知識ベース増殖のための中央データベース

西野 文人†, 杉山 健司†, 辻井 潤一‡
†{nishino,kenji}@edr.co.jp ‡ tsujii@is.s.u-tokyo.ac.jp
† (株) 日本電子化辞書研究所 ‡ 東京大学

1 はじめに

我々はコーパスに対してシンタグマティックな関係（横につながっている語がどのようにまとめられ，互いにどんな関係にあるか）とパラダイマティックな関係（互いに排除的である集合の中からどのような要素を排除することによって用いられているか）を解析し，辞書や知識ベースのような外部知識とともにこれらの解析で得られた結果も利用してさらに処理を繰り返すことで，それぞれの語に関する知識や，語と語の関係の知識などの言語的知識を獲得・増殖することを考えている．それぞれの言語処理ルーチンがお互いの処理結果を利用するためには，各ルーチン間のインタフェースとなる形式を定め，各ルーチンの処理結果を中央データベースに蓄えていく必要がある（図1）．

本稿ではまず中央データベースの考え方を述べ，次に具体的に形態素解析結果や構文解析結果などをどのような形式で格納しようとしているのを示し，そして内部構造案を示す．

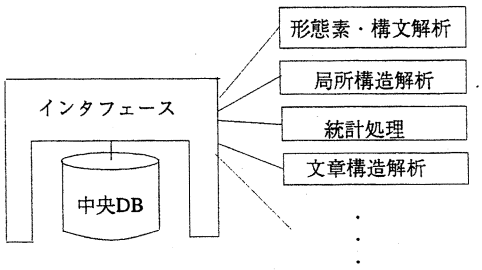

図 1: 中央データベース

2 中央データベースとは

言語知識を獲得するには大量のコーパスが必要となる．コーパスとしては原文そのままのタグなしコーパスのほかに，形態素解析や意味解析をした結果を括弧やタグなどで示したタグつきコーパスが提供されている [1]（EDR コーパス [2]，RWC コーパス [3] など）．しかしこのような言語的解析を施した高品質な（解析結果のレベルが統一されており，誤りがない）コーパスを作成しようとすると，機械的な処理が作成過程にあったとしても，最終的には人間のチェックが必要不可欠である．その結果コーパス作成には手間もかかり，コーパスの量もどうしても少ないものとならざるを得ない（それでもまだ現実には解析方針のゆれを含めコーパスには多くの誤りが含まれている）．

ここで一つの大きな疑問が湧いてくる．それは，はたしてコーパスの解析レベルは統一されていて誤りのないものでなければならないのだろうかというものである．すなわち，コーパスの解析レベルは必要ならば論理的レベル（フィルタを通して利用者が認識するレベル）で統一すればいいのであって，物理的なレベルでは統一されていなくてもいいのではないだろうか．また人間がチェックしたところで，誤りは皆無とはいかないのだから，誤りが多いかもしれないが機械的処理そのままのものをコーパスとしても構わないのではないだろうか．ただし，誤りがどのくらい含まれている可能性があるのかという信頼度のようなものを個々の解析結果に対して付与しておくというのは，コーパスに基づく処理の信頼性を向上させるためには必要なことであろう．また逆にこのような誤りを含むコーパスは，誤りの例を解析したり，言語処理システムの性能を比較するには役立つものとなるであろう．このような考え方に基づくと，コーパスは，原文のままのものから，形態素に分割しただけのものや，意味構造まで解析し

たものまであらゆる種類の解析レベルのものを含み，解析の信頼度としても，機械処理をした結果そのままのものから，人間によってチェックされたものまであらゆるレベルのものを含むことになる．これはすなわち，あらゆる言語処理ルーチンは何らかのレベルのコーパスを出発点として処理するものであり，これらのルーチンによって処理された結果もまたコーパスとなることを意味している．これはまさしく原コーパスから出発して各種の処理が施され，その処理結果として得られた知識がコーパスに反映され，その知識をもとにまた新たに各種の処理が行なわれ，知識が増殖していくというループであり，コーパスとはそのための中央知識データベースなのである．

