Internal Use Only (非公開)

TR-I-0259

対話翻訳における談話情報の管理と利用
-ASURA における談話管理機能 -
Managing Contextual Information for Discourse Interpretation

菊井 玄一郎
Gen-ichiro KIKUI

1992.3

梗概

対話翻訳システムを高度化するためにはいわゆる「文」の境界を越えた情報を適切に利用する必要がある。本稿では、これを実現するための最小限の処理系として開発された「談話翻訳管理システム (DTM:Discourse Translation Manager)」の処理内容、および、必要なインタフェース情報について述べる。本処理系の主な機能は (1) 処理単位の動的変更 (複数文の結合など)、(2) 談話構造解析モジュールと翻訳処理とのインタフェース、(3) 談話処理系作成のための開発支援環境の提供、である。

ATR 自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories

©(株) ATR 自動翻訳電話研究所 1992
©1992 ATR Interpreting Telephony Research Laboratories

目次

1 はじめに (Introduction) 1

2 処理系の概要 3
- 2.1 談話翻訳管理の位置付け 3
- 2.2 談話翻訳管理のモジュール構成 3

3 談話履歴を表すデータ構造 (Data Structure) 5
- 3.1 発話 (utterance) 5
- 3.2 談話 (discourse) 5
- 3.3 発話クラスタ (utterance_cluster) 5

4 翻訳処理の実行 8
- 4.1 入力処理 8
- 4.2 談話構造解析の呼び出し 9
- 4.3 翻訳単位制御 9
- 4.4 変換処理 9
- 4.5 生成処理 9

5 翻訳単位の制御 (DTUC:Translation Unit Control) 10
- 5.1 書き換えシステムの呼び出し 10
  - 5.1.1 書き換えの入力となる素性構造 10
  - 5.1.2 書き換え環境 11
- 5.2 書き換え結果の解釈 11
  - 5.2.1 出力素性構造の形式 11
  - 5.2.2 大域変数の更新 12
  - 5.2.3 変換・生成処理の制御 12
  - 5.2.4 出力素性構造の例 12
- 5.3 翻訳単位制御用素性構造書き換え規則の例 13

6 談話情報の提供 14
- 6.1 談話構造解析結果との関係 14
- 6.2 相手側言語での談話情報 14

7 翻訳処理システムの開発支援 16
- 7.1 処理の部分実行 (dtm_translate_reuse) 16
- 7.2 談話履歴の表示 (dtm_list_history) 16
- 7.3 談話履歴の外部ファイル保存 16

8 関数リファレンス 19
- 8.1 翻訳の実行 19
- 8.2 開発支援関係の関数 19
  - 8.2.1 談話履歴情報の書き出し 19
  - 8.2.2 談話履歴情報の読み込み 20
  - 8.2.3 変換入力の書き出し 20
- 8.3 談話情報 20

8.3.1 要求発話の発話番号 20
8.3.2 応答発話の発話番号 20
8.4 素性構造操作用ユーティリティ関数 21
8.4.1 アトミックタイプの素性構造の生成 21
8.4.2 素性構造の付加 21
8.4.3 標準的な文脈素性の作成 21
8.5 そのほかの関数 22
8.5.1 初期化 22
8.5.2 コマンドインタープリタ 22

A 外部モジュール呼び出しのインタフェース **24**
A.1 談話構造解析 (DIANA) で用意すべき関数群 24
A.2 変換 (TRANSFER) で用意すべき関数 25
A.3 生成 (GENERATION) で用意すべき関数 26

B 翻訳単位制御規則の記述例 **27**
B.1 処理の内容 27
B.2 翻訳単位制御アルゴリズム 28
B.3 素性構造書き換え規則の記述例 29

C 談話情報利用の例 **32**

1 はじめに (Introduction)

対話を構成する各発話は本質的に文脈依存である。従って、高品質な対話翻訳を実現するためにはいわゆる「文」を越えた処理が不可欠である。そこで、我々は、文脈を考慮した翻訳システムの実験ツールとして、以下のような機能をもった談話翻訳管理システム (Discourse Tranlation Manager:DTM) を作成した。

1. 翻訳単位制御機能

   日本語解析で適当とされる処理単位[1]が訳出するため (日英変換以降) の十分な情報を常に担っているとは限らない。従って、訳文品質を向上させるためには、解析から与えられた処理単位を越える範囲の情報を参照したり、複数の処理単位を一括して訳出したりすることが必要となる。

   ところが、会話翻訳において、あらかじめ談話を構成する全ての発話を見通してから訳出することは原理的に不可能である。これは、適当な時点で翻訳結果が与えられてはじめて、相手側言語での談話を進めることができるからである。

   もちろん、多くの発話はそれまでに与えられた情報を用いることで訳出できる。そこで、我々は、ある入力に対する解析結果が訳出可能なことがそれまでに与えれている入力の範囲から明らかなときには即座に訳出し、そうでない場合にはその処理単位の訳出を一時保留して、後続する入力が与えられてから訳出を行なうことにした。

   本処理系では、素性構造に対するパタンマッチ機能を備えた有限状態トランスデューサによって

   (a) 訳出の保留

   (b) 保留の解除 (保留されている文と新たな文とを別々に訳出)

   (c) 保留されている素性構造と新たな入力素性構造の結合 (保留されている文と新たな文とを一つの文として訳出)

   などの制御を行なう

2. 談話構造解析とのインタフェース機能

   適切な訳出を行なうためには関係する先行発話を参照する必要がある [1]。一般に談話を構成する発話の間には様々な関係が存在するが、訳出においてはターンテイキングの関係である「発話対」が有効である。

   本処理系では、別に作成されている談話構造解析モジュールを呼び出すことによって、処理単位の異なる 2 言語の発話列が扱える談話履歴データを作成し、変換・生成などのモジュールからの要求に応じて適切な談話情報を提供する。

3. 開発支援機能

   言語処理ソフトウエアの開発においては実行結果を見ながらのチューニングが不可欠である。

   これを効率良く行なうために、本処理系では談話翻訳実行時に作成された様々な素性構造や談話構造情報をファイルに保存して再利用したり、翻訳処理を部分的に再実行させたりする機能を備えている。

[1]多くの場合、「文」

以下ではまず2節で他モジュールとの関係を含む本処理の概要について解説し、3節で基本的なデータ構造について述べたあと、4-7節で処理の内容について述べる。8節はコマンド(関数)リファレンスである。

2 処理系の概要

2.1 談話翻訳管理の位置付け

本稿で述べる処理に関連する部分のモジュール構成を図1に示す。本稿で述べるのは図中の点線で囲まれた部分である。

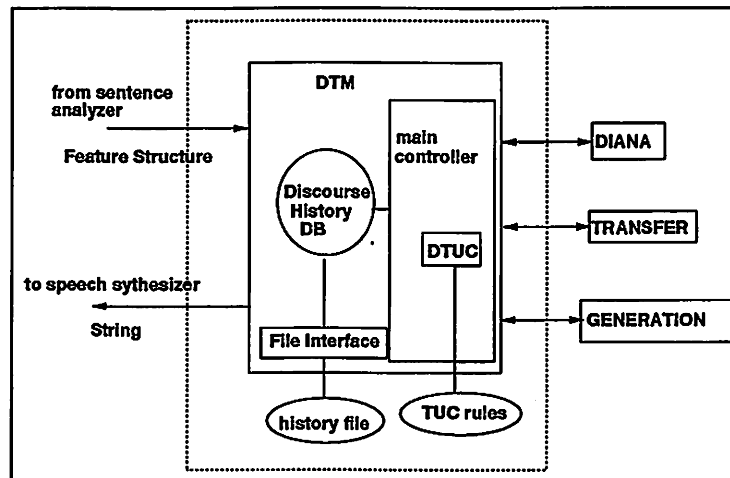

図 1: 処理系全体のブロック図

- DTM:Discourse Translation Manager

  解析以降の翻訳処理全体の制御を行なう部分である。

  入力文の素性構造(および表層文字列)を入力として受け取り、翻訳結果の文字列(および素性構造)を出力する。

- DIANA:DIscourse ANAlyser [2]

