Vol.2009-NL-193 No.11
2009/9/29

自然言語のモジュラー構文構造について[1]

安原宏[†]

自然言語の構文構造をどのように捉えるかは，それ以降の自然言語処理に重要な影響を与える．自然言語のアプリケーションやサービスシステムの開発において，種々の言語処理ツールを組み合わせて構築されることを想定すると，構文解析の出力インタフェースの要件は，できるだけ文法カテゴリに依存しない表現かつ，どの自然言語にも共通的に適用できる表現が望ましい．これまでは，各ツールが単独であったり，特定のツールを想定した垂直的な開発が主であったためこのような制約条件は余り注目されていなかった．しかし，様々な機関や個人で開発された NLP ツール，機械翻訳システム，自然言語サービスが公開されるようになると，このようなインタフェースの共通化への要求が自然に生まれてくる．ここでは，構文解析の段階でこの要件を満たすと考えるモジュラー構文構造を提案し，オープンな構文解析エンジンの解析出力を当該モジュラー構造に変換することを示す．また，この構造を利用することで得られる有効性の一つの例示として文の言い換え処理を取り上げ論理的に素直に実現できることを示す．

Modular Syntactic Form of Natural Language

Hiroshi Yasuhara[†]

What kind of syntactic structures are better for natural language applications? In the syntactic analysis field, there are various formats such as phrase structure, dependency structure, LFG structure and case structure. These structures are dependent on the underlining linguistic grammars. The age of Web services requires a standardized interface between syntactic analysis and their applications such as machine translation systems because application systems should be dynamic and free from components. In order to satisfy the conditions of grammar independent and language independent syntactic structures, a modular syntactic form is provided and shown to be implemented in an open syntactic parser. The modular syntactic form shows logical processing features in sentence paraphrasing processing.

1. はじめに

日本語処理，自然言語処理（NLP）は，機械翻訳という明確なサービスのための中核技術として長年研究開発が進められて来た．最近では，NLP の基盤となる形態素解析や構文解析のためのツールが公開されおり，自然言語処理応用システムの開発への弾みとなっている．今後もこのようなオープンリソースの時代が継続することを想定すると，様々なところで開発されているツールやサービスのインタフェースを共通化して欲しいという要求が自然に発生する．現時点での切り口は，形態素解析，構文解析であろう．ここでは，構文解析の出力形式について議論する．構文解析の出口は，多種多様な自然言語処理応用につながっているためである．構文解析そのものが様々な言語理論をベースに開発されているため個別の文法とは独立でかつ特定の自然言語に依存しないインタフェースを設定できれば，それ以降のフェーズに位置する自然言語応用システムの開発に大きなインパクトを与えることが期待できる．例えば，A 大学の形態素解析，B 大学の構文解析を利用して，C 社の読解支援システム及び D 社の機械翻訳システムといった種々の組み合わせが可能になる．

ここでは，２でモジュラー構文構造の定義を与え，その言語的な特徴を述べる．３で，モジュラー構文構造をオープンにされている構文解析エンジンを利用して実装したことを述べる．４でモジュラー構文構造の応用について述べ，この構造が多くの可能性を持っていることを示す．

2. モジュラー構文構造

2.1 背景

一般的に構文構造は，辞書（形態素解析の出力）と構文解析規則により生成されるものであり，文法と密接に関連する．例えば，係り受け解析を使用するときは係り元の文節から係り先の文節への係り受け関係を含めた情報の集合となる．１つの文節が必ずどれか一つの文節に係るという条件の元では，それは木構造になる．各ノードには単語が入る．また，句構造文法を用いたときは，句構造文法が使用する文法カテゴリをノードにした木構造になる．単語はリーフノードにのみ出現する．このほかにも，格文法や LFG 等の種々の言語理論の基でそれぞれの構文構造が出力される．

