までのモデルでも扱える部類である．構造非隣接に含まれるものが，3.1 節で述べた，今までのモデルでは扱えないものである．この言い足しの RPD と RP の間には**「RPD と RP は同じ物，同じ様態，同じ動作を示していなければならない」**という制限がある．

言い直しは，RPD の中に間違った情報が含まれているものである．言い直しは，手がかり句が挿入されている**明示的**な言い直しとそうでない**非明示的**な言い直しとに区別する．これは，非明示的な構造非隣接に分類される類いの発話を人間が通常することはなく，仮に発話されても人間の聞き手ですら混乱し，理解できないと仮定するからである．

リスタートは新たな発話を開始するもので，これも明示的なものと非明示的なものとに分ける．本論文では，紙面の都合上，リスタートの取り扱い

[3]自己修復を示す手がかり句は編集表現と呼ぶ.「ごめんなさい」,「じゃない」,「ちがう」等.

についての説明は省略する．

3.3 自己修復表現の処理

本手法では Hindle(1983) と同様に，自己修復個所の検知と修正を，構文解析と平行して行う．ただし，Hindle の解析手法が決定的であるのに対し，本手法では自己修復の処理においても複数の仮説を生成する．複数の仮説が生成される場合，仮説に尤度を与える必要があるが、その方法については本論文では省略する。

新しい単語が仮説にプッシュされて係り受け解析が行われると，係り受け解析が終わった仮説から順に自己修復表現に対する処理を受ける．自己修復の検出と修正の処理は，係り受け解析と同様にスタックの上 2 つの要素を処理することで行う．すなわち，スタックトップの要素の木が RP で，スタックの上から 2 番目の要素の木が RPD である (あるいは RPD を含んでいる[4]) と考える．要素を 2 つ取り出した後の処理は，言い足し及び言い直しとリスタートの 2 つで別れる．図 4 では，言い足しと言い直しは別に分けたが，実際の処理は似ている部分が多いので 1 つにまとめて処理をする．

言い足し及び言い直しの処理

まず，構造非隣接型の自己修復をどのように捉えるのかを決める．ここで，コーパスの観察などから，構造非隣接に分類されるタイプの発話は，“ ... RPD ... 動詞 DF RP ... ” という形しか取らず，RPD は，主格や目的格として必ず動詞に係っていると仮定する．この結果，構造非隣接で RPD と RP になることができる組は，名詞句の組か副詞句の組しか無いことになる．

この構造非隣接型の冗長性を解消するための解釈の仕方は 2 通り考えられるが，本論文では，RP が動詞に後ろから係ると考える方法を採用する．この時既に動詞に係っている RPD は，後から述べられた RP によって置き換えられると捉える．つまりこの解釈では，自己修復が倒置と組み合わさったものと考える．この解釈を選択した理由と，もう 1 つの解釈については [2] を参照されたい．

検出処理　ある仮説スタックが与えられると，検出は**図 6** の手順で行われる．図 6 の手順の中で使用されている置き換え条件とは，$rw1$ あるいは d_i^{rw1}(図 6 参照) と $rw2$ がそれぞれに付属している機能語も含めて満たさなければならない条件である．この条件が満たされるとき，$rw1$(あるいは d_i^{rw1}) は $rw2$ で置き換えられる．

置き換え条件の一部 (名詞に対する条件) を**図 5** に示す．

[4]構造非隣接の場合

1. スタックの上 2 つの要素を，上から要素 2，要素 1 として取出す．ただし，要素 1 が編集表現ならば，明示的な自己修復であるというフラグを立てて，要素 1 の下のスタックの要素を取り出して，それを要素 1 とする．要素 1 のルート語が $rw1$，要素 2 のルート語が $rw2$ である．
2. $rw1$ と $rw2$ の品詞が同じ場合 (構造隣接)
   (a) $rw1$ と $rw2$ が置き換え条件を満たせば，$rw2$ は $rw1$ の自己修復である可能性があると返して終了．
3. $rw1$ と $rw2$ の品詞が異なる場合 (構造非隣接)
   (a) $rw1$ が動詞でなければ，自己修復である可能性は無いと返して終了．
   (b) $rw2$ が名詞でも副詞でもなければ，自己修復である可能性は無いと返して終了．
   (c) $rw1$ に係っている内容語の中に $rw2$ と置き換え条件を満たすもの (d_i^{rw1} とする) があれば，$rw2$ は d_i^{rw1} の自己修復である可能性があると返して終了．ただしこの時，明示的でないならば $rw2$ と d_i^{rw1} は同じ物もしくは同じ様態を示していなければならない．