  日本語の談話構造を解析する。当面は会話におけるターンテイキングの構造である「発話クラスタ(発話対)」[3] の抽出に限定する。このモジュールは発話の意味構造から「要求」「応答」「確認」からなる局所的な談話構造を作成する。

- TRANSFER

  変換処理を行なう。原言語の素性構造を入力とし、目的言語にける可能な素性構造のリストを出力する。

- GENERATION

  生成処理を行なう。一つの素性構造を入力とし、可能な生成結果を出力する。

外部モジュールの提供すべき機能については付録 A を参照されたい。

2.2 談話翻訳管理のモジュール構成

談話処理モジュールの中心は

1. 翻訳処理全体を制御する「主制御 (main controller)」と

2. 談話履歴を保持する「談話履歴データベース (discourse history database)」

である。

前者は解析結果を受けとり、外部モジュールを呼び出すことによって目的言語の発話を出力する。このとき、翻訳単位制御モジュール (DTUC:Discourse Translation Unit Controller) を呼び出してその時点で翻訳すべき素性構造を決定する。

後者は談話履歴を管理するデータベースであり、ファイルインタフェースによって情報を外部ファイルに書き出したり外部ファイルから読み込んだりすることができる。

3 談話履歴を表すデータ構造 (Data Structure)

談話履歴は「発話」「談話」「発話クラスタ」という3つの基本的なデータ構造によって表現する。

3.1 発話 (utterance)

翻訳処理を構成するサブモジュール (解析、変換など) の処理単位を「発話」と呼び、発話オブジェクトというデータ構造で表現する。本処理系では、翻訳単位制御処理によって、日本語側と英語側とで処理単位が異なる可能性があるため、日本語発話と英語発話は対訳関係にある場合でも別々の発話オブジェクトで表現される。ここで、日本語側とは日本語解析と談話構造解析、英語側とは変換処理と生成処理を指す。

発話のデータ構造を図2に示す。このうち、「原言語を示すフラグ」は、その発話オブジェクトが原言語に属するときTとなる。translation には相手側言語で翻訳関係にある発話オブジェクトのリストが入る。このスロットがリストになっているのは一般に翻訳関係は一対一とは限らないからである。なお、リストの先頭要素は対訳関係にある発話オブジェクトのうちもっとも主要なものであるとする。

図中の「発話のクラス」「発話クラスタオブジェクト」については後述する。

NAME	TYPE	NOTES
id	fixnum	発話番号 (utterance id)
source	T or nil	原言語であることを示すフラグ (If the uttrance belongs to the source language, then T else nil)
string	string	表層文字列 (surface string)
fs	rws::node	意味素性構造 (feature structure)
cluster	dtm_u_cluster	発話クラスタオブジェクト (an utterance cluster object)
uclass	atom	発話のクラス (class of the utterance)
translation	a list of dtm_utterance	相手側言語で対応する発話オブジェクト

図 2: 発話を表すデータ構造: 構造体名 "dtm_utterance"

3.2 談話 (discourse)

協調的な対話を構成する発話の列をここでは「談話」と呼ぶ。すなわち、談話は発話オブジェクトの一次元列である。本処理系では日本語側と英語側と二つの談話を保持し、両言語間の発話の対応関係は発話オブジェクト間のリンクで表現する (図3参照)。

3.3 発話クラスタ (utterance_cluster)

談話を構成する発話の間には何らかの関係が存在する。お互いに関係付けられている発話同士はまとまって別の発話または発話のまとまりと関係付けられ、一般に、全体として「構造」をなす (「談話構造」)。本処理系では、発話の間の関係を、質問 - 応答、要求 - 受諾などの呼応関係 (ターンテイキングの関係) に限定し、この呼応関係をなすひとまとまりの発話を「発話クラスタ」と呼ぶ。発話クラスタを構成する発話は「要求発話」「応答発話」「確認発話」という3

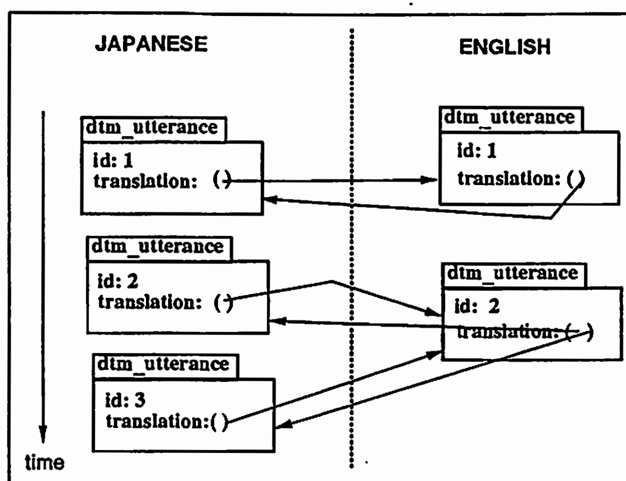

図 3: 談話データと対訳関係

つの発話のクラスに分類される。ここで、ある発話クラスタに別の発話クラスタが(時間的に)埋め込まれていても良いものとする[2]。

発話の中には、相手の発話を傾聴していることを表したり、間合いをとったり、といった、ターンテイキングを形成する機能が希薄なものが存在する。これらの発話は発話クラスタの要素にはならない。すなわち、本処理系ではどの発話クラスタにも含まれない発話が存在しても良い。このような発話を総称して「あいづち」とよぶ[3]。「あいづち」は発話クラスタには含まれないが便宜的に一つの「発話クラス」とする。

発話クラスタは一つの「発話クラスタオブジェクト」によって表現する。このデータ構造を図 4に示す。

NAME	TYPE	NOTES
demand	a list of dtm_utterance	要求発話の発話オブジェクトのリスト
response	a list of dtm_utterance	応答発話の発話オブジェクトのリスト
acknowledgment	a list of dtm_utterance	確認発話の発話オブジェクトのリスト

図 4: 発話クラスタを表すデータ構造: 構造体名 "dtm_u_cluster"

発話クラスタをなす要求発話と確認発話の話者は同一であり、要求発話と応答発話の話者は異なっていなければならない。これは誠実性など協調的な対話の条件から妥当な制約である。

また、各発話クラスの要素数は次の制約を満たすものとする。

1. 要求のスロットは必ず要素数が 1 である。

2. 確認のスロットが埋まっていれば、応答のスロットも埋まっている。

この制約は少し強すぎるかも知れないが、処理系の単純化のために設ける。

[2]ある発話クラスタを構成する発話を時間順に並べた列を考える。発話クラスタの(時間的な)埋め込みとは、この列の隣接する二つの発話の間に別の発話クラスタが開始し終了していることをいう

