TR-I-0188

SL-TRANSにおける変換過程の処理内容について

Details of the transfer process in the ST-TRANS translation system

長谷川敏郎
Toshiro HASEGAWA

1990年11月1日

概要

音声言語翻訳実験システムSL-TRANSにおける日本語依存意味構造から英語依存意味構造への変換過程での処理内容について述べる。本報告では、モデル会話(A, B, 1 – 5)の翻訳実験における変換プロセスの処理内容を、変換規則の例を挙げながら説明する。また、各処理における問題点も指摘してゆく。

ATR自動翻訳電話研究所
ATR Interpretin Telephony Reserach Laboratories

目次

1 はじめに

音声言語翻訳実験システムSL-TRANSにおける日本語依存意味構造から英語依存意味構造への変換過程での処理内容について述べる。本報告では、モデル会話A,Bおよびモデル会話1から5の翻訳実験における変換プロセスの処理内容を、変換規則の例を挙げながら説明する。また、各処理における問題点も指摘してゆく。

2 意味構造の表現形式

変換過程の入出力である意味構造の形式について簡単に述べる。意味構造は素性構造によって表現され、発話のタイプを表現する部分と命題内容を表現する部分から構成される。

発話のタイプを表現する部分は、意味構造のトップ（根）に位置し、relation素性、agent素性、recipient素性、object素性を取る。発話のタイプはrelation素性をもちいて表現される。agent素性には話し手が、recipient素性には聞き手を示す素性構造が入る。object素性には命題内容が記述される。また、発話のタイプに応じてmanner素性、attitude素性をとることがある。

```
[[reln  utterance-type]
 [agen [[label *speaker*]]]
 [recp [[label *hearer*]]]
 [obje  propositional-contentes]]]
```

命題内容を表す部分は基本的には格構造で表現される。命題内容は名詞類と述語類の二つの類型がある。名詞類は以下の表現形式で与えられる。

```
[[reln !x[]]
 [restr [[reln 会議-1]
         [entity !x]]]]
```

これは会議の意味を表す。述語類は、述語の核心部分を表すrelnとその値、およびそれに付随する意味格によって表される。例えば、「学生が会議に申し込む」の意味は次のように表される。意味表現の詳細については吉本[3], 久米[4]を参照のこと。

```
[[reln 申し込む-1]
 [agen [[parm !x[]]
        [restr [[reln 学生]
                [entity !x]]]]]
 [obje [[parm !y[]]
        [restr [[reln 会議]
                [entity !y]]]]]]
```

解析の出力であり変換の入力である素性構造は、relation素性の値に日本語の語彙[1]を用いて表現され、変換の出力である素性構造はrelation素性の値に英語の語彙

[1]意味構造の構成要素である概念を語彙あるいは関係名と呼ぶ

を用いて表現される。英語の語彙定義は小学館プログレッシッブ英和辞典[5]に基づいており、“語彙名-品詞-語彙番号”の形式をしている。語彙名はプログレッシブ英和辞典の見出しに対応しており、品詞は見出し内で定義されている品詞、語彙番号は品詞内で定義されている番号である。

変換の基本的な方式は、意図伝達方式[1]に基づいており、原則として、発話のタイプは変換対象とせず、そのまま生成処理に渡す。一方、日本語の語彙を用いて表現された命題内容は、英語の語彙を用いた表現に変換過程で書き換えられる。この意味構造の変換は、素性構造の書き換えシステム[6][?]を用いて実現されており、入力素性構造に書き換え規則を適用することによって、目的の構造である英語の意味構造(素性構造)を得る。

3 変換過程における処理の流れ

変換過程は五つのサブプロセスより構成され、すべての入力素性構造に対して、常に一定の順序で[2]処理される。変換過程におけるサブプロセスは、規則“:main”(規則1)によって生成される。(規則1)は、入力素性構造に対して書き換え環境を特定の状態に設定して書き換え呼び出しを行うことにより、サブプロセスが生成される。(規則1)[3]から生成されるサブプロセスは、更にいくつかのサブプロセスから構成されるものもある。今回の実験で設定したサブプロセスと各サブプロセスに含まれる規則数を表1に示す。以下、順をおって各サブプロセスの処理内容を説明する。

表1: 変換過程のサブプロセス

サブプロセス1	サブプロセス2	規則数
発話のタイプの決定	一般規則の適用	26
	デフォルト規則の適用	1
日本語内の書き換え	—	29
日英の変換	イディオム規則の適用	13
	一般規則の適用	188
	デフォルト規則の適用	35
省略補完	—	
tense, aspect の決定	英語辞書引き	
	tense, aspect の決定	

[2]「はい」、「もしもし」など固定的な表現に対してもすべてのサブプロセスにおける処理を行なっているが、このような訳語の選択を必要としない表現に関しては、別の簡略化したパスで処理した方が効率的であろう。

[3]書き換え規則は付録A「書き換え規則の例」参照

4 発話のタイプの決定

このサブプロセスでは、解析結果の意味構造から発話のタイプの決定を行う。このサブプロセスでは、日本語の文末表現に対応する意味(部分)構造、および、特定の副詞に対応する意味構造から、入力発話を以下に示す9種の発話のタイプのいずれかに分類する。また、入力発話が重文あるいは複文の時には命題内容は複数存在するが、これに対して、入力素性構造に一つの発話のタイプが与えられる。複数の命題内容に対して、発話のタイプを一つ与えるだけで十分であるか否かには、疑問が残る。

発話のタイプを、久米[2]の分類を基に、

- PHATIC
- REQUEST
- INFORM
- QUESTIONIF
- QUESTIONREF
- QUESTIONCONF
- RESPONSE
- EXPRESSIVE
- PROMISE

の9種類に分類した。

発話のタイプを決定する規則には、一般規則とデフォルト規則がある。まず、入力素性構造に対して一般規則の適用が試みられる。一般規則の適用によって発話のタイプのいずれかに分類できなかった発話に対しては、発話のタイプのデフォルトと仮定しているINFORMタイプがデフォルト規則(規則2)によって与えられる。

発話のタイプの決定を一般規則の適用とデフォルト規則の適用の二段回で処理しているのは、書き換えシステムの規則における適用制御方式と書き換え規則の記述方法の制限からである。ここで利用した書き換えシステムでは、ある時点で適用可能規則がn個ある場合、すべての規則の適用が成功するとn個の解を返す。また、ある時点での適用可能な規則はすべて同等に扱われ、規則間の優先順位等は与えられていない。入力素性構造に対して9種発話のタイプのいずれかに分類しなければならならず、一般規則で発話のタイプが与えられなかった時には、デフォルト規則で無条件に入力意味構造に発話のタイプとしてinformタイプを与えている。ここで、inform以外のタイプを与える一般規則とinformタイプを無条件に与えるデフォルト規則を同時に適用すると、入力にinformタイプ以外のタイプが与えられる時には、常にあるタイプとinformタイプの二つのタイプが与えられてしまう。したがって、段階的に規則を適用することにより規則間の競合をなくしている。

4.1 PHATIC

挨拶等で用いられるイディオム的な発話が、このタイプに含まれる。電話会話での挨拶に関する関係名(もしもし_OPEN_DIALOGUE、さようなら-CLOSE-DIALOGUE等)を参照し(規則3, 4参照)、入力素性構造に発話のタイプを表現する素性構造を付与するかたちで、書き換えがおこなわれる。

```
[[reln もしもし_OPEN_DIALOGE]    ⇒    [[reln PHATIC]
 [agen [[label *speaker*]]]            [agen !sp [[label *speaker*]]]
 [recp [[label *hearer*]]]]            [recp !hr [[label *hearer]]]
                                       [obje [[reln もしもし_OPEN_DIALOGE]
                                              [agen !sp]
                                              [recp !hr]]]]
```

図 1: 発話のタイプPHATICの付与

発話のタイプにPHATICを与えた発話を以下に列挙する。

- もしもし。(01)
- さようなら。(03)
- それでは失礼します。(01)
- では失礼します。(05)
- どうも失礼致します。(A)
- 失礼します。(04)
- 失礼致します。(02)

4.2 EXPRESSIVE

話者の感情表現に関するイディオム的な発話がこのタイプに含まれる。入力意味構造を参照し、PHATIC同様入力素性構造に発話のタイプを表現する構造を付与するかたちの書き換えが行なわれる(規則6)。

- ありがとう
- どういたしまして

4.3 PROMISE

約束を表す発話がこのタイプに含まれる。入力意味構造を参照し、入力素性構造(もらう-receive_favor, させる-promise)を置き換えるかたちで発話のタイプが与えられる(規則7)。

- 登録用紙を至急送らせて頂きます。
  させる-PERMISSIVE + もらう-RECEIVE_FAVOR ⟹ PROMISE

```
登録用紙を至急送らせて頂きます。⇒
[[reln もらう -RECEIVE_FAVOR]
 [agen !sp [[label *speaker*]]]
 [recp !hr [[label *hearer*]]]
 [obje [[reln させる -PERMISSIVE]
        [agen !hr]
        [recp !sp]
        [obje [[reln 送る -1]
               [agen !sp]
               [recp !hr]
               [obje ...]]]]]]
```

```
[[reln PROMISE]
 [agen !sp [[label *speaker*]]]
 [recp !hr [[label *hearer*]]]
 [obje [[reln 送る -1]
        [agen !sp]
        [recp !hr]
        [obje ...]]]]
```