† 特定非営利活動法人セマンティックコンピューティング研究開発機構
Institute of Semantic Computing

1 本研究の一部は，（財）機械システム振興協会が(財)JKA の競輪補助金の交付を受けて(財)日本特許情報機構に委託したものの財源をもとに受託したものである．

2.2 モジュラー構文構造の定義と性質

モジュラー構文構造は，依存構造をより構造化したものである．依存構造は，単語と単語の係り受け関係を表現するもので，公開されている日本語構文解析の出力も依存構造である [1][2]．係り側と受け側の文節は内容語（自立語）とし，係り受け関係は機能語（付属語）に対応する．つまり，係り受けの文節とその間の関係を表す有向ラベル付きアークで記述できる．図 1 では A,B,C が文節，r，s が関係である．以降，係り元を子供，係り先を親という表現も使う．親に係る全ての子ノードを含んだ親のノードをモジュラー構文構造（MSF: Modular Syntactic Form）と呼ぶ．一部を含むのは MSF ではない．MSF は入れ子構造であり，各々をノードと呼ぶ．これはリカーシブに適用される．図 1 の下側の，A を含んだ B，A と B を含んだ C が MSF である．各ノードには，モジュラー文法とモジュラー辞書が関連する．これは，従来の依存構造では明示的に議論されていない部分である．モジュラー文法と辞書は後ほど説明する．以上のような対象であるモジュラー構文構造を特徴付ける性質を述べると，（1）係り先のノードが係り元ノードを支配していると見なすことである．つまり，複数文節から修飾される文節はそれらを全て支配していると見ることである．図 1 の C は A,B を支配している．（2）親は，子供たちの言語標記を受け継ぎ，自らの言語標記と合わせて，一つの充足した言語標記になる．充足したとは，正規な文法で解釈できるという意味で，モジュラー構文構造は非文法的なものではないという性質を持つ．例で示すと，「私の好きな歌手」には 2 つの MSF が存在する． B=(私の)好きなと C=(私の好きな)歌手であるが，何れも文法的に正しい表現である．末端の A は単語である．（3）さらに，モジュラー構文構造は表層的に連続した単語列であるということである．このことは論理的に意味のあるものに限定した N-gram であるともいえる．このため，モジュラー構文構造は自然言語を特徴付ける拡張単語クラスを与えるという見方が出来る．このクラスは単語辞書の拡張であり，文法的に正しい語列であるので正規型の用例とも見なせる．以上のような見方から自然言語はモジュラー構文構造の積み重ねで成立するのではないかという大きな命題が生まれてくる．その結論として以下を主張する．