[3]歴史的な理由による。「あいづち」という言葉には「相手の発話に対する肯定応答」というニュアンスが含まれるが、ここではこのような「呼応的」なニュアンスは無視している

前節の発話のデータ構造のうち、「発話クラスタ」とはその発話の属する発話クラスタオブジェクトを格納し、「発話クラス」とは自らの属する発話クラスタにおける「要求、応答、確認、あいづち」の別を表すシンボルである。

なお、現在利用可能な談話処理系は日本語に対するものだけなので、この発話クラスタに関する情報は日本語の談話に対してまず与えられ、英語側は日本語側の情報から必要に応じて導出する。

4 翻訳処理の実行

本処理系の中心的な機能がここで述べる「翻訳処理の実行」である。翻訳処理の基本的な流れは次の通りである。

1. 入力処理 (原言語側発話オブジェクトの生成)
2. 談話構造解析処理の呼び出し (談話解析処理結果に基づいて、入力発話の属する発話クラスタを決定。既存の発話クラスタに属さない場合は、新たな発話クラスタオブジェクトの作成)
3. 翻訳単位制御 (訳出の保留など)
4. 変換処理の呼び出し
5. 生成処理の呼び出し

4.1 入力処理

新たな発話オブジェクトを作成し、原言語側談話に追加する。
入力はリスト形式で表され、この入力に応じて次に示すような発話オブジェクトを作成する。

1. 解析成功

   入力データ: (:aoutfile file_name string)

   発話オブジェクト:

   string string
   fs　file_name から読み込まれる素性構造

2. 同一話者による発話の終了

   入力データ: (:over)

   発話オブジェクト:

   string “”
   fs　[[prag [[u_state utterance_finish]]]]

3. 電話の呼び出しを行なっている場合

   入力データ: (:calling :caller office or client)

   発話オブジェクト:

   string “”
   fs　[[prag [[u_state calling]]]]

4. 電話を切った (切られた) 場合

入力データ: (:hangup)

発話オブジェクト:

```
string “”
fs      [[prag [[u_state hangup]]]]
```

4.2 談話構造解析の呼び出し

解析結果の素性構造を談話構造解析に送り、現在の発話の属する発話クラスタに関する情報を受けとって発話クラスタオブジェクトを作成したり更新したりする。なお「あいづち」という発話クラスはここで作成される (談話構造解析では「質問、応答、確認」という3つのクラスにしか分類しない)。

4.3 翻訳単位制御

談話構造解析の終った素性構造、および、翻訳状況を表す幾つかの大域変数もとにその時点で翻訳すべき素性構造を決定する。本モジュールはメモリを持ち、訳出を一時保留したり、保留されている素性構造と新たに入力された素性構造を結合して新たな素性構造を作ったりする。

もし、何も出力されなかった場合には、変換以降の処理は行なわず解析結果 (入力) 待ちとなる。

この処理については次節で詳しく述べる。

4.4 変換処理

翻訳単位の制御によって作成された訳出すべき素性構造を変換処理に送る。

このとき、目的言語側の発話オブジェクトを作成して、目的言語側の談話に追加する。

4.5 生成処理

前節の変換処理によって作成された素性構造を入力として生成処理を起動する。生成結果を目的言語側の発話オブジェクトの文字列スロットに書き込む。

5 翻訳単位の制御 (DTUC:Translation Unit Control)

本モジュールでは、先行する文の処理状況を表す「内部状態」および「解析結果メモリ」を保持し、これらの履歴情報と入力文Sの素性構造を参照することによって、変換・生成に渡すべき素性構造を作成する。本処理は素性構造に対するパタンマッチングの機能を持つ「素性構造書き換えシステム [3][4]」を用いて実現する。

5.1 書き換えシステムの呼び出し

書き換えシステムを呼び出す時に必要な情報は

1. 書き換えの入力となる素性構造と
2. 書き換え環境

である。

5.1.1 書き換えの入力となる素性構造

素性構造書き換えシステムに入力として与える素性構造は、談話構造解析によって「発話のクラス」の付与された日本語解析結果と大域変数 *dtuc_memory* に存在する素性構造を結合させたものである。

結合は、前者を first 素性、後者を rest 素性の素性値とした素性構造を作成することによって行なう。

書き換え処理に渡される素性構造 (入力) の形式を図 5に示す。

```
<input> ::= [[first <current_udata>]
             [rest  <feature structure>]]

<current_udata> ::= [[cluster <cluster_info>
                      [id 現在の発話の発話 ID(Source Lang.)]
                      [semprag <semprag>]]

<cluster_info> ::= [{[demand <id>]}
                    {[response <id>]}
                    {[acknowledge <id>]} ]


<semprag> ::= [[prag ....]
               [sem .....]]

<feature structure> ::= *dtuc_memory* にバインドされている素
性構造

({} で囲まれた要素はあってもなくてもよい)
```

図 5: 翻訳単位制御の入力

semprag 素性には談話構造解析済みの原言語の素性構造、あるいは、言語外情報素性がが入る。前者は基本的には、解析結果の素性構造であるが、prag 素性に発話のクラスを示す uclass

素性が付加されていることが異なる。cluster_info は発話クラスタに関する情報を表す。ここには、現在の発話の属する発話クラスタの各クラスの発話の発話 ID が記述される。なお、一つの発話クラスに複数の発話が存在する時には主たる発話の ID を記述する。

[例]

```
[[SEM [[RELN  はい-AFFIRMATIVE]
       [ASPT  -]
       [AGEN !X1[[LABEL  *SPEAKER*]]]
       [RECP !X2[[LABEL  *HEARER*]]]]]
 [PRAG [[SPEAKER !X1]
        [HEARER !X2]
        [UCLASS RESPONSE]]]  <== 発話のクラス
```

また、後者は話者交替を表す素性構造など、直接対応する狭義の音声信号のない情報である。これは理想的にはポーズ、抑揚などを含めた広い意味での物理信号 (知覚情報) から得られるべきものである。

```
[[prag [[u_state utterance_finish]]]]
```

< feature structure > には大域変数 *dtuc_memory* の素性構造がそのまま入る。

5.1.2 書き換え環境

書き換え環境は談話翻訳制御を示す (:PHASE :DTUC) と大域変数 *dtuc_u_state* によって保持されている「状態」とから成る。

環境属性	値
:phase	:DTUC(翻訳単位制御を表す)
:u_state	大域変数 *dtuc_u_state* の内容 (初期状態は: NORMAL)

5.2 書き換え結果の解釈

翻訳単位制御系は書き換え結果の素性構造を解釈して、

1. 内部 (大域) 変数の更新

2. 変換・生成に渡すべき素性構造の出力

を行なう。

5.2.1 出力素性構造の形式

書き換えの出力として想定される素性構造を図 6に示す。

図の next_state 素性および tuc_memory 素性が内部変数の更新に用いられ、actions 素性が変換以降の処理の制御に用いられる。

```
<output> ::=  [[next_state <next_state>    ]
               [memory <feature structure> ]
               [actions  <actions>     ]]

<actions> ::= [[A1 <action>]
               [A2 <action>] ...]

<action> ::= [[transfer [[id <id>]
                          [semprag <semprag>]]]]

<id>  ::=  symbol |
           [[main symbol]
            [sub <id>]]
```