さて，このようなコーパスはどのようにあるべきだろうか．ここでコーパスに要求されることは，色々なレベルの知識を混在して記述でき，様々な言語処理のインタフェースになり得るものである．我々はこの要求を満たすものとして独自のインタフェース言語を設計するのではなく，なるべく多くの人が受け入れやすいように，各種の文書記述の標準化を狙ったものとして著名なTEI(Trxt Encoding Initiative)が提唱しているSGMLのDTD（文書構造定義体）に沿った形式[4]で記述することにした．

3 コーパス構造例

ここではコーパスをTEI形式にするとどのようなものになるかを示す．図2はEDRコーパスの形態素情報をTEI形式に変換したものである．ここでは各形態素を<m>タグで示し，品詞，読み，対応概念を属性として示している．実際の概念を知るためには外部知識としてEDR概念辞書とのリンクを図ってやる必要がある．また各タグにユニークなidをつけることで他からの参照が可能なようにしている．図3は同じく現在のEDRコーパスにある構文情報をTEI形式で記述したものであり，図4は意味情報をTEI形式で記述したものである．我々は現在のEDRコーパスをTEI形式に変換することを考えている．

現在，EDRで進めている「知識ベース増殖のためのソフトウェアプロジェクト」[1]では，色々な自然言語処理ルーチンを開発して，プロジェクト終了後にはフリーソフトとして提供する予定である．ここで開発するソフトウェアを使ってコーパスに日付，場所，組織名などのタグを付与することも考えている（図5）．

[1]これはIPAの創造的ソフトウェアプロジェクトとして進めているものである

```
<text id=text006000000c8b_s>
 <m id=m1 type='名詞' orth='ＩＴＯＳ'
      concept='"＝Ｚ　１９７０年代"'>ＩＴＯＳ</m>
 <m id=m2 type='名詞' orth='ファッション'
      concept='1065e4'>ファッション</m>
 <m id=m3 type='助詞' orth='ガ'
      concept='2621d5'>が</m>
 <m id=m4 type='名詞' orth='マチ'
      concept='3cfc38'>街</m>
 <m id=m5 type='助詞' orth='ニ'
      concept='2621d5'>に</m>
 <m id=m6 type='動詞' orth='モド'
      concept='3ce6a4'>戻</m>
 <m id=m7 type='語尾' orth='リ'
      concept='2621cc'>り</m>
 <m id=m8 type='動詞' orth='ハジメ'
      concept='1e8aa5'>はじめ</m>
 <m id=m9 type='助詞' orth='テ'
      concept='2621d5'>て</m>
 <m id=m10 type='動詞' orth='イ'
      concept='0e52dc'>い</m>
 <m id=m11 type='語尾' orth='ル'
      concept='2621d0'>る</m>
 <m id=m12 type='記号' orth='。'
      concept='2621d8'>。</m>
</text>
```

図 2: 形態素情報

```
<phr type='+'>
 <phr function=main type='kakari'>
  <phr type='+'>
   <phr function=main type='kakari'>
    <m id=m1>ＩＴＯＳ</m>
    <m function=main id=m2>ファッション</m></phr>
    <m id=m3>が</m></phr>
   <phr function=main type='kakari'>
   <phr type='+'>
    <m function=main id=m4>街</m>
    <m id=m5>に</m></phr>
   <phr function=main type='+'>
    <m function=main id=m6>戻</m>
    <m id=m7>り</m>
    <m id=m8>はじめ</m>
    <m id=m9>て</m>
    <m id=m10>い</m>
    <m id=m11>る</m></phr></phr></phr>
 <m id=m12>。</m>
</phr>
```

図 3: 構文情報

```
<fs type=node>
 <f name=main id=c6><str>戻</str></f>
 <f name=attribute>
    <sym value='present state begin'></f>
 <f name=goal id=c4><str>街</str></f>
 <f name=a-object>
  <fs type=node>
   <f name=main id=c2>
      <str>ファッション</str></f>
   <f name=time id=c1>
      <str>´７０Ｓ</str></f>
  </fs></f>
</fs>
```