図 2: 発話のタイプ PROMISE の付与

4.4 QUESTION

疑問文に対応する発話のタイプであり、さらに QUESTIONIF、QUESTIONREF、QUESTIONCONF の三種類に下位分類される。

1. **QUESTIONIF**

   yes-no 疑問文と解析された文がこのタイプに入る。解析結果では S-REQUEST と INFORMIF の二つの関係名で表現される。発話のタイプの分類処理では、この部分構造を発話のタイプ QUESTIONIF で書き換える。「〜うか」のように、推測を表す「う (う -GUESS)」と yes-no 疑問文を表す S-REQUEST および INFORMIF とが共起した場合は、関係名う -GUESS の部分構造は削除している。

   - そちらは会議事務局ですか？ (01)
     S-REQUEST + INFORMIF ⟹ QUESTIONIF(規則 8)
   - お名前をお伺いできますでしょうか？ (05)
     S-REQUEST + INFORMIF + う -GUESS ⟹ QUESTIONIF (規則 8) (規則 10)
   - では、誰かが私の代わりに参加することはできますか？ (05)
   - 会議の案内書はお持ちですか？ (04)
   - 参加料は要るのでしょうか。 (B)
   - 登録料を払い戻して頂けますか？ (05)
   - 発表が日本語で行われる場合英語への同時通訳はあるのですか？ (03)

2. **QUESTIONREF**

   WH 疑問文と解析された文がこのタイプに入る。解析結果では、S-REQUEST と INFORMREF との二つの関係名で表現され、この部分構造を発話のタイプを表す QUESTIONREF という一つの関係名に書き換える。questionif の場合と同様に、関係「う -GEUSS」は削除している。

   - いま会議に申し込めば、参加料はいくらですか？ (02)
     S-REQUEST + INFORMREF ⟹ QUESTIONREF(規則 5)

```
事務局ですか
[[RELN  S-REQUEST]
 [AGEN !X04[[LABEL  *SPEAKER*]]]
 [RECP !X03[[LABEL  *HEARER*]]]
  [OBJE [[RELN  INFORMIF]
         [AGEN !X03]
         [RECP !X04]
         [OBJE [[RELN  だ-IDENTICAL]
                [ASPT  STAT]
                [OBJE !X03]
                [IDEN [[PARM !X02[]]
                       [RESTR [[RELN  NAMED]
                               [ENTITY !X02]
                               [IDEN  会議事務局-1]]]]]]]]]]
⇒
[[RELN  QUESTIONIF]
 [AGEN !X04[[LABEL  *SPEAKER*]]]
 [RECP !X03[[LABEL  *HEARER*]]]
 [OBJE [[RELN  だ-IDENTICAL]
        [ASPT  STAT]
        [OBJE !X03]
        [IDEN [[PARM !X02[]]
               [RESTR [[RELN  NAMED]
                       [ENTITY !X02]
                       [IDEN  会議事務局-1]]]]]]]]
```

図 3: 発話のタイプ QUESTIONIF の付与

- どのようなご用件でしょうか？ (01)
  S-REQUEST + INFORMREF + う-GUESS ⟹ QUESTIONREF (規則 5) (規則 9)
- どうすればよろしいですか。(B)
- どのような手続をすればよろしいのでしょうか。(01)
- 参加料はどのようにお支払いしたらよいのですか？ (02)

3. QUESTIONCONF

確認を表す「〜ね」の文型がこのタイプに含まれる。関係名“ね-CONFIRMATION”を QUESTIONCONF に置き換える操作が行われる。

- 372 の 8018 でございますね。(04)
  ね-CONFIRMATION ⟹ QUESTIONCONF (規則 11)
- 既に登録料の 8 万 5 千円を振り込まれておられますね？ (05)

4.5 REQUEST

話者が(主に聴者に対して)何らかの要求を表すタイプである。

基本的には、“たい”、“下さい”、“願う”などの要求を表す表現が用いられた時、REQUEST タイプに分類している。REQUEST タイプの発話は manner 素性(値は direct あるいは indirect) と attitude 素性(値は declarative あるいは interrogative) を

とり、要求の間接性や文型を表現している。要求表現を和らげる働きのある語(文末表現として、“が(が-moderate)”、“思う”や副詞“失礼ですが”など)が用いられた時には、manner素性の値をindirectとして表現している。このとき、要求を表す部分意味構造(「下さい-desire」や「願う-resuest」など)は発話のタイプの表現で置き換えられる。さらに、要求表現を和らげる働きをする語に対応する部分意味構造は削除している。

```
失礼ですが、名前と住所をお願いします。
[[RELN 願う -REQUEST]
 [ASPT UNRL]
 [AGEN !X05[[LABEL  *SPEAKER*]]]
 [RECP !X03[[LABEL  *HEARER*]]]
 [OBJE [[RELN  と -COORDINATE]
        [ARG-1 !X04[[PARM !X02[]]
                    [RESTR [[RELN  名前 -1]
                            [ENTITY !X02]]]]]
        [ARG-2 !X08[[PARM !X07[]]
                    [RESTR [[RELN  住所 -1]
                            [ENTITY !X07]]]]]]
 [INFATTD [[PARM !X09[]]
           [RESTR [[RELN  失礼ですが -1]
                   [ENTITY !X09]]]]]]
```

⇒

```
[[RELN REQUEST]
 [ASPT UNRL]
 [MANNER INDIRECT]
 [ATTD DECLARATIVE]
 [AGEN !X05[[LABEL  *SPEAKER*]]]
 [RECP !X03[[LABEL  *HEARER*]]]
 [OBJE [[RELN  と -COORDINATE]
        [ARG-1 !X04[[PARM !X02[]]
                    [RESTR [[RELN  名前 -1]
                            [ENTITY !X02]]]]]
        [ARG-2 !X08[[PARM !X07[]]
                    [RESTR [[RELN  住所 -1]
                            [ENTITY !X07]]]]]]]
```

図4: 発話のタイプREQUESTの付与

- 案内書に記載されている口座番号に振り込んで下さい。(02)
  下さい-REQUEST ⟹ REQUEST(manner:direct) (規則12)
- ご住所とお名前をお願いします。(01)
  願う-REQUEST ⟹ REQUEST(manner:direct) (規則13)
- 失礼ですがお名前とご住所をお願い致します。(04)
  REQUEST + 失礼ですが ⟹ REQUEST(manner:indirect) (規則13) (規則14)
- よろしくお願いします。(01)

「〜たい」などの願望表現は、原則としてINFORMタイプに分類され、INFORMとWANTを用いて表現されるが、「頂く」などの受給表現と共起したときには聞き手への働きかけが認められるのでREQUESTタイプに分類している。

- 会議の参加料について教えて頂きたいのですが。(02)
  てもらう -RECEIVE_FAVOR + たい -DESIRE + が -MODERATE ⟹
  REQUEST(manner:indirect) (規則16)(規則19)(規則18)

原則として、疑問文はquestionタイプの発話のタイプが与えられるが、可能性を尋ねている時には、requestタイプに分類した。疑問文にrequestタイプを与えた時には、attitude素性の値をinterrogativeにして、発話が疑問文であった平叙文であったことを表す。また、yes-no疑問文にrequestタイプを与える時には、単にs-request + informif + できる-possibleの三つの関係名の共起だけでなく、命題内容(ここでは「伺う」)の行為者が話者であることを制約として課している。

cf. では、誰かが私の代わりに参加することはできますか？

- お名前をお伺いできますでしょうか？
  できる-POSSIIBLE + QUESTIONIF ⟹ REQUEST(規則17) [4] [5]

```
お名前をお伺いできますでしょうか                    ⇒    [[RELN  REQUEST]
[[RELN  S-REQUEST]                                        [AGEN !sp[[LABEL  *SPEAKER*]]]
 [AGEN !sp[[LABEL  *SPEAKER*]]]                           [RECP !hr[[LABEL  *HEARER*]]]
 [RECP !hr[[LABEL  *HEARER*]]]                            [MANNER INDIRECT]
 [OBJE [[RELN  INFORMIF]                                  [OBJE [[RELN  聞く -1]
        [AGEN !hr]                                              [AGEN !sp]
        [RECP !sp]                                              [RECP !hr[]]
        [OBJE [[RELN  う -GUESS]                                [OBJE ...]]]]
               [ASPT  STAT]
               [EXPR !sp]
               [OBJE [[RELN  できる -POSSIBLE]
                      [ASPT  UNRL]
                      [EXPR !sp[]]
                      [OBJE [[RELN  聞く -1]
                             [AGEN !sp]
                             [RECP !hr[]]
                             [OBJE ...]]]]]]]]]]
```

図5: 発話のタイプREQUESTの付与

[4]発話のタイプの決定をおこなうサブプロセスでは、素性構造の根から葉に向かってトップダウンに書き換えをすすめゆく。したがって、S-REQUEST, INFORMIFで、一度QUESTIONIFに書き換えられ(発話のタイプが決定され)、その後、「できる-possible」にであった時点で、REQUESTタイプに更に書き換えられる。

[5]実験では、省略されている「できる-possible」のexprience格、あるいは「聞く-1」のagent格の補完できないので、発話のタイプにquestionifが与えられてる。

4.6 RESPONSE

質問などに対する応答や合いづちなどの発話がこのタイプに分類される。

- いいえ(01)
- はい(01)
- そうですか(04)