図 6: 翻訳単位制御の出力素性構造

5.2.2 大域変数の更新

1. 状態 (*dtuc_u_state*)

   next_state 素性の値を表すシンボルがこの変数に代入される。もし、この素性が存在しない場合、状態は更新されない。

2. 翻訳単位制御用一時メモリ (*dtuc_memory*)

   memory 素性の値がそのままこの変数に代入される。もし、この素性が存在しない場合、メモリは更新されない。なお、処理系は指定された素性値をそのまま大域変数に保存するだけで push 操作などは行なわない。

5.2.3 変換・生成処理の制御

actions は変換以降の動作を指定する素性であり、動作の各ステップを An 素性 (n は整数、$0 \leq n$) の素性値として記述する。An の n は処理の順序を示す。

現在、指定可能な動作は transfer のみである。

1. transfer

   素性構造を変換部に送る。transfer は素性の素性値は id 素性と semprag 素性からなり、前者は対応する原言語側の文番号、後者は変換すべき素性構造である。もし、複数の原言語文が一つの semprag に含まれている場合の文番号素性は、原言語文の主となる方の文番号を main、従となる方を sub 素性として表現する。sub の方は再帰的に埋め込み可能である。

```
[[transfer [[id [[main 2]
                 [sub 1]]]
            [semprag ....]]]]
```

5.2.4 出力素性構造の例

出力素性構造の例を図 7 に示す。この例では、新たな状態が:NORMAL であり、保存すべき情報が空である。訳出に関しては、まず fs1 を原言語文 1 に対応する翻訳として訳出し、次に、fs2 を原言語文 3 に対応する翻訳として訳出することを示している。

```
[[actions [[a1 [[transfer [[id 1]
                           [semprag <fs1>]]]]]
          [[a2 [[transfer [[id 3]
                           [semprag <fs2>]]]]]
```

図 7: 出力素性構造の例

5.3 翻訳単位制御用素性構造書き換え規則の例

翻訳単位制御用素性構造書き換え規則の例を付録 A に示す。

6 談話情報の提供

本処理が他のモジュールに対して、提供できる情報は次の 3 つである。

1. 与えられた発話 ID の原言語および目的言語側の素性構造
2. 与えられた発話 ID の発話のクラス
3. 与えられた発話 ID の属する発話クラスタに属する別のクラスの発話 (の発話 ID)。

与えられた発話の発話クラス	提供可能な発話のクラス
:demand	null
:response	:demand
:acknowledge	:demand , :response
:aiduchi	null

6.1 談話構造解析結果との関係

本処理の提供する談話構造情報は基本的に談話構造解析モジュール (DIANA) からの出力である。談話構造解析モジュール出力と本処理系からの出力の違いは「あいづち」の扱いである。前者は発話を「要求、応答、確認」の 3 つに分類し「あいづち」というクラスは生成しない。これに対して後者は「応答クラス」に属する発話の一部のうち、対応する要求発話のない発話を「あいづち」クラスと認定する。

このアルゴリズムを次に示す。

```
class ← (DM_uclass id)
if class = :response
  then if ((surface id) ∈ *aiduchi-surface*) and ((DM_demand id) = nil)
          then :aiduchi
          else return :response
  else return class
```

DM_uclass, DM_demand は付録を参照されたい。

6.2 相手側言語での談話情報

現在、発話クラスタ構造は日本語談話に対してのみ与えられている。従って、英語発話の談話構造に関する情報が必要な場合には、対応する日本語談話の情報をそのまま用いることにする。

たとえば、談話履歴が図 8のようになっていたとしよう。この場合、d の発話のクラスは、対応する日本語側発話 a の発話のクラスである demand となる。発話 e のように、もし、対応する日本語発話が複数ある場合には、主節に対応する日本語発話の情報を優先して用いる。また、e の発話に対する要求発話は、日本語側において c に対する要求発話の翻訳である d となる。

この機能を用いて生成処理に文脈の扱いを導入した例を付録に示す。

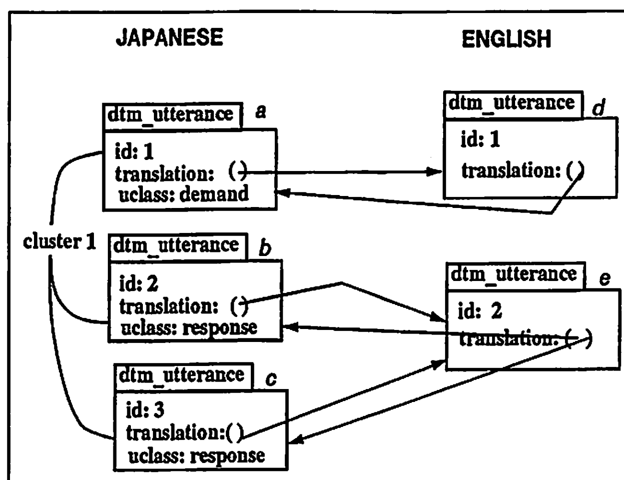

図 8: 談話クラスタ情報の推論

7 翻訳処理システムの開発支援

7.1 処理の部分実行 (dtm_translate_reuse)

すでに存在する談話履歴データを用いて、翻訳処理を部分的に実行することができる。
再実行できる処理は

1. 全過程 (談話解析以降)
2. 翻訳単位制御以降
3. 生成処理のみ

の3種類である。処理系は再実行に関係する談話履歴データをクリアしてから各処理を実行する。

7.2 談話履歴の表示 (dtm_list_history)

談話履歴の内容を表示する (図9)。

```
$> (dtm_list_history)
==== JAPANESE SIDE =====
ID:4 UCLASS:RESPONSE DEMAND:(2) RESPONSE:(3 4) TRANSLATION:(3) ++ そうです
ID:3 UCLASS:RESPONSE DEMAND:(2) RESPONSE:(3 4) TRANSLATION:(3) ++ はい
ID:2 UCLASS:DEMAND DEMAND:(2) RESPONSE:(3 4) TRANSLATION:(2) ++ そちらは会議事務局ですか
ID:1 UCLASS:DEMAND DEMAND:(1) RESPONSE:NIL TRANSLATION:(1) ++ もしもし
==== ENGLISH SIDE =====
ID:3 UCLASS:RESPONSE DEMAND:NIL RESPONSE:NIL TRANSLATION:(4 3)++Yes, it is.
ID:2 UCLASS:DEMAND DEMAND:NIL RESPONSE:NIL TRANSLATION:(2) ++Is this the conference office?
ID:1 UCLASS:DEMAND DEMAND:NIL RESPONSE:NIL TRANSLATION:(1) ++Hello.
```