図 4: 意味情報

```
<num type=cardinal value='123'>百二十三</num>
<date value='28-3-1997'>1997年3月28日</date>
<rs type=person>小林さん</rs>は<rs type=place>
小山</rs>にある<rs type=person>彼</rs>の家から
<rs type=org>EDR</rs>に通勤している.
```

図 5: 局所構造解析結果

さて，このような言語処理ルーチンが出力する結果にはあいまい性が残ることがある．したがって，あいまい性を表現できることも必要である．また，処理結果として付与されたタグの信頼度はどの程度なのか，それはどのルーチン（誰）によって付与されたものかを明示しておくことも必要であろう．これらの情報は，コーパスからデータを抽出する際に，抽出するデータを信頼度に応じて制御するなどの際に役立つであろう（図 6）．さらに，使用されている単語に関する統計処理をとる場合や，キーワード検索を行なう場合などでは，表記のゆれを標準化する処理が必要になる．また，原文中に誤りがあった場合にも修正をしておきたい場合がある．しかし校正支援システムのような研究では誤りデータそのものが貴重であり，誤りデータも正解データと同様に保存しておくことも考慮すべきであろう（図 7）．このようにコーパスには様々な処理結果を保存しておくことを考えている．

4 コーパスの内部構造とインタフェース

コーパスは SGML(TEI) 形式で表現されるので，すべてテキストデータとして保持することが可能である．すなわちコーパスと各種の言語処理とのインタフェースを SGML 形式のテキスト表現とすることができる．しかし，コーパス中には様々なレベルの知識が渾然一体となって含まれているので，それぞれの言語処理ルーチンがコーパスの SGML 形式から自分に必要な部分だけを取り出して，処理した結果得られた知識を SGML 形式にして出力するのでは，各言語処理ルーチンに負担がかかり過ぎる．そこで各言語処理ルーチンとコーパスとのインタフェースとしてコーパス管理ルーチンを用意する．各言語処理ルーチンからは自分の処理に必要な知識（タグセット）を要求し，現実のコーパスの物理的な記述がどうなっていようが，言語処理ルーチン側からコーパスをながめた時にコーパスの見え方が一定であるように，コーパス管理ルーチンが要求に応じてコーパスを変形して各言語処理ルーチンとのコミュニケーションをとる．すなわち現実には存在してもそのツールから見れば全く不要なタグは付与されていないように見せ，逆に要求元の言語処理ルーチンが仮定しているタグは実際には付与されていなくても必要ならタグ付与ルーチンを呼び出すことで付与して要求元に見せるようにする．そのためには，各言語処理ルーチンがどのような知識を必要とし，またどのよ

```
<s>今度の
  <name type=person id=family
        exclude=city  >川崎</name>
  <name type=place  id=city
        exclude=family>川崎</name>
市長</s>
<!-- ... -->
<altGrp targType='name name'>
  <alt mode=excl targets='family city'
        weights='20 80'>
</altGrp>

今度の<placeName id=p1>川崎</>市長
...
<certainty target=p1 locus='#gi' degree='0.6'>
<certainty target=p1 locus='#gi' degree='0.4'
    assertedValue='persName'>
<respons target=p1
    locus='#gi #location' resp=MOR>
```

図 6: あいまい性，信頼度，責任者

```
袖振り合うも<reg orig='他生'>多生</reg>の縁
```

図 7: 校正結果

うな知識を供給するのかを管理する．これにより，要求された知識がコーパス内にないときは，その知識を得るためにそれに適した言語処理ルーチンを呼び出すことができる．これらによって，それぞれの言語処理ルーチンからはコーパスは論理的に統一したレベルで捉えることができるのである．