4.7 INFORM

デフォルトの発話のタイプであり、上記のどのタイプとも決定できなかった時に、informタイプに分類している。「〜たい」のような話者の願望を表す発話は、話者の間接的な要求である場合も多いが、ここではinformタイプに分類している。「たい」などの願望の表現は関係名WANTを用いて表現している。

- 会議に申し込みたいのですが。(01)
  たい-DESIRE + が-moderate ⟹ INFORM(manner:indirect) + WANT (規則19)(規則18)

```
... したいのですが                ⇒      [[reln INFORM]
[[reln が-MODERATE]                       [manner indirect]
 [agen !sp [[label *speaker]]]            [agen !sp[[label *speaker*]]]
 [recp !hr [[label *hearer]]]             [recp !hr[[label *hearer*]]]
 [obje [[reln たい-DESIRE]                [obje [[reln WANT]
        [agen !sp]                               [agen !sp]
        [obje ..]]]]                             [recp !hr]
                                                 [obje ...]]]]
```

図6: 発話のタイプINFORMの付与(願望表現)

接続副詞(「それでは」、「ところで」等、CONECT素性に入る)、陳述副詞(「どうも」、「よろしく」等、INFMANN素性に入る)、発言副詞(「実は」等、INFATTD素性に入る)は、発話のタイプが記述されるレベルに引き上げている(トップレベルに移動させる)。

以下に、発話のタイプを決定するために参照した関係名を列挙する。"*"は発話のタイプを決定する際に参照されるだけで意味表現中に現れる。無印は発話のタイプの表現に吸収され意味表現中には現れない。

s-request	informif	informref
ありがとう-thanking*	いいえ-negative*	う-guess
思う-1(manner:indirect)	どういたしまして-your-welcome*	が-moderate
下さい-request	させる-permissive	さようなら-close-dialogue*
失礼する*	失礼ですが-1	そうですか-confirmation*
できる-possible	たい-desire(want)	てもらう-receive_favor
ね-confirmation	願う-request	はい-affirmative*
もしもし-open-dialogue*		

```
それでは登録用紙を送ります      ⇒      [[RELN INFORM]
[[RELN  送る-1]                          [AGEN !SP [[LABEL  *SPEAKER*]]]
 [ASPT  UNRL]                            [RECP !HR [[LABEL  *HEARER*]]]
 [AGEN !X03[[LABEL  *SPEAKER*]]]         [OBJE [[RELN  送る-1]
 [RECP !X02[[LABEL  *HEARER*]]]                 [ASPT  UNRL]
 [OBJE [[PARM !X06□]                            [AGEN !SP]
        [RESTR [[RELN  登録用紙-1]              [RECP !HR]
                [ENTITY !X06]]]]]               [OBJE [[PARM !X06□]
 [CONECT [[PARM !X05[]]                                [RESTR [[RELN  登録用紙-1]
         [RESTR [[RELN  それでは-1]                           [ENTITY !X06]]]]]]]
                [ENTITY !X05]]]]]]       [CONECT [[PARM !X05[]]
                                                 [RESTR [[RELN  それでは-1]
                                                        [ENTITY !X05]]]]]
```

図 7: 接続副詞の引き上げ

5 日本語内の書き換え

このサブプロセスでは、日本語依存意味構造を日本語依存意味構造へ変換する。このサブプロセスは、日本語内で構造の縮退や正規化(標準化)を行い、日英の変換規則数を低減し、日英規則を簡略化し、複雑な書き換えをさける目的で設定した。

5.1 サ変名詞の処理

「サ変名詞＋を＋する」の意味表現は、「する」の対象格にサ変名詞をとる形で表現される。

- 会議に参加をする

```
[[reln する-1]
 [agen ...]
 [obje [[parm !x[]]
        [restr [[reln 登録-1]
                [entity !x]]]]
 [sloc [[parm !y[]]
        [restr [[reln 会議-1]
                [entity !y]]]]]
```

図 8: サ変名詞の意味表現

「サ変名詞＋を＋する」を解析過程で動詞として表現することも可能であろうが、サ変名詞に対する連体修飾句の統一的な扱いが困難であることから、このような表現形式が取られている。

「サ変名詞＋を＋する」の日英の変換過程では、言語間での対応する概念のマッピング処理と、サ変名詞を修飾する連体修飾句の連用修飾句への変換処理という二つの性格の異なる処理が必要である。そこで、まず日本語内でサ変名

```
登録用紙で手続きをする
[[reln する -1]
 [obje [[parm !x[]]
        [restr [[reln 手続き -1]
                [entity !x]]]]]
 [instr [[parm !y[]]
         [restr [[reln 登録用紙 -1]
                 [entity !y]]]]]]
```

⇒

```
登録用紙で手続きする
   [[reln 手続きする -1]
    [instr [[parm !y[]]
            [restr [[reln 登録用紙 -1]
                    [entity !y]]]]]]
```

図 9: サ変名詞の処理

詞句を動詞句に変換した後、英語の意味構造に変換する。この処理は関係名「する」をトリガとして、「する」の対象格である名詞句を動詞句に書き換える書き換え呼び出し[6]にって動詞句化を行い、書き換え呼びだしが正常終了すれば、その結果をパラフレーズの結果として置き換える(規則33)。この処理のトリガとなる関係名には、「する」以外に「できる」「行なう」「希望する」等がある。

- 取消に対する払い戻しはできません (会話 05)(規則 28)(規則 26)(規則 33)
- 割引を行っておりません (会話 02)(規則 30)(規則 34)
- 発表を希望されるのですたら (会話 04)(規則 31)(規則 35)

5.2 表現の縮退

日本語の表現そのまま訳出すると英語側での表現が冗長になる時、表現の縮退を行なっている。表現の縮退を行う明確な基準は設定しにくいが、実験で表現の縮退を必要とした対象は、以下の二つに分類される。

- 説明的表現の縮退
  一つの名詞(単純概念)で表現できるものを、説明するかたちで言い表したものの縮退
- 内容節の縮退 内容節と(形式)名詞で表現されたものの縮退

5.2.1 説明的表現の縮退

一つの名詞(単純概念)の内容を説明的に表現する方法は多様であるから、名詞の意味構造に内包的な記述を与えない限り一般的に処理するのは困難である。ここでは、各々の記述的な表層表現に対して一つづつ書き換え規則を定義して対処している。

[6]名詞を動詞に派生させる規則、“の”で連結された連体修飾句を動詞化されたものの格に埋め込む規則、形容詞を副詞に派生させる規則などが適用される。

- 分からない点があれば…
  分かる-1 + NEGATE + 点-1 ⟹ 質問-1 (規則23)

```
[[PARM !X2[[PARM !X1[]]      ⇒        [[parm !x[]]
                   [RESTR [[RELN 点 -1][restr [[reln 質問 -1]
                            [ENTITY !X1[]entity !x]]]]
          [RESTR [[RELN NEGATE]
                 [OBJE [[RELN 分かる -1]
                        [EXPR []]
                        [OBJE !X2]]]]]]
```

5.2.2 内容節の縮退

「もの」「こと」などの形式名詞が内容節を取る場合、内容節をそのまま動詞句として処理すべきものと、内容節を動詞句から名詞句へ派生させるべきものとがある。

- では誰かが私の代わりに参加することはできますか
  参加する-1 + 外の関係の連体修飾 + こと-1 ⟹ 参加する (規則26)(規則21)
- 詳しいことを教えて下さい
  詳しい-1 + 外の関係の連体修飾 + こと-1 ⟹ 詳細 (規則22)

内容節＋形式名詞の表現を名詞句として表現するか動詞句として表現するかは、日本語の動詞だけでは判断できない。つまり、形式名詞を補語にとる日本語の動詞に対応する英語の動詞が、補語に(不定詞、動名詞も含めて)動詞句を取るか名詞句を取るかという訳語の統語的な制約によって、形式名詞を修飾する内容節の扱いが決まる。

例えば、「できる」の場合には、“possible” を対訳として選択すると、形式名詞を “that” に対応させ、構造的に日本語に似た形で生成処理に渡すことができる。一方、「できる」に助動詞類 “can” あるいは “be able to” を対訳として与えると、“can” は形式名詞を直接補語に取らないので、内容節を取り出し(形式名詞を意味構造から削除し)“can” の補語とする必要がある。

ここでは、より統一的な意味表現形式を用いるという目的で、「できる」(できる-possible) を “can”、“be able to”、“possible” などを包含した抽象概念 “possibility” で表し、possibility の対象としては、形式名詞を除いたより単純な形式に日本語内でパラフレーズしている。

```
参加することができる
[[reln できる -possible]
 [agen !sp [[label *speaker*]]]
 [recp !hr [[label *hearer*]]]
 [obje [[parm !x1[[parm !x2[]]
                 [restr [[reln こと -1]
                         [entity !x2]]]]]
        [restr [[reln 外の関係の連体修飾]
                [arg-1 !x1]
                [arg-2 [[reln 参加する -1]
                        [agen []]
                        [sloc []]]]]]]]
```

⇒

```
[[reln できる -possible]     Rightarrow      [[reln possibility]
 [agen !sp [[label *speaker*]]]]              [agen !sp [[label *speaker*]]]
 [recp !hr [[label *hearer*]]]                [recp !hr [[label *hearer*]]]
 [obje [[reln 参加する -1]                    [obje [[reln 参加する -1]
        [agen []]                                    [agen []]
        [sloc []]]]]                                 [sloc []]]]]
```