図 9: 談話履歴の表示例

7.3 談話履歴の外部ファイル保存

処理系内部で管理している談話履歴は外部ファイルに保存したり、ファイルから処理系内に読み込んだりすることができる。

ファイルは原言語側のものと目的言語側のものからなり、それぞれテキスト形式に変換された発話オブジェクトを発話の順に書き出したものである。

談話履歴ファイルの日本語側の例を図10、英語側の例を図11に示す。

```
(DTM_UDATA
  :STRING "もしもし"
  :ID 1  :SOURCE T :UCLASS :DEMAND :CLUSTER 0
  :FS " [[PRAG [[HEARER !X2[[LABEL *HEARER*]]]
         [SPEAKER !X1[[LABEL *SPEAKER*]]]
         [UCLASS DEMAND]]]
  [SEM [[RELN もしもし-OPEN_DIALOGUE]
        [AGEN !X1]
        [RECP !X2]
        [ASPT -]]]]"
  :TRANSLATION (1))

(DTM_UDATA
  :STRING "そちらは会議事務局ですか"
  :ID 2  :SOURCE T :UCLASS :DEMAND :CLUSTER 1
  :FS " [[PRAG [[RESTR [[IN [[FIRST [[RELN POLITE]
                                    [AGEN !X4[[LABEL *SPEAKER*]]]
                                    [RECP !X3[[LABEL *HEARER*]]]]]
                             [REST !X7[]]]]
                        [OUT !X7]]]
         [ASPE [[IN []]
                [OUT []]]]
         [HEARER !X3]
         [PRSP-TERMS [[IN []]
                      [OUT []]]]
         [SPEAKER !X4]
         [SPECIFIC [[IN []]
                    [OUT []]]]
         [TOPIC [[IN [[FIRST [[FOCUS !X6[[PARM !X1[]]
       ................................
                                    [TOPIC-MOD HA]]]
                             [REST []]]]
                        [OUT []]]]
         [UCLASS DEMAND]]]
  [SEM [[RELN S-REQUEST]
        [AGEN !X4]
        [RECP !X3]
        [OBJE [[RELN INFORMIF]
               [AGEN !X3]
               [RECP !X4]
               [OBJE !X5]]]]]]"
  :TRANSLATION (2))

(DTM_UDATA
  :STRING "はい"
  :ID 3  :SOURCE T :UCLASS :RESPONSE :CLUSTER 1
  :FS " [[PRAG [[HEARER !X2[[LABEL *HEARER*]]]
         [SPEAKER !X1[[LABEL *SPEAKER*]]]
         [UCLASS RESPONSE]]]
  [SEM [[RELN はい-AFFIRMATIVE]
        [AGEN !X1]
        [RECP !X2]
        [ASPT -]]]]"
  :TRANSLATION (3))
```

図 10: 日本語談話履歴ファイル

```
(DTM_UDATA
  :STRING "Hello."
  :ID 1  :SOURCE NIL :UCLASS :DEMAND :CLUSTER NIL
  :FS ( " [[PRAG [[GEN_LC []]
         [HEARER !X1[[LABEL *HEARER*]]]
         [SPEAKER !X2[[LABEL *SPEAKER*]]]
         [UCLASS DEMAND]]]
  [SEM [[RELN PHATIC]
        [AGEN !X2]
        [RECP !X1]
        [OBJE [[RELN HELLO-INTERJ-1]
               [AGEN !X2]
               [RECP !X1]
               [ASPT -]]]]]]")
  :TRANSLATION (1))

(DTM_UDATA
  :STRING "Is this the conference office?"
  :ID 2  :SOURCE NIL :UCLASS :DEMAND :CLUSTER NIL
  :FS ( " [[PRAG [[GEN_LC [[PREVIOUS [[PRAG [[HEARER !X3[[LABEL *HEARER*]]]
                                           [SPEAKER !X4[[LABEL *SPEAKER*]]]
                                           [UCLASS DEMAND]]]
                                    [SEM [[RELN PHATIC]
                                          [AGEN !X4]
                                          [RECP !X3]
                                          [OBJE [[RELN HELLO-INTERJ-1]
                                                 [AGEN !X4]
                                                 [RECP !X3]
                                                 [ASPT -]]]]]
                                    [SYN [[CAT S-TOP]
                                          [INV -]]]]]]]
         [HEARER !X6[[LABEL *HEARER*]]]
         [POLITENESS [[AGEN !X5[[LABEL *SPEAKER*]]]
                      [RECP !X6]
                      [DEGREE 1]]]
         [SPEAKER !X5]
         [SPECIFIC [[IN []]
                    [OUT []]]]
         [TOPIC [[FOCUS !X8[[PARM !X7[]]
            ...............
                [TOPIC-MOD HA]]]
         [UCLASS DEMAND]]]
  [SEM [[RELN QUESTIONIF]
        [AGEN !X5]
        [RECP !X6]
        [OBJE [[RELN BE-VI-5]
               [OBJE [[PARM !X2[]]
                      [RESTR [[RELN THIS-PRON-1]
                              [ENTITY !X2]]]]]
               [ASPECT [[PERF -]
                        [PROG -]]]
               [ASPT !X9]
               [IDEN [[PARM !X1[]]
                      [RESTR [[RELN NAMED]
                              [ENTITY !X1]
                              [IDEN CONFERENCE_OFFICE-IDIOM-1]]]]]
               [TENSE PRESENT]]]]]]")
  :TRANSLATION (2))
```

図 11: 英語談話履歴ファイル

8 関数リファレンス

8.1 翻訳の実行

---

dtm_translate *id fs string* [*Function*]

---

args	id	発話 ID
	fs	素性構造
	string	表層文字列
returns	目的言語の表層文字列のリスト	

fs を解析結果の素性構造として、談話解析、翻訳単位制御を含む翻訳の全過程を実行する。

---

dtm_translate_file *file* [*Function*]

---

args	file 解析結果ファイル
returns	T

解析結果をファイル (file) から読み込み全処理過程を実行。ファイルはテキスト形式で、表層文字列と対応する素性構造 (のプリント形式) を空行を区切りとして交互に並べる。なお、文字列も素性構造も引用符 (" や ') で囲んではならない。

---

dtm_translate_reuse *&key start phase* [*Function*]

---

args	start	再実行する文の範囲を原言語 (日本語側)ID で指定する。default は 1
	phase	再実行する処理の指定
returns	t	

すでに読み込まれている素性構造を用いて、翻訳を (部分的に) 実行。

「再実行する処理の指定」は次の通り。

:all 全過程の実行 (default)

:transfer 翻訳単位制御以降の処理の実行

:generation 生成処理のみの実行

8.2 開発支援関係の関数

8.2.1 談話履歴情報の書き出し

---

dtm_save *file* [*Function*]

---

args	file:string　ファイル名。
returns	t

原言語側履歴情報は、引数として与えられたファイル名に拡張子.jhist を付加したファイルに保存され、目的言語側の履歴情報は.ehist を付加したファイルに保存される。

8.2.2 談話履歴情報の読み込み

---

dtm_load *file* [*Function*]

---

args	file:string　ファイル名。
returns	t

原言語側履歴情報は、ファイル名に拡張子.jhist を付加したファイルから読み込まれ、目的言語側の履歴情報は.ehist を付加したファイルから読み込まれる。

8.2.3 変換入力の書き出し

---

dtm_write_transfer_input *&optional file &key if-exists* [*Function*]

ファイル名を指定して呼び出した時、変換に渡される素性構造のファイル出力を開始する。ファイル名を指定せずに呼び出した時、この出力を終了する。

:if-exists パラメータは出力先ファイルがすでに存在している場合の動作の指定で、:new-version か:append。default は: new-version。

8.3 談話情報

8.3.1 要求発話の発話番号

---

dtm_get_demand *id lang* [*Function*]

発話番号が id の発話に対する要求発話を返す。lang は現在考慮の対象としている発話の属する言語の指定で:j または:e である。たとえば、

(dtm_get_demand 3 :e)