以上からコーパス管理ルーチンを通して，毎回コーパスから要求された形式への変形が必要になってくる．したがってコーパス管理ルーチンが管理するコーパスが，単純にTEI構造をそのまま文字列にしておいて毎回SGMLのタグ構造を解析するというものでは非効率的である．要求された知識のみを取り出すことを効率良く行なえるような内部構造にしておき，必要に応じて純粋な文字列構造との相互変換を行なえるようにしておくことが望ましい．

さて，各種の言語処理とのインタフェースとしてのコーパスデータベース，つまりSGML(TEI)構造データベースに要求される機能はどのようなものであろうか．それを考えるために，まずコーパスデータベースの使われ方を検討してみよう．まず原文そのものは，ほとんど書き換えられないものと考えてよい．その代わり，各種の言語処理によって，頻繁に新しい知識，すなわちタグの付与が行なわれる．また，各種の言語処理ルーチンから見た論理的な知識構造を統一するために，付与されているタグの論理的な削除（言語処理ルーチンにはそのタグが付与されていないように見せかける）が頻繁に行なわれる．また，付与されるタグはIDを保有し個別に参照されることが多い．さらに外部の知識ベース（言語辞書）への参照も行なわれる．このような要求を満足させるための内部構造として我々は次のような形式を採用した．まずタグのまったくない原文をタグ部とは分離してそのまま保持する．そしてタグ部には付与されているタグ名と原文上のどこにオープンタグおよびクローズタグが付与されるのかの位置情報を与える．またタグの入れ子構造を明確に表現するために各タグに対して，タグ構造の自分の妹と娘のタグへのポインタを持たせる．例えば，`<p>a<q>b</q><r>c</r></p>` というSGML構造に対しては，図8に示すような構造を持たせる（図8では属性に関しては省略して示してある）．

このような構造を持たせることで，新しい知識（タグ）の付与に対して原文部の書き換えは起こらず，タグ部へのレコードの追加だけですませることができる．またタグつきテキストとしての出力はタグ部の構造を娘リンクと妹リンクを順にたどることによって行なえ，指定されたタグ群のみの出力は個々でタグ名をチェックすることで容易に処理することができるのである．

id	タグ名	開始位置	終了位置	妹	娘
0	p	0	3	NULL	1
1	q	1	2	2	NULL
2	r	2	3	NULL	NULL

図 8: 内部構造

5 おわりに

本稿では各種の言語処理ルーチンのインタフェースとしての中央データベースの構想を示した．そして具体的なTEI形式による記述案を示し，論理的インタフェースとしてのコーパスデータ管理部と効率的なアクセスのためのコーパス内部表現形式について述べた．我々はこのような技術をオープンにしていき標準化の叩き台にしたいと考えている．

EDRでは，各種言語処理によって知識を増殖させるためのソフトウェア群の作成を行なっているが，従来のEDRコーパスを本稿で示した形式に変換するとともに，実際の言語処理ルーチンの処理結果もコーパスとして蓄えていきたいと考えている．このようなコーパスは，新たに言語処理ルーチンを作成したい研究開発者を，周辺の言語処理ルーチンの開発から解放することになると期待している．本稿では形式的な構造についてのみ述べたが，各種の言語処理ルーチンのインタフェースをとるためには品詞をはじめとする内容の見直しも必要である．この品詞体系の考察については[5]を参照されたい．

参考文献

[1] 電子協：自然言語処理システムの動向に関する調査報告書 (1995).

[2] 日本電子化辞書研究所：EDR電子化辞書説明書, 第二版 (1995), http://www.iijnet.or.jp/edr より入手可能.

[3] 技術研究組合 新情報処理開発機構 データベースワークショップテキストグループ：テキストデータベース報告書 (1995).

[4] C.M.Sperberg-McQueen, and Burnard, L. eds.: *Guidelines for Electronic Text Encoding and Interchange* (1994).

[5] 出羽達也, 西野文人, 辻井潤一：知識獲得に向けての品詞体系の考察, 言語処理学会第3回年次大会, D2-1 (1997).