一方、「教える」の場合は名詞的概念をとり、かつ、「詳しいこと」をより単純な記述で表現する目的で、内容節を名詞に派生させる。名詞句化する(規則23)は形式名詞をキーとして名詞句全体を一度に名詞に変換する。

また、形式名詞以外、例えば、「予定」が内容節を取り、ダ-文として現れた時には、サ変名詞「予定」を動詞に派生させ、内容節を動詞の対象格に埋め込む、パラフレーズを行なった。

```
参加する予定だ                                    ⇒    参加を予定する(規則20)
[[reln だ -identical]                                  [[reln 予定する]
 [obje []]                                              [agen []]
 [iden [[parm !x7[[parm !x1[]]                          [obje [[reln 参加する -1]
                 [restr [[reln 予定 -1]                        [agen []]
                        [entity !x1]]]]]                       [obje []]]]]
       [restr [[reln 外の関係の連体修飾]
              [arg-1 !x7]
              [arg-2 [[reln 参加する -1]
                     [agen []]
                     [obje []]]]]]]]]
```

5.3 構文の変換

多くの規則には単語(概念)対単語(概念)の局所的な対応関係が記述される。しかし、日本語の部分構造が直接英語で対応するものがない場合には、構文の変換が必要になるものがある。

- お気の毒ですが、出来ません。(会話05) ⟹ 出来ないのは気の毒です (規則24) (規則25)
  I am sorry that it is not possible.

この例では、主節と従属節とを入れ換えるパラフレーズ(正確には、「お気の毒ですが」で一つの副詞として表現されているので、まず副詞句を動詞句に変換し、その後、従属節と主節を入れ換える処理)を行なっている。

このような構文の変換を行った場合、変換前と変換後で発話のタイプを保つことが困難になることが予想される。また、副詞句から動詞句への変換では、新たに空の必須格が生まれる。変換過程でこのような複雑な操作を避けるために、副詞「お気の毒です」をより抽象度の高い概念として表現し、意味表現の構造を保ったまま副詞句として生成に渡し、生成過程で処理する方法も考えられる。としても、上記のような訳を生成するためには、同様の操作を生成過程で必要とする。

```
お気の毒ですができません          ⇒        [[reln 気の毒だ-1]
[[RELN  NEGATE]                              [expr []]
 [ASPT  STAT]                                [obje [[reln NAGATE]
 [OBJE [[RELN  できる-POSSIBLE]                      [obje [[reln できる-POSSIBLE]
        [EXPR []]                                          [expr []]
        [OBJE []]]]                                        [obje []]]]]]]
 [INFATTD [[PARM !X04[]]
           [RESTR [[RELN  お気の毒ですが-1]
                   [ENTITY !X04]]]]]]
```

5.4 変換過程における省略補完の必要性

格パタンを変更するパラフレーズでは、空の必須格が生じることがある。

- 参加する予定だ ⟹ 参加を予定する (規則 20)

このようなパラフーレズを行った時には、述語「予定する」の agent 格を埋める必要性が生じる。現在は変換過程で生じた空の必須格を埋める処理は行っていない。

6 日英の変換

日本語に依存した(日本語の関係名で記述された)意味構造から英語に依存した意味構造への変換を行なう。日英変換サブプロセスはさらに、イディオム規則の適用、一般規則の適用、デフォルト規則の適用の三つのサブプロセスから構成され、この順にサブプロセスが生成される。このサブプロセスの分類は、各サブプロセスに属する規則が対応づける概念間の対応関係の側面から見ると、

- イディオム規則は対応関係が特殊であり適用制約が強く概念間の関係が二言語間で遠いもの
- 一般規則は二言語間で概念間にほぼ対応が取れ、適用制約が意味素性[7]などで記述可能なもの
- デフォルト規則は規則適用の条件が非常に緩く、ほとんど無条件に日英の概念間の対応を取るもの

と分類することができる。変換プロセスをいくつかのサブプロセスに分割する理由は、規則の競合を少なくし計算量を軽減すること、かつ、段階的に規則を適用することにより規則の増加をおさえ、最適な解を容易に得るためである。現在の書き換えシステムのインプリメントでは、ある部分素性構造に適用可能な規則が複数存在する場合、各々の規則に対応して複数個の解が得られる。そこで、規則を idiom, general, default に分類し、規則の適用に順序を与えることで、規則間に優先順位を持たせている。これにより、生成される解の数を減らし、計算量の低減をはかってい

[7]現在は変換過程では意味素性やシソーラスなどを用いておらず、共起する格要素の関係名を直接記述している。

る。また、日本語内での書き換え、日英の書き換えというように、段階的に変換してゆくことで、入力意味構造の正規化をおこない、規則数の増加をおさえている。

6.1 英語の概念設定

多くの英語の概念(関係名)設定基準は、日本語に概念設定に依存している。日本語から英語への対応関係として処理しているので、英語の概念設定が日本語の概念設定に依存している。日本語では一語であるが英語にすると複数の単語から構成されるものについては、一つの関係名として表現したものがある。原則としては、日本語の概念を英語に対応させた時に、英語側で動詞を含む複合概念になる時には、それを動詞と格要素といった複数の関係名で表現している。一方動詞概念を含まない時には、単一概念として一つの関係名で表現している。

日本語で単一概念として表現され、英語で複数の単語から構成されるが単一概念として表現しているものを以下に示す。

今年-1	→	this_year-IDIOM-1
予稿集代	→	proceedings_fee-IDIOM-1
銀行振り込み-1	→	bank_transfer-IDIOM-1
電話番号-1	→	phone_number-IDIOM-1
研究分野-1	→	research_field-IDIOM-1
あらかじめ-1	→	in_advance-IDIOM-1
ところで-1	→	by_the_way-IDIOM-1
どのような-1	→	what_kind-IDIOM-1
さようなら-Close_Dialogue	→	good_bye-INTERJ-1
どういたしまして-GREETING	→	you_are_welcome-IDIOM-1 (イディオム規則)
まだ-1 + だ-statement	→	not_yet-IDOM-1 (イディオム規則)
そう-1 + だ-statement	→	that_is_right-IDIOM-1 (イディオム規則)

日本語で単一概念として表現され、英語で複数概念として表現されるものを以下に示す。

決まる-1	→	PASSIVE + decide-VT-1(規則47)
詳しい-1	→	know-VT-1 + well-ADV-4
割引する-1	→	make + discount (規則44)
(存在を表す)ない	→	NEGATE + be-VI-1 (規則45)

6.2 イディオム規則の適用

イディオム規則は最も優先順位が高く、他の規則に先だって適用が試みられる。イディオム規則は、処理の対象として意味構造の部分的な対応づけが可能なものと、部分的な対応づけが困難であり(意味がなく)文全体で対応づける必要があるものとに分類される。

1. 部分的な対応づけが可能なもの

   ここに分類される規則には、受給表現等を処理する規則がある。受給表現などは一つの関係名で表現され動詞を引数(格要素)に取る。受給表現を“やる”、“くれる”、“もらう”といった日本語表層表現に対応した一つの関係名で表現するのではなく、視点の違いなどの素性の組合せで表現することも可能だろうから、これらを変換処理で処理せず、生成過程で処理することも可能である。

   - 参加料はどのようにお支払したらよいのですか?
     たら-conditional + よい-1 ⟹ possibility(規則 42)
   - 先ず、登録用紙で手続きしていただかなくてはなりませんが
     てもらう-RECEIVE_FAVOR + なくてはいけない-MUST ⟹ obligation(規則 43)

2. 文全体を一つの変換対象としたもの

   - そうですか⟹All right
     そうですか-CONFIRMATION⟹all_right-IDIOM (規則 36)
   - どうもありがとう ⟹ Thank you very much
     どうも-1 + ありがとう-THANKING ⟹ thank-VT-1 + much-ADV-1 + very-ADV-1 (規則 37)
   - そうです ⟹ That is right
     そう-1 + だ-STATEMAENT ⟹ that_is_right-IDIOM-1 (規則 38)
   - どういたしまして ⟹ You are wellcome
     どういたしまして-GREETING ⟹ you_are_wellcome-IDIOM-1 (規則 39)
   - どのような用件ですか ⟹ May I help You?
     QUESTIONREF + どのような-1 + 用件-1 + だ-IDENTICAL ⟹ QUESTIONIF + permission + HELP-VT-1 (規則 40)
   - それではよろしくお願いします ⟹ Thank you very much
     REQUEST + よろしく -1 ⟹ EXPRESSIVE + thank-VT-1 + much-ADV-1 + very-ADV-1 (規則 41)

   変換対象が会話特有の固定表現である場合、その意味構造を構成する個々の概念を部分的に変換し英語の固定表現に対応する意味構造を得ることは容易ではない。また、それらの固定表現は数も限られているので、発話全体を一つの変換対象として日英の対応づけを行っている。

   ここでは、発話全体の日英での対応づけを行おうとしているのであり、命題内容の対応づけを行う処理ではなくなっている。したがって、もはや変換対象として発話のタイプや命題内容といった区別はなく、意味構造を構成する概念の対応関係は全く無視して、意味構造全体を一度に対応づけている規則もある。

   - どのような要件ですか(発話のタイプ:QUESTIONREF) ⟹ May I help you? (発話のタイプ:QUESTIONIF)