は英語側の発話 id が 3 の発話に対する英語側要求発話の id を返す。なお、要求発話が存在しない場合の戻り値は nil である。

8.3.2 応答発話の発話番号

---

dtm_get_response *id lang* [*Function*]

発話番号が id の発話に対する応答発話を返す。lang は現在考慮の対象としている発話の属する言語の指定で:j または:e である。なお、応答発話が存在しない場合の戻り値は nil である。

8.4 素性構造操作用ユーティリティ関数

8.4.1 アトミックタイプの素性構造の生成

---

dtm_make_atomic_fs *val* [*Function*]

---

val を値とするアトミックタイプの素性構造を生成する。

8.4.2 素性構造の付加

---

dtm_add_fs *value feature-name fs* [*Function*]

もし、素性構造 fs が素性名 feature-name の素性を持てば、この素性の値を value にする。そうでなければ、素性構造 fs に素性名 feature-name 素性値 value の素性を追加する。従って、後者の場合、この操作は素性構造 fs と素性構造 [[feature-name value]] を単一化する操作と同等である。

```
>(rws::pprint-fs *fs1*)
[[A  X]
 [B  Y]]

>(rws::pprint-fs *fs2*)
[[D Z]]

>(dtm_add_fs *fs2* 'C *fs1*)
[[A  X]
 [B  Y]
 [C  [[D Z]]]]
```

8.4.3 標準的な文脈素性の作成

---

dtm_make_standard_local_context *id accessfn lang* [*Function*]

引数:

- id
  この id で示される発話に対する文脈情報を生成する。
- accessfn
  引数として文番号を取り、その文番号に対してあらかじめ保存されている素性構造を返す関数。
- lang
  対象としている発話 id の属する言語を与える (:e or :j)。

戻り値: 次のような素性構造である。

[[PREVIOUS idで示される発話の直前の発話 id を引数として addessfn によって検索され
る素性値]
[DEMAND idで示される発話に対する要求発話の ID を引数として addessfn によって検索され
る素性値]
[RESPONSE idで示される発話に対する応答発話の ID を引数として addessfn によって検索され
る素性値]]

なお、対応する発話 (id) が存在しない素性は省略する。図11の英語談話履歴ファイルにおいて ID2 の発話の GEN_LC 素性がこの関数によって作成された素性値である。この発話は要求発話であるから、DEMAND,RESPONSE の各素性は存在せず、直前発話の素性を表す PREVIOUS 素性のみが埋まっている。

8.5 そのほかの関数

8.5.1 初期化

---

dtm_init [*Function*]

談話履歴など全ての情報の初期化。

8.5.2 コマンドインタープリタ

---

dtm_send *method args* [*Function*]

メソッドと引数を解釈して個別関数を呼び出す。

メソッド名	引数	呼び出される関数
:je_translate	id fs	dtm_translate id fs
:calling		dtm_init
:over		dtm_over
:hang_up		dtm_end
:je_exec	id phase	dtm_transfer_reuse id phase

参考文献

[1] 菊井 玄一郎, 鈴木 雅実, " 表層表現の型を用いた対話文の生成について ", 情報処理学会自然言語処理研究会資料 81-11, 1991

[2] 山岡 孝行, 菊井 玄一郎, " 談話構造解析モジュールDIANA", ATR テクニカルレポート, TR-I-00256, 1992

[3] 山岡 孝行, 飯田 仁, " 文脈を考慮した音声認識結果絞り込み手法 ", 情報処理学会自然言語処理研究会資料 78-10, 1990

[4] 長谷川 敏郎, " 素性構造書き換えシステムマニュアル ", ATR テクニカルレポート, TR-I-0093, 1989

[5] 長谷川 敏郎, " 素性構造書き換えシステムマニュアル (改訂版)", ATR テクニカルレポート, TR-I-00187, 1990

A 外部モジュール呼び出しのインタフェース

A.1 談話構造解析 (DIANA) で用意すべき関数群

DM_analysis *id fs* [*Function*]

---

returns: :demand, :response, :acknowledge

発話番号が id, 素性構造が fs なる発話を談話に加え、 fs の属する発話のクラスを返す。

DM_uclass *id* [*Function*]

---

returns: :demand, :response, :acknowledge

id の属する発話のクラスを返す。

談話構造は表 1の情報伝達行為の分類に従って作成される。従って、「あいづち」を表す「はい」や「ええ」は demand のない response となる。

表 1: 情報伝達行為の分類 ([3] より引用)

1.Demand Class:	
ASK-ACTION	: 「*ACT* は *WH* ですか。」
CONFIRM-ACTION	: 「*ACT* (する／できるの) ですか。」
REQUEST-ACTION	: 「*ACT* して下さい。」
	: 「*ACT* していただけますか。」
	: 「*ACT* していただきたいのですが。」
OFFER-ACTION	: 「*ACT* します。」
	: 「*ACT* させていただきます。」
ASK-VALUE	: 「*OBJ* は *WH* ですか。」
	: 「*OBJ* をお願いします。」
	: 「*OBJ* を教えて／聞かせて下さい。」
	: 「*OBJ* を聞くことができますか。」
	: 「*OBJ* をお聞きしたいのですが。」
CONFIRM-VALUE	: 「*OBJ* は *VAL* ですか。」
	: 「*OBJ* は *VAL* ですね。」
ASK-STATEMENT	: 「*STA* は *WH* ですか。」
CONFIRM-STATEMENT	: 「*STA* ですか。」「*STA* ですね。」
GREETING-OPEN	: 「もしもし。」*
GREETING-CLOSE	: 「失礼します。」*「さようなら。」*
	: 「ありがとうございました。」

2.Response Class:	
INFORM-ACTION	: 「*ACT* して下さい。」
	: 「*ACT* しなくてはいけません。」
	: 「*ACT* です。」
INFORM-VALUE	: 「*OBJ* は *VAL* です。」
	: 「*VAL* を／が *STA* ます／です。」
INFORM-STATEMENT	: 「*STA* です (が)。」
	: 「*STA* したい (のですが)。」
AFFIRMATIVE	: 「はい。」*「そうです。」*
NEGATIVE	: 「いいえ。」*
	: 「(否定表現)。」
ACCEPT-ACTION	: 「わかりました。」*
	: 「*ACT* は問題ありません。」
REJECT-ACTION	: 「*ACT* できません。」
ACCEPT-OFFER	: 「ありがとうございます。」*
	: 「(よろしく) *ACT* 願います。」
REJECT-OFFER	: 「(いいえ) 結構です。」
GREETING-OPEN	: 「はい。」*
GREETING-CLOSE	: 「失礼します。」*「さようなら。」*
	: 「どういたしまして。」*
3.Confirm Class:	
CONFIRMATION	: 「わかりました。」*「そうですか。」*

表層表現内のイタリック文字は変項を表し、それぞれ *ACT* は ACTION に関する内容、 *OBJ* は OBJECT に関する内容、 *STA* は STATEMENT に関する内容、 *WH* は疑問詞を表す。また、／は選言を表し、() 内は付いても付かなくても良い表現である。

DM_cluster_demand *id1* [*Function*]

---

returns: id, nil, :demand

id1 で表される発話が:demand である場合:demand を返し、:response または:acknowledge である場合、対応する:demand 発話の id を返す。後者の場合で対応する:demand 発話がない場合は nil を返す。