図 6: 言い足しと言い直しの検出手順

修正処理　既存の研究での修正処理は，単純に RPD を削除することで行われていた．しかしこれでは，RPD の中には存在するが RP では省略されてしまった情報まで削除してしまい好ましくない．そこで本手法では，$rw1$ をルートとする部分木 ($t1$) が $rw2$ をルートとする部分木 ($t2$) で置き換えられるときに，$t1$ には存在するが $t2$ には存在しない情報を，$t1$ から $t2$ に付け替える処理を行う．これは，具体的には $rw1$ に係っている語の内，$t2$ では省略された語を $rw2$ に付け替えることで行う．ただし，この時 $rw2$ に付け替えることで矛盾が起きるような語は付け替えないで捨てる．

語 $rw1$ をルートとする部分木 $t1$ を，語 $rw2$ をルートとする部分木 $t2$ で置き換えるとする．この時，$rw1$ に係っている m 個の語を $d_1^{rw1}, d_2^{rw1}, ..., d_m^{rw1}$ とする．$rw2$ に係っている n 個の語を $d_1^{rw2}, d_2^{rw2}, ..., d_n^{rw2}$ とする．この $t1$ と $t2$ に対して，付け替えを行う関数 $dmerge(rw1, rw2)$ の概要を**図 7** に示す．付け替え処理は，部分木のルートだけではなく，その中間ノードに対しても再帰的に適用される．例えば

$t1$:((赤い) 玉を) (君が) 押して)

$t2$:(((大きい) やつ) 押して)

という言い足し表現からは，

(((赤い) (大きい) 玉を) (君が) 押して)

という結果が生成される．図 7 のアルゴリズムで，語と語の対応を取るのに役割を使うのは，基本的には助詞が同じかどうかを見ることと同じである．しかしながら，助詞は省略される場合があるので，助詞を見るだけでは解決できない場合がある．また，もともと助詞がない場合 (名詞に係る形容詞など) も，すでに役割のラベルが与えられているので，新たに意味素性などの情報を用いて対応関係を計算する必要がなく，処理が効率的になる．

$dmerge(rw1, rw2)$: $rw1$ をルートとする部分木から $rw2$ をルートとする部分木への付け替え処理を行う

1. $d_1^{rw1}, ..., d_m^{rw1}$ と $d_1^{rw2}, ..., d_n^{rw2}$ の間で同じ語であることを基準に対応を取る (対応 1).
2. $d_1^{rw1}, ..., d_m^{rw1}$ と $d_1^{rw2}, ..., d_n^{rw2}$ の間で同じ役割を持つことを基準に対応を取る (対応 2).
3. 対応 1 と対応 2 それぞれのなかで 1 対 1 の対応であり，なおかつ対応 1 と対応 2 の間で食い違いがない対応をもつ語のペアを取り出し，$dmerge$ を再帰的に適用する．
4. $d_1^{rw1}, ..., d_m^{rw1}$ の内，対応 1・対応 2 のどちらからも対応づけを与えられなかったものだけ，$rw2$ にそのまま付け替える．
5. $rw2$ の意味クラスが $rw1$ の意味クラスよりも上位であれば，$rw2$ を $rw1$ で置き換える．

図 7: 付け替え処理のアルゴリズムの概略

4 提案手法のコーパスによる評価

本論文で提案した自己修復処理の手法を評価するために,「ソフトウェアロボットとの疑似対話コーパス」[6] に人手で適用した．このコーパスは，全 15 対話，532 発話を含み，71 個所の自己修復表現の内，正しく解決できないと思われるものが 16 個所存在した．