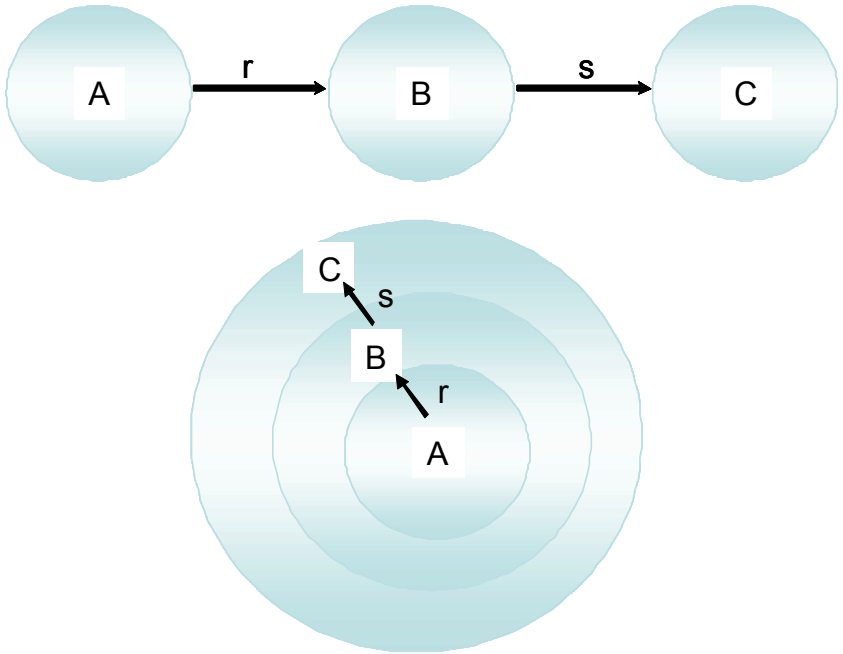


図 1　係り受け構造とモジュラー構文構造

基本法則：　自然言語は，モジュラー構文構造から構成される．

以下，この主張に対する説明を行う．先ず，日本語や英語の文は係り受け関係から，ルート（係り受けの終端ノード）が一つの木構造になる．他の言語に関しても，逐語的に翻訳できると仮定すればこのことは正しいといえる．モジュラー文法とモジュラー辞書に関しては，図 2 のような見方をとる．各ノードに対してモジュラー文法とモジュラー辞書が対応していて，各ノードは文法的に解釈できる．ここで，ノードとは単語ではなく子供やその子孫を含めたそれら親の配下の全てが集まったもののことである．ノードを解釈する文法をモジュラー文法と呼ぶ．モジュラー辞書とは，モジュラー構文構造を辞書登録したもので単語辞書を包含している．従って，モジュラー辞書は拡張単語辞書の概念を提供する．親子関係を持つ一般的なノードにおいては，全体を辞書に登録することも可能であり，一部の下位ノードを辞書登録にすることも可能である．つまり，図 3 の様に辞書に記載する項目と文法の複雑度は逆関係になる．一方が詳細に記録されると他方は単純で済

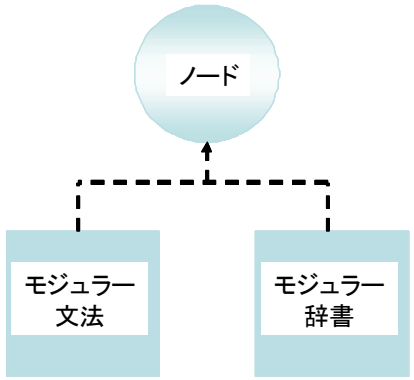


図 2　ノードに対応する文法・辞書

むという関係である．図２のノードを文相当とすると，これは文を解釈する文法と辞書の集合になる．このように，モジュラーという表現は構造だけではなく，辞書と文法にも関係する概念である．単語をベースとする辞書から複数単語（Multi-Words-Entry）辞書への拡張は，共起関係や連語等で個別に検討，作成されてきたが，モジュラー辞書は拡張単語辞書の体系的な構築方式を与える．文法的にも，モジュラー文法はサブグラマーという見方をより体系的に記述する方式を与える．文法は文を解釈するというよりもモジュラー構文構造を解釈すると考える．ノードが複雑になれば，その文法も複雑になる．しかし，文法自体がモジュラー構造を対象にするので組織立ったビルディングブロック方式になる．拡張辞書を増やせば文法は単純化する．もし文中に出現する全てのモジュラー構文構造が辞書登録されていれば文法はパターンマッチングで済む．

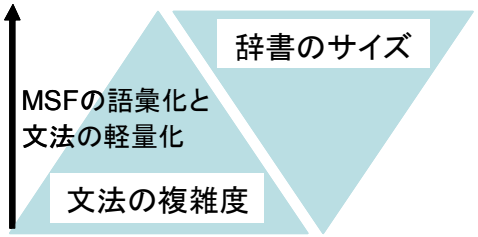


図３　文法と辞書のトレードオフ

親ノードの配下に n 個（n=1～4，8）の単語を支配するモジュラー構文構造 MSFn の具体例を以下に示す．網掛けの文字列の右端が親となる単語である．後述する CDL による形式的な表現を付録に示す．

【MSF1 の例】

①機械翻訳の 一方法として最近盛んに研究されている方式に，統計的機械翻訳がある．
②なお以下の説明では，第１の 言語を日本語とし，「Ｊ」と表記する．
③上述のインサイド - アウトサイド確率の組合せにより，対の 累積発生数を求める以下の式が得られる．
④すなわち，ある しきい値以下のスコアしか持たない出力候補は翻訳の候補から除外される．
⑤対訳コーパス７０に含まれる対訳文の各々は，日本語の文と，それに 対応する英語の文とからなっている．