6.3 一般規則の適用

日英の部分的な対応関係をとる変換過程における主たるサブプロセスである。一般に日英の概念間の対応関係は多様であり、日本語および英語の概念の設定基準に依存する。今回の実験における概念設定レベルのように、比較的表層に近い概念設定レベルでは、日英の概念間の対応づけ(訳し分け)は複雑な問題になる。しかし、今回の実験では日本語での異なり語数は429(うち名詞106、動詞24、副詞32、その他)であり、翻訳の実験として非常に小規模であり、対話領域も限られていた。このため、多様な訳し分けをおこなう必要性は少なく、このサブプロセスで特に訳し分けが必要になったのは、わずか一つだけであった。

- 参加料には歓迎会費が含まれています $\Longrightarrow$ include
- 今回の会議には広範囲な研究分野が含まれています $\Longrightarrow$ cover

動詞の訳し分けは一般には動詞と格要素の共起関係で処理することができる。共起関係の表現の効率性を考えると、名詞の意味素性(semantic feature)あるはシソーラスなど意味的分類に関する情報は利用は一般的であるが、今回の実験では適当な意味素性のセットやシソーラスがなかったため、意味的分類に関する情報は用いなかった。今後、訳語選択に意味的情報の利用が不可欠と考えられる。

また、ある時点(サブプロセス)における適用可能な規則の適用および規則の適用結果はすべて同等に扱われており、複数解が得られた時に適切な解を選択すべき枠組が存在しない。規則の優先順位の制御あるいは規則を適用した結果にプリファレンスを与える手法など、訳し分けにおいて何らかの方法で解に優劣をつける手法が要求される。

6.4 訳語選択におけるヒューリスティック

空の必須格を増やさないような、訳の選択を行っている。例えば、“会議で発表する”は二つの対訳が考えられる。

$$\text{会議で発表する} \Longrightarrow \begin{cases} \text{make presentation} \\ \text{present} \end{cases}$$

この場合“発表する”の対象格が省略されている時には“make presentation”を、対象格が埋まっている時には“present”を選択することで、英語の表層表現において空の必須格が少なくなるように変換している。[8]

[8]このような訳語選択処理は表層表現とは離れたより抽象的なレベルで概念の設定をおこない、上記の二つの英語の表層表現に対しても同一の意味表現をとることができるなら、生成過程における訳語選択処理と考えることができる。つまり、生成過程において不必要に空の表層格をつくらないといった語彙選択基準をもうけて表層の単語選択処理で同様の処理ができる。したがって、変換、生成過程における処理分担は、意味表現(特に概念の設定基準)に依存している。

6.5 助動詞の変換

モーダルに関する表現は、比較的表層に近い日本語の概念をより抽象的な概念に変換し、そこから英語の表層の選択は生成過程でおこなっている。この対応関係を表2に示す。抽象概念の定義は[7]にもとづいており、あくまで英語側での分類であり、日英の対応関係を考慮したものではない。モーダルに関する表現を日英で一貫して抽象的概念で表現できるならば、発話のタイプの扱いと同様に変換過程の前処理あるいは解析段階で処理し、変換過程では原則として変換対象としないという態度がとれる。

表2: 助動詞の対応関係

日本語での関係名	抽象概念	英語の表層
たらよい-should ばよい-should なくてはならない-must	OBLIGATION	should
させる-permissive	PERMISSION	may
もらえる-possible える-possible できる-possible	POSSIBILITY	can

6.6 デフォルト規則の適用

デフォルト規則は日本語の概念が変換過程でそのまま残らないよう、何らかの英語概念を日本語概念にほぼ無条件に与えるものである。デフォルト規則は、日英変換プロセスでイディオム規則および一般規則が適用されなかった部分意味構造(語彙)に対して適用される。

7 テンス、アスペクトの決定

英語のテンスとアスペクトを変換の最後のサブプロセスで決定している。テンスおよびアスペクトは、解析過程で決定された状況アスペクト（命題に対して決定される）と動詞の六種類の分類の組合せによって決定される。状況アスペクトは、stative, unrealized event, past event, resulted state, progression, iteration の6種類に分類されている。アスペクトに関して動詞は dynamic, achivement, stative, momentaneous, activity, accomplish の六種類に分類され、これらは、表3に示すアスペクト素性で表現される。

表 3: 語彙の性質を表現するアスペクト素性

素性	素性名	説明
momentaneous	mome	単純時制で瞬間副詞 just, at once, at8:30, etc. と共起するか
process	proc	進行形になるか except— I was being innocent.
duration	durt	単純時制で持続副詞 since, till, untill, for six hours, while CLAUSE, etc. と共起するか

状況アスペクトと動詞の分類から決定される表層のアスペクトとテンスの関係を下表に示す。

dynamic type		
STAT	[mome +][porc +][durt +]	progressive
UNRL	[mome +][porc +][durt +]	"will" future
PAST	[mome +][porc +][durt +]	past progressive
RSLT	[mome +][porc +][durt +]	past or (past) perfect
PROG	[mome +][porc +][durt +]	progressive
ITER	[mome +][porc +][durt +]	present

achivement type		
STAT	[mome +][proc +][durt −]	past or perfect
UNRL	[mome +][proc +][durt −]	"will" fututre
PAST	[mome +][proc +][durt −]	past or past perfect
RSLT	[mome +][proc +][durt −]	past or (past) perfect
PROG	[mome +][proc +][durt −]	progressive
ITER	[mome +][proc +][durt −]	progrressive

stative type		
STAT	[mome +][proc −][durt +]	present
UNRL	[mome +][proc −][durt +]	"will" future
PAST	[mome +][proc −][durt +]	past
RSLT	[mome +][proc −][durt +]	present
PROG	[mome +][proc −][durt +]	—
ITER	[mome +][proc −][durt +]	—

momentaneous type		
STAT	[mome +][proc −][durt −]	—
UNRL	[mome +][proc −][durt −]	"will" future
PAST	[mome +][proc −][durt −]	past or (past) perfect
RSLT	[mome +][proc −][durt −]	past or (past) perfect
PROG	[mome +][proc −][durt −]	—
ITER	[mome +][proc −][durt −]	progressive

activity type		
STAT	[mome −][proc +][durt +]	progressive
UNRL	[mome −][proc +][durt +]	"will" future
PAST	[mome −][proc +][durt +]	past or (past) perfect
RSLT	[mome −][proc +][durt +]	past or (past) perfect
PROG	[mome −][proc +][durt +]	progressive
ITER	[mome −][proc +][durt +]	perfect

accomplish type		
STAT	[mome −][proc +][durt −]	—
UNRL	[mome −][proc +][durt −]	"will" future
PAST	[mome −][proc +][durt −]	(past) perfect
RSLT	[mome −][proc +][durt −]	(past) perfect
PROG	[mome −][proc +][durt −]	—
ITER	[mome −][proc +][durt −]	(past) perfect

状況アスペクトから英語の表層のテンス、アスペクトの決定は、英語の単語が決まった時点で、動詞のアスペクト素性を書き換え規則によって付加し、その後、状況アスペクトと動詞のアスペクト素性の組合せから書き換え規則によって表層のテンスとアスペクトを決定する。

入力発話: それは登録用紙をお送り致します

```
[[RELN  送る -1]
 [ASPT  UNRL]
 [AGEN !X03[[LABEL  *SPEAKER*]]]
 [RECP !X02[[LABEL  *HEARER*]]]
 [OBJE [[PARM !X06[]]
        [RESTR [[RELN  登録用紙 -1]
                [ENTITY !X06]]]]]
 [CONECT [[PARM !X05[]]
          [RESTR [[RELN  それでは -1]
                  [ENTITY !X05]]]]]]
```

⇓英語の意味構造の生成

```
[[RELN  send-1]
 [ASPT  UNRL]
 [AGEN !X03[[LABEL  *SPEAKER*]]]
 [RECP !X02[[LABEL  *HEARER*]]]
 [OBJE [[PARM !X06[]]
        [RESTR [[RELN  registration_form-IDIOM-1]
                [ENTITY !X06]]]]]
 [CONECT [[PARM !X05[]]
          [RESTR [[RELN  then-ADV-4]
                  [ENTITY !X05]]]]]]
```

⇓語彙のアスペクト素性の付加(規則48)

```
[[RELN  send-1]
 [ASPT  UNRL]
 [lex [[aspect [[MOME +]
                [PROC -]
                [DURT -]]]]]
 [AGEN !X03[[LABEL  *SPEAKER*]]]
 [RECP !X02[[LABEL  *HEARER*]]]
 [OBJE [[PARM !X06[]]
        [RESTR [[RELN  registration_form-IDIOM-1]
                [ENTITY !X06]]]]]
 [CONECT [[PARM !X05[]]
          [RESTR [[RELN  then-ADV-4]
                  [ENTITY !X05]]]]]]
```

⇓英語の表層のテンス、アスペクトの決定(規則49)
(規則50)

```
[[RELN  send-1]
 [ASPT  UNRL]
 [tense future]
 [aspect [[PERF -]
          [PROG -]]]
 [lex [[aspect [[MOME +]
                [PROC -]
                [DURT -]]]]]
 [AGEN !X03[[LABEL  *SPEAKER*]]]
 [RECP !X02[[LABEL  *HEARER*]]]
 [OBJE [[PARM !X06[]]
        [RESTR [[RELN  registration_form-IDIOM-1]
                [ENTITY !X06]]]]]
 [CONECT [[PARM !X05[]]
          [RESTR [[RELN  then-ADV-4]
                  [ENTITY !X05]]]]]]
```

今回の実験では状況アスペクトから表層のテンスとアスペクとの決定は変換過程で処理した。しかし、表層のテンスおよびアスペクトは、発話のタイプやモダリティに関する表現が、表層でどのように表現されるかに依存しているので、生成過程で処理したほうがすっきりしたものになるだろう。

入力発話:用紙を送ってください
(最終的な変換結果)

```
[[RELN REQUEST]
 [ASPT UNRL]
 [AGEN !X2[[LABEL *SPEAKER*]]]
 [RECP !X1[[LABEL *HEARER*]]]
 [OBJE [[RELN SEND-VT-1]
        [ASPECT [[PERF -]
                 [PROG -]]]
        [TENSE FUTURE]
        [LEX [[ASPECT [[MOME +]
                       [PROC -]
                       [DURT -]]]]]
        [AGEN !X1]
        [RECP !X2]
        [OBJE [[PARM !X3[]]
               [RESTR [[RELN FORM-N-14]
                       [ENTITY !X3]]]]]]
```

生成結果:Please send me the form.