DM_cluster_response *id1* [*Function*]

returns: id, nil, :response, :demand

id1 で表される発話が:demand また:response である場合:demand または:response を返し、:acknowledge である場合、対応する:response 発話の id を返す。(後者の場合で対応する:response 発話がない場合は nil を返す。)

要求発話と応答発話の対応関係は表 2の通りである。

表 2: 要求発話と応答発話の対応 ([3] より引用)

Demand Class	Response Class
ASK-ACTION	INFORM-ACTION
CONFIRM-ACTION	AFFIRMATIVE,NEGATIVE INFORM-ACTION
REQUEST-ACTION	ACCEPT-ACTION,REJECT-ACTION
OFFER-ACTION	ACCEPT-OFFER,REJECT-OFFER
ASK-VALUE	INFORM-VALUE
CONFIRM-VALUE	AFFIRMATIVE,NEGATIVE INFORM-VALUE
ASK-STATEMENT	INFORM-STATEMENT
CONFIRM-STATEMENT	AFFIRMATIVE,NEGATIVE INFORM-STATEMENT
GREETING-CLOSE	GREETING-CLOSE

A.2 変換 (TRANSFER) で用意すべき関数

TRJE_transfer *id fs* [*Function*]

---

returns: a list of fs (rws::node)

発話番号が id, 素性構造が fs なる発話の日英変換を行ない、変換結果の素性構造のリストを返す。変換に失敗した場合は空リストを返す。

A.3 生成 (GENERATION) で用意すべき関数

TRE_generation *id fs* [*Function*]

---

returns: a list of strings

発話番号が id, 素性構造が fs なる発話の英文生成を行ない、生成結果の文字列のリストを返す。

TRE_generation_core *id fs* [*Function*]

---

returns: a list of trees (s_node)

発話番号が id, 素性構造が fs なる発話の英文生成を行ない、生成結果の統語構造のリストを返す。

B　翻訳単位制御規則の記述例

B.1　処理の内容

「いいえ」および伝達行為タイプが response である「はい」を入力として受けとった場合には、この"文"の訳出を保留して、次の入力を待ち後続する文を考慮した処理を行なう。[4]後続した文として考えられるのは

1. 話者交替情報 (言語外情報)
2. 定型的応答表現
3. 反復応答表現
4. その他の表現

の4つである。以下でこれらに対応する処理について説明する。

1. 話者交替情報
   保留している「はい」「いいえ」をそのまま変換生成に送る。
2. 定型的応答表現
   定型的な応答表現で、ほとんどがメタレベルの表現である。
   [例]
   はい。　そうです。
   はい。　分かりました。
   いいえ。　違います。
   素性構造を結合して新たな素性構造を作成し (「はい」「いいえ」を後続する文の adjunct とし)、変換に送る。
3. 反復応答表現
   先行する疑問文の一部を反復するもの。ダ文応答も含む。
   [例]
   Q:　登録用紙は既にお持ちですか?
   A1:　はい。持っています。
   A2:　いいえ。 まだです。
   A3:　いいえ。 まだ持っていません。
   この表現は、2文の緊密度という点で先の「定型的応答表現」とこの後に示す「その他の表現」との中間に位置するものである。現時点では特に後者との自動判別が困難なため、後者と同様の扱いとする。

---

[4]次の例に示すような aiduchi の「はい」は特に後続の文と関係付けた訳出はしない。

今申し込むと参加料はいくらですか?
はい。 (aiduchi)
参加料はお一人 3500 円です。

4. その他の表現

   二つの文の発話タイプがともに response であり、これらに対応する demand 発話が同一のとき、二つの文の素性構造を結合して新たな素性構造を作成し (「はい」「いいえ」を後続する文の adjunct とし) 変換に送る。そうでないとき、二つの文の素性構造を入力の順で別々に翻訳処理に渡す。

B.2 翻訳単位制御アルゴリズム

前節で述べた処理を実現するためには:NORMAL, :SUSPENDING-HAI :SUSPENDING-IIE という 3 つの状態を用いて次のような遷移を行なえば良い。

1. :NORMAL(初期状態)

```
IF
   sem.reln = はい-affirmation and prag.uclass = response
THEN memory_data ← sent1
       NEXT_STATE ← :SUSPENDING-HAI
ELSE IF
          sem.reln = いいえ-negation and prag.uclass = response
THEN NEXT_STATE ← :SUSPENDING-IIE
ELSE
      sent1 ← 現在の文
      NEXT_STATE ← :NORMAL
```

2. :SUSPENDING-HAI

   [注意]「はい」とこれに後続する定型的応答表現は必ず要求発話を共有する。

```
IF sem.reln = はい-affirmation and prag.uclass = response
THEN NEXT_STATE ← :SUSPENDING-HAI
ELSE IF 現在の文の要求発話 id = 保留している文の要求発話 id THEN
          THEN sent1 ← 入力文に「はい」の素性構造 (memory_data) を付加語として加えたもの
          NEXT_STATE ← :NORMAL
          ELSE
                sent1 ← memory_data
                sent2 ← 現在の文
                NEXT_STATE ← :NORMAL
```

3. :SUSPENDING-IIE

```
IF
   sem.reln = いいえ-negation and prag.uclass = response
THEN GOTO :SUSPENDING-IIE
ELSE IF 現在の文の要求発話 id = 保留している文の要求発話 id THEN
          sent1 ← 入力文に「いいえ」の素性構造 (memory_data) を付加語として加えたもの
          NEXT_STATE ← :NORMAL
          ELSE
                sent1 ← memory_data
                sent2 ← 現在の文
                NEXT_STATE ← :NORMAL
```