【MSF2 の例】

①この発明は統計的機械翻訳装置に関し，特に，対訳コーパスを用いた学習により，構造が 大きく 異なった言語間でも精度よく翻訳する事が可能な統計的機械翻訳装置に関する．
②翻訳時には，具体的には 次の 様な作業を行なう．
③統計的機械翻訳では，第１の言語の文と第２の言語の文との対訳文を多数含む対訳コーパスを用いた学習により予め翻訳モデルを作成しておき，この 翻訳モデルを 用いて翻訳を行なう．（付録参照）
④探索空間が 非常に 大きいので，上記した二段階の双方において，出力の一部のみを残すためのサイズしきい値を設定する．
⑤この 対訳コーパスの 概略を以下の表１に示す．

【MSF3 の例】

①さらに，各チャンクにヘッド語を設定する事により，ヘッド語からの位置によって，２段階の 並べ替えに 対する 制約を設ける事が可能になっている．
②さらにこの 推定された 条件確率に 基づいて，各モデルのパラメータを計算する（Ｍステップ）
③すなわち ＥＭアルゴリズムである 最大値解に 収束したとしても，それがグローバルな最大値であるという保証はない．
④要求される 膨大な 計算量に 対処するために，インサイド - アウトサイド推定手順に近似を適用する．
⑤さらに，実例に よる スコアの 加算方式を導入する．

【MSF4 の例】

①この 式に ベイズの 定理を 適用する 事により，＾Ｅ＝ａｒｇｍａｘＥＰ（Ｅ）Ｐ（Ｊ｜Ｅ）が得られる
②この 従来例で 用いられている 各モデルでの シンボルの意味については，非特許文献１を参照されたい
③こうした 手続を 用いて 翻訳モデルを 生成する事により，同種の言語間の翻訳モデルではその対応関係を比較的精度よく捕らえる事ができる．
④同様に，挿入される語は語彙モデルパラメータを用いて選択され，二項分布に より 決定される 位置に 挿入されるに過ぎない．
⑤出力チャンク列選択手段は，出力チャンク作成手段により作成される出力チャンク列のうち，尤度が 所定の 値以上の ものを 選択してチャンク並べ替え手段に与えるための手段を含んでもよい．

【MSF8 の例】

①統計的機械翻訳では，第１の 言語の 文と 第２の 言語の 文との 対訳文を 多数含む対訳コーパスを用いた学習により予め翻訳モデルを作成しておき，この翻訳モデルを用いて翻訳を行なう．（付録参照）
②なお，語アライメントでは，ソース文の 単語の 各々に 対して １対多の 関係でのターゲット単語の 生成を 許すものとする．
③しかし，語の削除及び挿入の様な現象についてこの様に弱いモデル化しか行なわない場合，日本語と 英語の 様に 互いに 大きく 異なる 言語の 組合せに ついては，十分な翻訳性能を期待する事はできない．
④こうする事により，各チャンクは情報の 付加も 削除も なしに ひとまとまりの 意味を 表すと 考える 事ができる．

⑤上記 した 処理が 可能な 様に プログラム された コンピュータを 用いて以下の実験を行なった．

ある特許明細書の 339 文中からモジュラー構文構造を抽出した．その出現回数を表 1 に示す．

表１　339 文中の MSF の重み毎の出現頻度

MSF 配下の単語数（重み）	重複を含む出現回数
MSF1：１個	６２６
MSF2：２個	３６８
MSF3：３個	２２５
MSF4：４個	１３６
MSF8：８個	４１