意味構造では発話のタイプやモダリティは抽象的な関係名を用いて表現されており、これらを表層をどのように表現するかは生成過程で決定される。上記の例では、変換過程で時性としてfutureを与えているが、要求表現として“Please ...”を生成過程で選択したため、動詞の時性は現在形として生成されている。このように、変換過程でデフォルトとしての時性と相を与えることはできるが、決定をおこなうのはあくまで生成過程であるから、状況アスペクトから表層の形式の決定まですべてを生成過程で処理したほうが簡潔で好ましい。

参考文献

[1] Hitoshi Iida, Kiyoshi Kogure, and Teruaki Aizawa. An experimental spoken natural dialogue translation system using lexicon-driven grammar. In *EUROSPEECH*, 1989.

[2] Masako Kume, Gayle K. Sato, and Key Yoshimoto. A descriptive framework for translating speaker's meaning towards a dialogue translation sysyerm between japanese and english. In *European Chapter of ACL89*, 1989.

[3] 吉本啓 and 小暮潔. 句構造文法にもとづく日本語文の解析. Technical Report TR-I-0049, ATR Interpreting Telephony Research Labs., 1988.

[4] 久米雅子, 豊島, and 永田昌明. 話し言葉翻訳のための日本語アスペクト処理. In 情報処理学会第*40*回全国大会, 1990.

[5] 小西友七, 安井稔, and 國廣徹夫, editors. 小学館プログレッシブ英和中辞典. 小学館, 1980.

[6] 長谷川敏郎. 素性構造書き換えシステムマニュアル. Technical Report TR-I-0093, ATR Interpreting Telephony Research Labs., 1989.

[7] ホーンビー英語の型と語法. オックスフォード大学出版局,, 1981.

A 書換え規則の例

規則1
```
on <> :main
   in= [[sem ?sem]
        ?rest]
   set ?SP to input.prag.speaker
   set ?HR to input.prag.hearer
   input = [[sem [[reln UNKNOWN-IFT]
                  [agen ?sp]
                  [recp ?hr]
                  [obje ?sem]]]
            ?rest]
   rewrite input.sem with :IF :REDUCE :TYPE :GENERAL    by :LOOP :RECURSIVE
   rewrite input.sem with :IF :REDUCE :TYPE :DEFAULT    by :LOOP :RECURSIVE
   rewrite input.prag with :PHASE :JAPANESE :PRSP :INIT by :RECURSIVE
   rewrite input.sem with :PHASE :JAPANESE              by :LOOP :RECURSIVE
   rewrite input.sem with :PHASE :J-E :TYPE :IDIOM      by :LOOP :RECURSIVE
   rewrite input.sem with :PHASE :J-E :TYPE :GENERAL    by :LOOP :RECURSIVE
   rewrite input.sem with :PHASE :J-E :TYPE :DEFAULT    by :LOOP :RECURSIVE
   rewrite input.prag with :PHASE :ELLIPSIS-RESOLUTION  by :RECURSIVE
   rewrite input.sem with :PHASE :ENGLISH               by :RECURSIVE
   rewrite input.sem with :PHASE :ASPECT-INIT           by :RECURSIVE
   rewrite input.sem with :PHASE :ASPECT                by :RECURSIVE
   return input
end
```

規則2
```
on <reln> UNKNOWN-IFT in :IF :REDUCE :TYPE :Default
   "DEFAULT IFT is INFORM"
   in= [[reln UNKNOWN-IFT]
[agen ?agen]
[recp ?recp]
[obje ?obje]]
   set ?output to [[reln inform]
   [agen ?agen]
   [recp ?recp]
   [obje ?obje]]
   if ?obje.infmann then
?output.infmann = ?obje.infmann
delete infmann from ?obje
   endif
   if ?obje.conect then
?output.conect = ?obje.conect
delete conect from ?obje
   endif
   set parameter :IFT :INFORM
   out= ?output
end
```