B.3　素性構造書き換え規則の記述例

```
;------------------------------------------------------------
;
;   処理単位の調整　素性構造書き換え規則
;
;   Feature structure rewriting rules for
; translation unit control.
;
;------------------------------------------------------------
(in-package "USER")

(rws::REMOVE-RULES-IN-ENVIRONMENT '((:phase . :dtuc)))
;
; 初期状態  STATE = :NORMAL
;
(rws::defrwrule
on <first semprag prag uclass> demand in
                  :phase :DTUC :u_state :NORMAL :subphase :main
  in= [[first ?semprag_and_id]
       ?rest]
  out= [[next_state :normal]
        [actions [[A1  [[transfer ?semprag_and_id]]]]]]
end)

(rws::defrwrule
on <first semprag prag uclass> response in
                :phase :DTUC  :u_state :NORMAL :subphase :main
  in= [[first ?semprag_and_id]
       ?rest]
  set ?it to [[dummy empty]]
  -> ?input with :phase :DTUC :subphase :suspendp
  if it has dummy
     then
    out= [[next_state :normal]
    [actions [[A1  [[transfer ?semprag_and_id]]]]]]
     else
     out= ?it
     endif
end
)

;  acknowledge, aiduchi は demand と同様


;
;  suspending の決定
;
(rws::defrwrule
on <first semprag sem reln> はい-affirmative in
                        :phase :DTUC  :subphase :suspendp
   in= [[first ?semprag_and_id]
?rest]
  out= [[next_state :suspending_hai]
```

```
        [memory [[first ?semprag_and_id]
        ?rest]]
        [actions []]]
   end)

(rws::defrwrule
on <first semprag sem reln> いいえ-negative in
                        :phase :DTUC  :subphase :suspendp
   in= [[first ?semprag_and_id]
?rest]
  out= [[next_state :suspending_iie]
        [memory [[first ?semprag_and_id]
        ?rest]]
        [actions []]]
   end)
;
;  :U_STATE :SUSPENDING_HAI
;  「はい」を保留している状態
;
(rws::defrwrule
on <first semprag prag uclass>  demand in
                   :phase :DTUC  :u_state :SUSPENDING_HAI :subphase :main
  in= [[first [[id ?id_main]
       [semprag [[sem ?sem_main]
 ?rest0]]
       ?rest0]]
       [rest [[first [[id ?prev_id]
      [semprag ?prev_sem_prag]
      ?rest1]]
      ?rest2]]]
  out= [[next_state :normal]
        [MEMORY []]             ;; メモリのクリヤ
[actions [[A1  [[transfer
[[id ?prev_id]
 [semprag ?prev_sem_prag]]]]]
 [A2  [[transfer
[[semprag [[sem ?sem_main]
   ?rest0]]
 [id ?id_main]
 ?rest1]]]]]]]
  end
)
;
;
(rws::defrwrule
on <first semprag prag uclass> response in
                   :phase :DTUC  :u_state :SUSPENDING_HAI :subphase :main
  in= [[first [[id ?id_main]
             [cluster [[demand ?id1]]]
       [semprag [[sem ?sem_main]
 ?rest0]]]]
       [rest [[first [[id ?prev_id]
                    [cluster [[demand ?id2]]]
```

```
      [semprag @prev_sem_prag
       [[sem [[reln ?prev_sem_rel]
      ?rest1]]
?rest2]]]]
      ?rest3]]]
  if ?id1 is ?id2 then
  add { [response ?prev_sem_rel] } to ?sem_main
  out= [[next_state :normal]
        [MEMORY []]
[actions [[A1  [[transfer
[[semprag [[sem ?sem_main]
   ?rest0]]
 [id [[main ?id_main]
      [sub  ?prev_id]]]]]]]]]]
  else
  out= [[next_state :normal]
        [MEMORY []]
[actions [[A1  [[transfer
[[id ?prev_id]
 [semprag @prev_sem_prag]]]]]
          [A2  [[transfer
[[semprag [[sem ?sem_main]
   ?rest0]]
 [id ?id_main]]]]]]]
  endif
  end
)
;  :U_STATE :SUSPENDING_IIE
;  「いいえ」を保留している状態 (suspending hai を利用する)
;
(rws::defrwrule
on <first semprag prag uclass>  demand in
                    :phase :DTUC  :u_state :SUSPENDING_IIE :subphase :main
  -> ?input with :phase :DTUC :u_state :SUSPENDING_HAI :subphase :main
  out= ?it
  end
)
;
(rws::defrwrule
on <first semprag prag uclass> response in
                    :phase :DTUC  :u_state :SUSPENDING_IIE :subphase :main
  -> ?input with :phase :DTUC :u_state :SUSPENDING_HAI :subphase :main
  out= ?it
  end
)
;
;  aiduchi 等についても同様
```

C 談話情報利用の例

以下は生成処理に談話処理を導入するプログラム例である。以下の例では次のような手順で生成処理を行なう。

1. 素性構造のマージ
   生成しようとする文Uの直前の発話とUを含む発話クラスタに属する発話に対して、生成独自で保存している素性構造を検索し、これらを変換から渡される素性構造の prag 素性内の GEN_LC 素性に埋め込む。
2. 生成処理
   生成処理を行なう
3. 生成結果の保存
   生成結果の素性構造を「素性構造書き換えシステム」で適当に変形し、これを現在の発話番号に対する生成独自のデータとして保存する。

この手順のリスプコードと生成結果から保存すべき素性構造を作成するための書き換え規則の例を以下に示す。

```
;-------------------------------------------------------
;  談話処理とのインタフェース(生成側)
;
;  (C) ATR Interpreting Telephony Research Labs.
;   Coded by G.Kikui
;-------------------------------------------------------
(in-package "USER")
;
;
(defvar *context_store_generation* (make-hash-table))
;
(defun gen_store_fs (fs id)     ;; 素性構造を保存する関数の定義
    (setf (gethash id *context_store_generation*) fs))

(defun gen_get_stored_fs (id)   ;; 保存去れている素性構造を
  (cond                         ;; 取り出す関数の定義
    ((gethash id *context_store_generation*))
    (t (rws::fs-create :leaf))))
;-----------------------------------------------------
;
; 生成処理のメイン関数
;
;-----------------------------------------------------
(defun TRE_GENERATION (id efs &aux rws_trees)
  ;;
  ;; 変換結果と文脈から得られる素性構造とのマージ
  (setq efs (dtm_gen_merge_context_data id efs))
  (when (member :generation_in *dtm_debug*)
    (format *msg-output* "~% ++++ GENERATION INPUT ++++~%")
    (rws::pprint-fs efs))
  ;;
  ;; 生成処理の実行
  (setq rws_trees
        (regulate_gen_out_structure
          (gen (rws-to-f_node efs))))
  ;;
  ;; 生成結果の素性構造を加工して保存
  (gen_store_fs (dtm_gen_context_data id (car rws_trees)) id )
  (list (mgen (car rws_trees))))
;
```

```
;  保存された素性構造の取りだし(生成入力の作成)
; (保存されている素性構造を変換結果の PRAG 素性の下の GEN_LC 素
;  性として付加する)
;
(defun dtm_gen_merge_context_data (id fs &aux prag)
  (dtm_add_fs
   (dtm_make_standard_local_context id #'gen_get_stored_fs :e)
   'GEN_LC
   (rws::fs-get-subfs fs '(PRAG)))
  fs)
;
;
;  保存すべき素性構造の作成(生成出力の加工)
;
(defun dtm_gen_context_data (id fs)
  (first
   (rws::transfer       ;; 素性構造書き換えシステムの呼び出し
    fs
    :control-rule-application nil
    :parameter-environment '((:PHASE . :GEN_CONTEXT)))))
```

```
;==================================================================
;
;   Contextual information for generation
;
;
;==================================================================
(rws::remove-rw-rule ':unspecified '(m))

(rws::defrwschema x x
   "on <M> :unspecified in :phase :gen_context
    in= [[M [[PRAG ?prag]
             [SEM ?sem]
     ?rest0]]
 [D1 ?d1]
 ?rest2]
    if  ?prag.GEN_LC then
        delete GEN_LC from ?prag
endif
    set ?context to  [[PRAG ?prag]
                      [SEM  ?sem]
                      ?rest0]
%  (setq *GEN_CONTEXT*
        ?context
%     )
;;  % (rws::pprint-fs
;;        ?context
;;  % )
    => ?D1 with :phase :gen_context_np
    out= ?context
    end"
)

(rws::remove-rw-rule 'S)

(rws::defrwschema x x
   "on <M SYN cat> S  in :phase :gen_context_np
    in= [[M [[SYN [[INV +]
                   ?msyn]]
             ?mrest]]
        [D1 [[M ?aux]]]
        [D2 [[M ?NP]
             ?d2rest]]
        ?rest]
    % (rws::fetch-global-fs ?context *GEN_CONTEXT*)
   add { [MAIN_SUBJ ?NP]
         [MAIN_V    ?aux] } to ?context
    out= ?input
    end"
)
```