MSF1 の場合，1 文当り平均 2 回出現していることになるが，個数が増えるに従い漸次低下している．

3. モジュラー構文構造解析の実装について

3.1 CDL(Concept Description Language)

モジュラー構文構造作成は，文節の係り受けの関係を主と従の関係で入れ子にすることが基本メカニズムである．このような構造記述に適しているのが CDL である．CDL の仕様は，[3]を参考にしていただきたい．ここでは，実装を理解するのに必要なところを述べる．CDL という名称に示されるように CDL は，概念を記述のベースとする．更に概念は，実体概念と関係概念に分類される．データベースの E-R 記述に近いものである．自然言語の場合は，表層的には自立語（内容語）が実体概念に対応し，付属語（機能語）が関係概念に対応する．これを深層概念で記述することも研究している．今回は表層レベルで実装している．実体概念は｛｝で記述し，関係概念は［］で記述する．前者をノードと呼び，後者をアークと呼ぶ．XML 構文の様に全て＜＞タグで記述するのも可能であるが，言語処理では両者を区別した方が，処理が簡潔になるだけでなく，内容が読みやすくなる．図式的に書けば実体概念ノード｛A｝から実体概念ノード｛B｝へ関係概念アーク［R］で結合した有向グラフで記述できる．実体概念は，ヘッドとボディに分ける．その区切りを；で記述する．ヘッド部では，実体概念を代表する名称と属性を記述し，ボディ部ではヘッドの実体概念を修飾する内部構造を記述する．最も基本の実体概念は，ボディが空の単語概念である．文中に出現する単語は単語辞書に定義されたクラス概念のインスタンスだと解釈する．その識別のためにノードは#id でラベル付けされている．

3.2 モジュラー構文構造化

モジュラー構文構造を生成するために，京都大学黒橋研究室で開発されている Juman(形態素解析)と KNP(構文解析)を利用した．前提条件は係り受け関係が抽出できればどのようなパーザを利用しても可能である．実際，今回の実装以外にも別の構文解析エンジンと結合して動作を確認している．

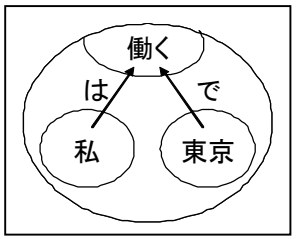
先ず，構文解析の出力である依存構造（係り受け構造）をそれと自然に対応する CDL 形式に変換する．以下のような例を一般化したものである．

文「私は東京で働く．」は，CDL 形式で前半に自立語対応ノード，後半に係り受け関係アークを置く．属性情報は，pos は品詞，fw は自立語に続く付属語を示す．用言に対しては活用形やモダリティーを付与することが可能であるが省略している．また，アークの左右の#id はソース側ノードの id 及びターゲット側ノードの id を示す．関係概念の名称は，表層関係を記述している．これは依存構造と等価な表現である．

```
{#s1  文  sent=<私は東京で働く．>;
{#0 私   pos=<名詞>          fw=<は>; }
{#1 東京          pos=<名詞>          fw=<で>; }
{#2 働く          pos=<動詞>          fw=<．>; }
[#0 未格          #2]
[#1 デ格          #2]
}
```

次にこの構造をモジュラー構文構造に構造変換する．上記の係り受けの依存関係が入れ子構造に変換される．つまり「働く」という親ノードに対して，それに依存する「私は」と「東京で」が親ノードのボディに入り，それらはアークでヘッドに関係付けられる．親が子の上に来る．グラフ表示を対応させると構造が分かり易い．

```
{#s1     文       sent=<私は東京で働く．>;
{#2 働く          pos=<動詞>          fw=<．>;
  {#0   私        pos=<名詞>          fw=<は>; }
  [#0   未格      #2]
  {#1   東京      pos=<名詞>          fw=<で>; }
  [#1   デ格      #2]}
}
```



CDL 表現の特徴は，関係概念をヘッド概念から切り離していること．つまり「で」は「東京」というモジュラー構文構造には含まれない．また，文の主概念をトップに置き，１ノード１行で下に続けるため，長文においても横に伸びる量は少ない．実際の実行結果を付録に掲載する．本来は，以下 rep 属性で示すようにモジュラー構文構造にその下位ノードの表現を含んで代表する文字列を記述するのが正しい表現であるが，冗長になるので省略している．

```
{#s1    文      sent=<私は東京で働く．>;
{#2 働く        rep=<私は東京で働く>        pos=<動詞>        fw=<．>;
  {#0   私      rep=<私>  pos=<名詞>        fw=<は>; }
  [#0   未格    #2]
  {#1   東京    rep=<東京>          pos=<名詞>          fw=<で>; }
  [#1   デ格    #2]}
}
```

以上の 2 段階の処理で CDL 形式のモジュラー構文構造が生成できる. 次に開発すべき項目はこの表現上で必要となるノードとアークの削除・生成・挿入・マッチング等のプリミティブな操作関数である．この上で，CDL 型 SQL のような構造検索関数が必要である.