規則3
```
"on <reln> UNKNOWN-IFT in :IF :REDUCE :Type :General
   もしもし-OPEN_DIALOGUE --> PHATIC"
   in= [[reln UNKNOWN-IFT]
[agen ?agen]
[recp ?recp]
[obje @OBJE
      [[reln もしもし-OPEN_DIALOGUE]
       [agen ?agen]
       [recp ?recp]
       ?rest]]]
   set parameter :IFT :PHATIC
   out= [[reln PHATIC]
 [agen ?agen]
 [recp ?recp]
 [obje @OBJE]]
end"
```

```
規則4 "on <reln> UNKNOWN-IFT in :IF :REDUCE :Type :General
        "失礼する -CLOSE_DIALOGUE --> PHATIC"
        in= [[reln UNKNOWN-IFT]
      [agen ?agen]
      [recp ?recp]
      [obje @OBJE
           [[reln 失礼する -CLOSE_DIALOGUE]
            [agen ?agen]
            [recp ?recp]
            ?rest]]]
        set parameter :IFT :PHATIC
        out= [[reln PHATIC]
              [agen ?agen]
              [recp ?recp]
       [obje @OBJE]]
      end"
```

```
規則5 on <reln> UNKNOWN-IFT in :IF :REDUCE :Type :General
        S-REQUEST + INFORMREF --> QUESTIONREF"
        in= [[reln UNKNOWN-IFT]
             [agen ?agen]
             [recp ?recp]
             [obje [[reln S-REQUEST]
                    [agen ?agen]
                    [recp ?recp]
                    [obje [[reln INFORMREF]
                           [agen ?recp]
                           [recp ?agen]
                           [obje ?obje]]]
                    ?rest]]]
        set parameter :IFT :QUESTION
        out= [[reln QUESTIONREF]
              [agen ?agen]
              [recp ?recp]
              [obje ?obje]
              ?rest]
      end
```

```
規則6 on <reln> UNKNOWN-IFT in :IF :REDUCE :Type :General
        "ありがとう -THANKING -> EXPRESSIVE"
        in= [[RELN UNKNOWN-IFT]
             [agen ?agen]
             [recp ?recp]
             [obje @OBJE
           [[reln ありがとう -THANKING]
            [agen ?agen]
            [recp ?recp]
            ?rest]]]
        set parameter :IFT :EXPRESSIVE
        out= [[reln EXPRESSIVE]
              [agen ?agen]
              [recp ?recp]
       [obje @OBJE]]
      end
```

```
規則7 on <reln> UNKNOWN-IFT in :IF :REDUCE :Type :General
        "させる -PERMISSIVE + てもらう -RECEIVE_FAVOR --> PROMISE"
        in= [[reln UNKNOWN-IFT]
      [agen ?sp]
      [recp ?hr]
      [obje [[reln てもらう -RECEIVE_FAVOR]
             [agen ?sp]
             [recp ?hr]
             [obje [[reln させる -PERMISSIVE]
            [agen ?hr]
            [recp ?sp]
            [obje ?obje]
            ?rest1]]
             ?rest2]]]
        out= [[reln PROMISE]
       [agen ?sp]
       [recp ?hr]
       [obje ?obje]
       ?rest1
       ?rest2]
      end


規則8 on <reln> UNKNOWN-IFT in :IF :REDUCE :Type :General
        S̈-REQUEST + INFORMIF --> QUESTIONIF¨
        in= [[reln UNKNOWN-IFT]
             [agen ?agen]
             [recp ?recp]
             [obje [[reln S-REQUEST]
                    [agen ?agen]
                    [recp ?recp]
                    [obje [[reln INFORMIF]
                           [agen ?recp]
                           [recp ?agen]
                           [obje ?obje]]]
                    ?rest]]]
        set parameter :IFT :QUESTION
        out= [[reln QUESTIONIF]
              [agen ?agen]
              [recp ?recp]
              [obje ?obje]
              ?rest]
      end


規則9 on <reln> QUESTIONREF in :IF :REDUCE :Type :General
        Q̈UESTIONREF + う -GUESS --> QUESTIONREF¨
        in= [[reln QUESTIONREF]
             [agen ?agen]
             [recp ?recp]
             [obje [[parm ?parm]
                    [restr [[reln う -GUESS]
                            [expr ?agen]
                            [obje ?obje]
                            ?rest1]]]]
             ?rest]
        out= [[reln QUESTIONREF]
              [agen ?agen]
              [recp ?recp]
              [obje [[parm ?parm]
                     [restr ?obje]]]
              ?rest
              ?rest1]
      end
```

```
規則 10 on <reln> QUESTIONIF in :IF :REDUCE :Type :General
          QUESTIONIF(IFT) + う -GUESS --> QUESTIONIF"
          in= [[reln QUESTIONIF]
               [agen ?agen]
               [recp ?recp]
               [obje [[reln う -GUESS]
                      [expr ?agen]
                      [obje ?obje]
                      ?rest2]]
               ?rest]
          out= [[reln QUESTIONIF]
                [agen ?agen]
                [recp ?recp]
                [obje ?obje]
                ?rest
                ?rest2]
       end
```

```
規則 11 on <reln> UNKNOWN-IFT in :IF :REDUCE :Type :General
          in= [[reln UNKNOWN-IFT]
               [agen ?sp]
               [recp ?hr]
               [obje [[RELN ね -CONFIRMATION]
                      [OBJE ?obje]
                      ?rest]]]
          out= [[RELN QUESTIONCONF]
                [obje ?obje]
                ?rest]
       end
```

```
規則 12 on <reln> UNKNOWN-IFT in :IF :REDUCE :Type :General
          下さい -REQUEST --> REQUEST"
          in= [[reln UNKNOWN-IFT]
               [agen ?agen]
               [recp ?recp]
               [obje [[reln 下さい -REQUEST]
                      [agen ?agen]
                      [recp ?recp]
                      [obje ?obje]
                      ?rest]]]
          set parameter :IFT :request
          out= [[RELN REQUEST]
                [attd declarative]
                [manner direct]
                [agen ?agen]
                [recp ?recp]
                [obje ?obje]
                ?rest]
       end
```

```
規則 13 on <reln> UNKNOWN-IFT in :IF :REDUCE :Type :General
          願う -REQUEST --> REQUEST"
          in= [[reln UNKNOWN-IFT]
               [agen ?agen]
               [recp ?recp]
               [obje [[reln 願う -REQUEST]
                      [agen ?sp]
                      [recp ?hr]
                      [obje ?obje]
                      ?rest]]]
          set parameter :IFT :REQUEST
          out= [[reln REQUEST]
                [attd declarative]
                [manner direct]
                [agen ?sp]
                [recp ?hr]
                [obje ?obje]
                ?rest]
       end
```

```
規則 14 on <infattd restr reln> 失礼ですが -1
          " ift:request + 失礼ですが --> ift:requst manner:indirect"
          in= [[reln request]
               [infattd [[parm !x[]]
                         [restr [[reln 失礼ですが -1]
                                 [entity !x]]]]]
               ?rest]
          input.manner = $indirect
          out= [[reln request]
                ?rest]
       end
```

```
規則 15 on <reln> REQUEST in :IF :REDUCE :Type :General
          てもらう -receive_favor + REQUEST
       ex) てもらい (たい と思う のです が)  -->  REQUEST"
          in= [[reln REQUEST]
               [agen ?sp]
               [recp ?hr]
               [obje [[reln てもらう -RECEIVE_FAVOR]
                      [agen ?sp]
                      [recp ?hr]
                      [obje ?obje]
                      ?rest1]]
               ?rest2]
          set parameter :IFT :request
          out= [[reln REQUEST]
                [agen ?sp]
                [recp ?hr]
                [obje ?obje]
                ?rest1
                ?rest2]
       end
```

```
規則 16 on <reln> INFORM in :IF :Reduce :Type :General
          in= [[reln INFORM]
               [agen ?sp]
               [recp ?hr]
               [obje [[reln WANT]
                      [expr ?sp]
                      [obje [[reln てもらう -RECEIVE_FAVOR]
                             [agen ?sp]
                             [recp ?hr]
                             [obje ?obje]
                             ?rest1]]
                      ?rest2]]
               ?rest3]
          set parameter :IFT :REQUEST
          out= [[reln REQUEST]
                [agen ?sp]
                [recp ?hr]
                [obje ?obje]
                ?rest1
                ?rest2
                ?rest3]
       end
```

```
規則 17 on <reln> QUESTIONIF in :IF :REDUCE :Type :General
          "できる -POSSIBLE + QUESTIONIF --> REQUEST"
          in= [[RELN QUESTIONIF]
               [agen ?agen]
               [recp ?agen]
               [obje [[RELN できる -POSSIBLE]
                      [expr ?expr]
                      [obje @OBJE
                            [[agen ?agen]
                             ?other]]]]
               ?rest]
          set parameter :IFT :REQUEST
          out= [[reln REQUEST]
                [attd interrogative]
                [manner indirect]
                [agen ?agen]
                [recp ?recp]
                [obje @obje]
                ?rest]
       end
```

```
規則 18 ON <reln> UNKNOWN-IFT IN :IF :REDUCE :Type :General
          "が +MODERATE -> INFROM (indirect)"
          in= [[reln UNKNOWN-IFT]
               [agen ?agen]
               [recp ?recp]
               [obje [[reln が -MODERATE]
                      [obje ?obje]
                      ?rest2]]
               ?rest]
          out= [[reln INFORM]
                [manner indirect]
                [agen ?agen]
                [recp ?recp]
                [obje ?obje]
                ?rest
                ?rest2]
       END
```

```
規則 19 on <reln> たい -DESIRE in :IF :Reduce :Type :General
          たい -DESIRE  --> want"
          in= [[reln たい -DESIRE]
               [expr ?expr]
               [obje ?obje]
               ?rest]
          out= [[reln WANT]
                [expr ?expr]
                [obje ?obje]
                ?rest]
       end


規則 20 on <iden parm restr reln> 予定 -1 in :phase :Japanese
          "... する予定だ --> 予定する "
          in= [[RELN だ -IDENTICAL]
               [OBJE []]
               [IDEN [[PARM !X7[[PARM !X1[]]
                                [RESTR [[RELN 予定 -1]
                                        [ENTITY !X1]]]]]
                     [RESTR [[RELN 外の関係の連体修飾]
                             [ARG-1 !X7]
                             [ARG-2 ?X]]]]]
               ?rest]
          out= [[reln 予定する -1]
                [agen []]
                [obje ?X]
                ?rest]
       end


規則 21 on <restr reln> 外の関係の連体修飾 in :Phase :Japanese :Event-or-Object :Event
          in= [[PARM !X7[[PARM !X1[]]
                         [RESTR [[RELN こと -1]
                                 [ENTITY !X1]]]]]
               [RESTR [[RELN 外の関係の連体修飾]
                       [ARG-1 !X7]
                       [ARG-2 ?arg2]]]]
          out= ?arg2
       end


規則 22 on <restr arg-2 reln> 詳しい -1 in :phase :Japanese
          "詳しいこと --> 詳細 "
          in= [[parm !x1[[parm !x2[]]
                         [restr [[reln こと -1]
                                 [entity !x2]]]]]
               [restr [[reln 外の関係の連体修飾]
                       [arg-1 !x1]
                       [arg-2 [[reln 詳しい -1]
                               [agen []]
                               [obje []]
                               ?rest]]]]]
          out= [[parm !x[]]
                [restr [[reln 詳細 -1]
                        [entity !x]]]]
       end
```

```
規則 23 on <reln> 分かる -1 in :PHASE :japanese :event-or-object :object
          "分からない点 --> 質問 "
          in= @question(restr obje)
              [[PARM !X2[[PARM !X1[]]
                         [RESTR [[RELN 点 -1]
                                 [ENTITY !X1]]]]]