コーパスの規模と，人手での作業であることから，この結果の正解率などについては議論しない．その代わりに，対処できなかった冗長表現をタイプ別に分類し，それぞれに必要な処理を**図 8** に示す．

5 おわりに

本論文では，これまでの自己修復表現のモデルを拡張し，係り受け解析によって与えられる意味役割を利用した自己修復表現の修正処理を提案した．これにより，これまでに提案されていた手法よりも多くの自己修復表現を扱えることを示した．またコーパスでの評価により，自己修復表現に関する研究の今後の方向性を示した．

参考文献

[1] 伝 康治: 統一モデルに基づく話し言葉の解析, 自然言語処理, Vol.4, No.1 (1997)

[2] 船越 孝太郎, 徳永 健伸, 田中 穂積: 音声対話システムにおける不適格性の処理, 情報処理学会研究報告, 2002-NL-147 (2002)

[3] Hindle, D.: Deterministic parsing of syntactic non-fluencies, Proc. of 21st Annual Meeting of ACL (1983)

[4] 中野 幹生, 島津 明: 言い直しを含む発話の解析, 情報処理学会論文誌, Vol.39, No.6 (1998)

[5] Nakatani, C and Hirshberg, J.: A speech-first model for repair identification and correction, Proc. of 31st Annual Meeting of ACL(1993)

[6] ソフトウェアロボットとの疑似対話コーパス (http://tanaka-www.cs.titech.ac.jp/pub/qdc/)

- **[4] の分類 A の (II) に属するタイプ**
  例 「**黒の**，ガンダムが**黒の**後ろに行って」
  説明 [4] ではこのタイプは「X[RPD] の単語列が，Z[RP] の単語列の部分列になっている場合」と定義され ([ ] 内は著者の注),「二十分 愛甲石田まで二十分もかからないから」という発話が例として挙げられている．係り受け解析を基本とする場合，このタイプのものが解決できないできないことは [4] で指摘されている．このタイプを係り受け解析の手法の上で解決するためには，構造非隣接の扱いを拡張するか，パターンマッチングによる対応付けが必要になる．

- **単純な置き換えでは情報を損なうタイプ**
  例 「**青を**，その**ブロックを**押して」
  説明 この例では,「青」は青いものを示す代名詞として使われている．この「青」を単純に「ブロック」で置き換えてしまうと，折角話者が提供した「青色」という情報を失い，システムは曖昧性を正しく解決できない恐れがある．これを防ぐためには，単純に表層のシンボルの操作として自己修復を扱うのではなく，より深い意味の操作として自己修復を扱う必要がある．

- **高度な意味処理が必要なタイプ**
  例 「**白いのが入るぐらいに**映して，**白いのを**映して」
  説明 この例では，複雑な表現を，端的な表現で言い直している．表層や単語レベルでの意味役割では捉えられず，意味的な解釈ができないと,「**白いのが入るぐらいに**映す」ことと「**白いのを**映す」ことが同じことであることは判らない．

- **主辞の省略補完を行わないと解決できないタイプ**
  例 「**右に**押して，カメラから見て (**右に**)」
  説明 この例を解決するためには「右に」あるいは「右に押して」までを，何らかの推論によって補完できる必要がある．

- **見かけは普通の言い直しだが，単純な言い直しとしては解決できないタイプ**
  例 「カメラもうちょっと右から**映してくれる**，右に**回り込んでくれる**」
  説明 この例の場合，一見動詞句の非明示的な言い直しのように見えるが，単純に「映して」を「回り込んで」で置き換えてしまうことはできない．ここでは，話者は「右から映す」ための手段として「右に回り込む」ことを依頼している．このような発話を正しく理解するためには，談話解析までも構文解析と並列化した仕組みが必要である．そして，外界の状況やユーザの意図に応じた処理を行わなければならない．

図 8: 提案手法では解決できない自己修復表現