4. モジュラー構文構造の応用

モジュラー構文構造を自然言語の基本構成要素と捉えると, その応用は NLP 全般に波及すると考えられる．単語を想定したときに単語辞書，単語切り出し，対訳辞書，単語検索，N-gram，キーワード検索等が NLP と関係するのと同様に，モジュラー構文構造の場合は，構造化を利用してより高度な NLP と関連する.

4.1 言い換え処理

言い換え処理の中で長文分割というのがある［4]．これは，長い文を短くすることである．実験では，CDL 型モジュラー構文構造から連体修飾と連用修飾を検出して，そこで 2 分割することを実施した.

連体修飾の場合，体言ノードに繋がる連体節ノードをアークから発見し（その際，連体節ノード$b の重みをチェックして，一定以上のときのみ実行)，体言ノードを連体節ノードに戻した文（分割 2）と連体節ノードを消去した文（分割 1）に分ける．プログラム的には，以下のような処理である：

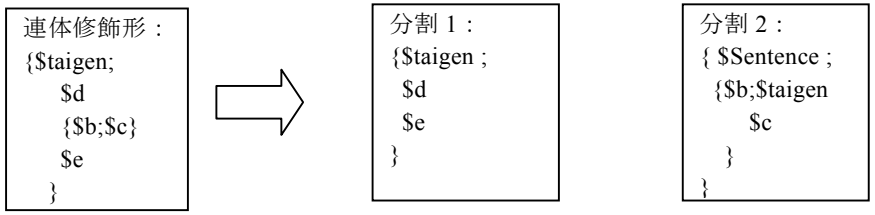


図４　CDL 表現の連体修飾文の分割

連用修飾の場合は，ノードの pos が用言かつアークの関係名は連用修飾を条件としてグラフマッチングを行う．分離するノードの重みが所定の値以上であれば分割したことになる．プログラム的には，以下のような処理である：

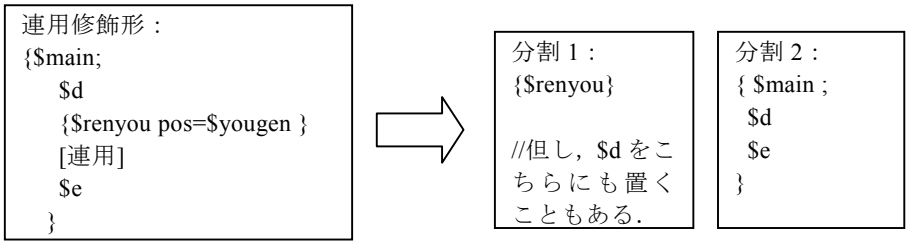


図５　CDL 表現の連用修飾文の分割

但し，いずれの場合も文と文の接続，省略，照応といった文脈処理が残っている.

4.2 NLP デバッガー

モジュラー構文構造を NLP のテストセットとして NLP デバッガーを作ることを検討している．例えば，ある分野の NLP を開発するために，その分野のテキストからモジュラー構文構造を出来るだけ多く抽出して DB 化する．これを配下の単語数によってクラス分けする．そして，これをテストデータとして小さいものから順に NLP に適用してみる（パーザ自体を評価することも出来る)．それが正確に結果を出せば，次のクラスに移動する. このステップは, 機械翻訳を始め, 従来体系化が困難であった NLP システム開発のための汎用的な方法論を提供するものと考えている.

5. おわりに

単語の拡張概念でもあるモジュラー構文構造が構文解析から抽出できることを示した．前提条件として，構文解析が正しく出来ていることが挙げられるが，逆にモジュラー構文構造から得られる拡張単語辞書を形態素解析に適用すれば，構文解析の性能向上につながる．これは，モジュラー構文構造が自然言語の重要な言語資源であることを示している．今後，モジュラー構文構造に対して多くの研究が試みられ，NLP の品質向上や新たな NLP サービスが実現されることを期待したい.

謝辞　本研究の機会を与えていただいた財団法人日本特許情報機構特許情報研究所守屋敏道専務理事及び渡邊豊英部長, 実装するためのモデルとなった CDL.core についてご指導いただいた ISeC 横井俊夫理事に深く感謝します．また，KNP を利用させて頂いた京都大学黒橋研究室に感謝します.