               [RESTR [[RELN NEGATE]
                       [OBJE [[RELN 分かる -1]
                              [EXPR []]
                              [OBJE !X2]]]]]]
          @questino = [[parm !x[]]
                       [restr [[reln 質問 -1]
                               [entity !x]]]]
          return @question
       end


規則 24 on <infattd> :unspecified in :phase :japanese
          "swap main clause and subordinate clause"
          in= [[infattd ?obje]
           ?rest]
          => ?obje with :phase :japanese :event-or-object :event
          if it is false then fail endif
          ?obje.obje = [?rest]
          out= ?obje
       end


規則 25 on <restr reln> お気の毒ですが -1 in :PHASE :Japanese :event-or-object :event
          in= [[parm !x[]]
               [restr [[reln お気の毒ですが -1]
                       [entity !x]]]]
          out= [[reln 気の毒だ -1]
                [expr []]
                [obje []]]
       end


規則 26 on <reln> できる -POSSIBLE in :phase :japanese
          in= [[reln できる -POSSIBLE]
               [expr ?expr]
               [obje ?obje]
               ?rest]
          -> ?obje with :phase :japanese :Event-or-Object :Event
          return input
       end


規則 27 on <restr reln> 対する -1 in :PHASE :Japanese :Event-or-Object :event
          in= [[parm ?agen]
               [restr [[reln 対する -1]
                       [agen ?agen]
                       [obje ?obje]
                       ?rest]]]
          -> ?agen with :Phase :japanese :event-or-object :event
          ?agen.obje = ?obje
          out= ?agen
       end
```

```
規則 28 on <restr reln> 払い戻し -1 in :PHASE :Japanese :event-or-object :event
          in= [[parm !x[]]
               [restr [[reln 払い戻し -1]
                       [entity !x]]]]
          out= [[reln 払い戻す -1]
                [agen []]
                [obje []]]
       end


規則 29 on <restr reln> 手続 -1 in :PHASE :Japanese :Event-or-Object :Event
          手続 (Noun) => 手続する (Verb)"
          in= [[parm !x[]]
               [restr [[reln 手続 -1]
                       [ENTITY !x]]]]
          out= [[reln 手続する -1]
                [agen []]]
       end


規則 30 on <restr reln> 割引 -1 in :Phase :Japanese :event-or-object :event
          in= [[parm !X[]]
               [restr [[reln 割引 -1]
                       [entity !X]]]]
          out= [[reln 割引する -1]
                [agen []]
                [obje []]]
       end


規則 31 on <restr reln> 発表 -1 in :Phase :Japanese :Event-or-object :Event
          in= [[parm !x[]]
               [restr [[reln 発表 -1]
                       [entity !x]]]]
          out= [[reln 発表する -1]
                [agen []]
                [obje []]]
       end


規則 32 on <purp parm restr reln> 代わり -1 in :phase :japanese
          "の代わりに -> instead case"
          in= [[purp [[restr [[reln の - 連体修飾]
                              [arg-1 ?x]
                              [arg-2 ?y]]]
                      ?rest1]]
               ?rest]
          out= [[instead ?y]
                ?rest]
       end


規則 33 on <reln> する -1 in :PHASE :Japanese
          "[サ変名詞] + する --> サ変動詞 "
          in= [[reln する -1]
               [OBJE ?OBJE]
               ?rest]
          -> ?OBJE with :PHASE :Japanese :Event-or-Object :Event
          add ?rest to ?OBJE
          out= ?OBJE
       end
```

```
規則 34 on <reln> 行う -1 in :phase :japanese
          「行う」の object 格要素を動詞に派生させる"
          in= [[reln 行う -1]
               [obje ?obje]
               ?rest]
          -> ?obje with :phase :japanese :event-or-object :event
          if it is false then fail endif
          add ?rest to ?obje
          out= ?obje
       end


規則 35 on <reln> 希望する -1 in :PHASE :Japanese
          "N を希望する --> N したい"
          in= [[RELN 希望する -1]
               [agen ?agen]
               [obje ?obje]
               ?rest]
          -> ?obje with :PHASE :Japanese :Event-Or-Object :Event
          if it is false then fail endif
          ?obje.agen = ?agen
          out= [[RELN WANT]
                [expr ?agen]
                [obje ?obje]
                ?rest]
       end


規則 36
       on <reln> そうですか -CONFIRMATION in :PHASE :J-E :type :idiom
          in= [[RELN そうですか -CONFIRMATION]
               [agen ?agen]
               [recp ?recp]
               ?rest]
          out= [[RELN all_right-IDIOM-1]
                [agen ?agen]
                [recp ?recp]
                ?rest]
       end


規則 37 on <reln> ありがとう -THANKING in :PHASE :J-E :type :idiom
          "change case, INFMANN -> MANN"
          in= [[RELN ありがとう -THANKING]
               [AGEN ?agen]
               [RECP ?recp]
               [INFMANN [[parm !X[]]
                         [restr [[reln どうも -1]
                                 [entity !X]]]]]
               ?rest]
          out= [[reln thank-VT-1]
                [agen ?agen]
                [recp ?recp]
                [mann [[parm !X[]]
                       [restr [[reln much-ADV-1]
                               [entity !X]
                               [DEGR [[parm !Y[]]
                                      [restr [[reln very-ADV-1]
                                              [entity !Y]]]]]]]]]
                ?rest]
       end
```

```
規則 38 on <reln> だ-STATEMENT in :PHASE :J-E :type :IDIOM :IFT :INFORM
          "そうだ --> That is right"
          in= [[reln だ-STATEMENT]
               [obje [[parm !X[]]
                      [restr [[reln そう-1]
                              [entity !X]]]]]
               ?rest]
          set ?SP to ROOT.PRAG.SPEAKER
          set ?HR to ROOT.PRAG.HEARER
          out= [[RELN that_is_right-IDIOM-1]
                [AGEN ?SP]
                [RECP ?HR]]
       end
```

```
規則 39 on <reln> どういたしまして-GREETING in :PHASE :J-E :type :idiom
          in= [[RELN どういたしまして-GREETING]
               [AGEN ?agen]
               [RECP ?recp]]
          out= [[reln you_are_wellcome-IDIOM-1]
                [agen ?agen]
                [recp ?recp]]
       end
```

```
規則 40 on <reln> questionref in :PHASE :J-E :Type :Idiom
          "どのような用件ですか -> May I help You?"
          in= [[RELN questionref]
               [AGEN @sp !X6[[LABEL *SPEAKER*]]]
               [RECP @hr !X5[[LABEL *HEARER*]]]
               [OBJE [[PARM !X [[parm !X1 [[PARM !X3[]]
                                           [RESTR [[RELN 用件-1]
                                                   [ENTITY !X3]]]]]
                                  [restr [[reln どのような-1]
                                          [arg-1 !X1]]]]]
                      [RESTR [[RELN だ-IDENTICAL]
                              [obje []]
                              [iden !X]
                           ?rest2]]]]
               ?rest]
          out= [[RELN QUESTIONIF]
                [AGEN @sp]
                [RECP @hr]
                [OBJE [[RELN permission]
                       [OBJE [[RELN HELP-VT-1]
                              [AGEN @sp]
                              [RECP @hr]]]]]]
       end
```

```
規則 41 on <reln> REQUEST in :PHASE :J-E :Type :Idiom
          "[それでは] よろしくお願いします -> Thank you very much"
          in= [[RELN REQUEST]
               [agen ?agen]
               [recp ?recp]
               [OBJE []]
               [INFMANN [[PARM !X1[]]
                         [RESTR [[RELN よろしく -1]
                                 [ENTITY !X1]]]]]
               ?rest]
          out= [[RELN EXPRESSIVE]
                [agen ?agen]
                [RECP ?recp]
                [OBJE [[reln thank-VT-1]
                       [agen ?agen]
                       [recp ?recp]
                       [MANN [[parm !X []]
                              [restr [[reln MUCH-ADV-1]
                                      [entity !X]
                                      [DEGR [[parm !Y[]]
                                             [restr [[reln VERY-ADV-1]
                                                     [entity !Y]]]]]]]]]]]
       end

規則 42 on <reln> よい -1 in :PHASE :J-E :type :idiom
          in= [[reln よい -1]
               [obje []]
               [cond [[parm !x[]]
                      [restr [[reln たら -CONDITIONAL]
                              [entity !x]
                              [iden ?obje]]]]]
               ?rest]
          out= [[reln possibility]
                [obje ?obje]]
       end

規則 43 on <reln> なくてはいけない -MUST in :PHASE :J-E :Type :Idiom
          "てもらわなくてはいけない --> obliteration"
          in= [[reln なくてはいけない -MUST]
               [AGEN ?AGEN]
               [OBJE [[reln てもらう -RECEIVE_FAVOR]
                      [AGEN ?AGEN]
                      [RECP ?RECP]
                      [OBJE ?OBJE]]]
               ?rest]
          out= [[reln OBLIGATION]
                [AGEN ?AGEN]
                [OBJE ?OBJE]
                ?rest]
       end

規則 44 on <reln>割引する -1
          in= [[reln 割引する -1]
               ?rest]
          out= [[reln make-VT-4]
                [obje [[parm !x[]]
                       [restr [[reln discount-N-1]
                               [entity !x]]]]]
                ?rest]
       end
```

```
規則 45 on <reln> ない -adjective
          in= [[reln ない -adjective]
               [OBJE ?OBJE]
               ?rest]
          out= [[reln NEGATE]
                [obje [[reln be-vi-1]
                       [OBJE ?OBJE]
                       ?rest]]]
       end


規則 46 on <reln> 発表する -1
          in= [[reln 発表する -1]
               [agen ?agen]
               [obje ?obje]
               ?rest]
          switch ?obje
          case []
               out= [[reln make-VT-4]
                     [agen ?agen]
                     [obje [[parm !X[]]
                            [restr [[reln presentation-N-2]
                                    [entity !X]]]]]
                     ?rest]
          default
              out= [[reln present-VT-2]
                    [agen ?agen]
                    [obje ?obje]
                    ?rest]
          endswitch
       end


規則 47 on <reln> 決まる -1
          ¨～が決まる --> be decided¨
          in= [[RELN 決まる -1]
               [obje ?obje]
               ?rest]
          out= [[RELN passive]
                [obje [[RELN decide-VT-1]
                       [obje ?obje]
                       ?rest]]]
       end


規則 48 on <reln> send-vt-1 in :Phase :ASPECT-INIT
          ¨send          [[mome +][proc -][durt -]]¨
          in= [[reln send-vt-1]
               ?rest]
          input.lex.aspect = [[mome +]
                              [proc -]
                              [durt -]]
          out= input
       end


規則 49 on <aspt> unrl in :PHASE :ASPECT
            in= [[aspt unrl]
                 ?rest]
            => input with :aspect :unrl
            return it
        end
```

```
規則50 on <lex> :unspecified in :aspect :UNRL
          in= [[lex [[aspect [[mome +]
                             [proc -]
                             [durt -]]]]]
                    ?rest]
          input.tense = $future
          input.aspect = [[perf -]
                          [prog -]]
          return input
       end
```