参考文献

1) 黒橋禎夫：結構やるな，KNP，情報処理, Vol.41, No.11, pp.1215-1220 (2000)
2) 工藤拓，松本祐治：チャンクの段階適用による日本語係り受け解析，情報処理学会論文誌, Vol.43, No.6, pp.1834-1842 (2002)
3) ISeC 技術資料：CDL.core 仕様書 第 1 版, ISeC, http://www.instsec.org/CDLcoreSpecV1.pdf, (2007)
4) 熊野明，安原宏，渡邊豊英：産業日本語の構想と特許文の言い換え実験，情報処理学会研究報告,2009-NL-190, pp. 15-20 (2009).

付録 Juman，KNP の解析結果を変換した CDL 形式モジュラー構文構造出力例

{#s1 文 sent=<統計的機械翻訳では，第１の言語の文と第２の言語の文との対訳文を多数含む対訳コーパスを用いた学習により予め翻訳モデルを作成しておき，この翻訳モデルを用いて翻訳を行なう．>;

```
{#21  行なう          pos=<動詞>          fw=<．>;
  {#16    作成しておき        pos=<動詞>          fw=<，>;
    {#13  より        pos=<動詞>          fw=<>;
      {#12          学習        pos=<名詞>          fw=<に>;
        {#11        用いた      pos=<動詞>          fw=<>;
          {#0       統計的機械翻訳      pos=<名詞>          fw=<では，>; }
          [#0       デ格        #11]
          {#10      対訳コーパス        pos=<名詞>          fw=<を>;
            {#9     含む        pos=<動詞>          fw=<>;
              {#7   対訳文      pos=<名詞>          fw=<を>;
                {#6             文          pos=<名詞>          fw=<との>;
                  {#3           文          pos=<名詞>          fw=<と>;
                    {#2         言語        pos=<名詞>          fw=<の>;
                      {#1       第１        pos=<名詞>          fw=<の>; }
                      [#1       ノ格        #2]}
                    [#2         ノ格        #3]}
                  [#3           ト格        #6]
                  {#5           言語        pos=<名詞>          fw=<の>;
                    {#4         第２        pos=<名詞>          fw=<の>; }
                    [#4         ノ格        #5]}
                  [#5           ノ格        #6]}
                [#6             連体        #7]}
              [#7   ヲ格        #9]
              {#8   多数        pos=<名詞>          fw=<>; }
              [#8   隣接        #9]}
            [#9     連格        #10]}
          [#10      ヲ格        #11]}
        [#11        連格        #12]}
      [#12          隣接        #13]}
    [#13  連用        #16]
    {#14  予め        pos=<副詞>          fw=<>; }
    [#14  連用        #16]
    {#15  翻訳モデル          pos=<名詞>          fw=<を>; }
    [#15  ヲ格        #16]}
  [#16    連用        #21]
  {#19    用いて      pos=<動詞>          fw=<>;
    {#18  翻訳モデル          pos=<名詞>          fw=<を>;
      {#17          この        pos=<指示詞>        fw=<>; }
      [#17          連体        #18]}
    [#18  ヲ格        #19]}
  [#19    連用        #21]
  {#20    翻訳        pos=<名詞>          fw=<を>; }
  [#20    ヲ格        #21]}
}
```

上記文中の MSF2 の例：「この翻訳モデルを用い」

```
{#19  用いて        pos=<動詞>          fw=<>;
{#18  翻訳モデル    pos=<名詞>          fw=<を>;
{#17  この          pos=<指示詞>        fw=<>; }
[#17  連体          #18]}
[#18  ヲ格          #19]}
```

上記文中の MSF8 の例：「第１の言語の文と第２の言語の文との対訳文を多数含む」

```
{#9   含む          pos=<動詞>          fw=<>;
{#7   対訳文        pos=<名詞>          fw=<を>;
        ・・・・・・・・・・・・・・・・・・・・・・・・
{#8   多数          pos=<名詞>          fw=<>; }
[#8   隣接          #9]}
```