语法开发平台的关键技术实现*

化柏林 王惠临

（山东理工大学科技信息研究所 淄博 255049）（中国科学技术信息研究所 北京 100038）

【摘要】 对语法开发平台中算法性很强的功能模块进行了算法剖析，包括句法结构线性表达的分析算法，规则与词典的提取算法，成分结构树图与功能结构集图的生成算法，成分结构与功能结构的转换算法。最后对系统作了简单的功能测试。

【关键词】 语法开发平台 语法提取 关键技术 核心算法 【分类号】 TP311 H043

Key Techniques of Grammar Development Platform

Hua Bolin

(*Institute of Scientific and Technical Information of Shandong University of Technology, Zibo 255049, China*)

Wang Huilin

(*Institute of Scientific and Technical Information of China, Beijing 100038, China*)

【**Abstract**】 This paper anatomizes some modules of Grammar Development Platform, which are characterized by complex algorithm. It introduces four functional modules, and concretely analyses algorithm of syntactic structure linear presentation, extraction algorithm of rule and lexicon, image creating algorithm of constituent structure tree and functional structure set, converting algorithm of constituent structure and functional structure. At last, it presents a simple test of grammar extraction subsystem.

【**Keywords**】 Grammar development platform Grammar extraction Key techniques Core algorithm

1 引 言

语法主要包括规则与词典。语法开发平台（Grammar Development Platform, GDP）主要是提供语法开发的一个工作平台。它首先提供一个手工开发语法的工作平台，在此基础上，对开发的语法进行词典与规则的统计与学习，然后构造自动分析算法，对大规模文本进行分析并验证，得到一部实用的语法。

平台的具体算法仅仅是以 LFG（Lexical Functional Grammer）为基本参考模型，并不完全依赖于 LFG 模型，本身并不依赖于语法表达的属性和值，只识别语法表达中的标记符号，对结点内容不做判断与分析。由于平台提供二次开发功能，因此数据是共享的，可以直接为其它上层应用系统所使用；平台的核心算法采用 Java 编程，前台展示采用 JSP，因此既可以运行于 Windows 平台，又可以运行于 Linux 或 Unix 平台上；数据库管理系统采用 Oracle 8.1.7，数据接口采用 Thin 方式。

国外许多著名大学和 IT 行业的巨头都纷纷投入到自然语言处理的研发工作中。他们纷纷建立语法开发平台并开始生产大规模语法。按功能分类有：单一模型的语法开发平台，有 Xerox 公司的 XLE[1]、XLFG[2]、宾西法尼亚大学的 XTAG[3,4]、斯坦福大学的 LinGO[5] 和 LKB[6]；多模型的语法开发平台，有微软研究院的大规模多语种开发平台[7]和德国的 GTU[8,9]；模型转换项目有辻井实验室的 XHPSG[10,11]等。

本语法开发平台由三个子系统构成：语法提取子系统、语法分析子系统、语义分析处理子系统。语法提取子系统是整个语法开发平台的基础，语法分析子系统是语法开发平台的核心，而语义分析处理子系统是语法开发平台的延伸与扩展。语法提取子系统是从句法结构中提取语法，是一个半自动化的过程。

2 句法结构线性表达的分析算法

对于句法结构线性表达，根据标记符号来分析线性

收稿日期：2005－08－29

*本文系国家自然科学基金项目“面向自然语言处理的逻辑语义表达与演算模型研究”（项目批准号：60173025）研究成果之一。

表达中的信息,包括各种各样的结点信息和句子的自然表达。主要包括成分结构的线性表达和功能结构的线性表达。

2.1 成分结构线性表达的分析算法

成分结构的线性表达有以下约束和规律:

不同层之间的结点用圆括号来表示,同层之间的结点用逗号来隔开。叶子结点的词项用尖括号来表示。形式合格的句子须是括号完全匹配,而且除 S 根结点外全封闭。

两个左括号之间肯定有结点,但两个右括号之间却不一定。也就是说不会有两个紧相邻的左括号,但会有两个或多个相邻的右括号。

对任意的句子 S,叶子结点 N_l 的个数等于标记逗号 C_f 的个数加 1,其中标记逗号的个数等于所有逗号 C_a 的个数减去句子自然表达中的逗号 C_n。因此 $\forall S$,

$$\sum N_l = \sum C_f + 1 \tag{1}$$

计算时采用

$$\sum N_l = \sum C_a - \sum C_n + 1 \tag{2}$$

对任意的句子 S,所有结点 N_a 个数等于左括号 B_{fl} 个数加上标记逗号个数加 1,也就等于叶子结点个数加上左括号的个数,其中左括号的个数不包括句子自然表达中的左括号 B_n。因此 $\forall S$,

$$\sum N_a = \sum C_f + \sum B_{fl} + 1 \tag{3}$$

计算时采用

$$\sum N_a = \sum C_a - \sum C_n + \sum B_{fl} - \sum B_{nl} + 1 \tag{4}$$

把公式(1)代入公式(3),我们可以得到公式(5)

$$\sum N_a = \sum N_l + \sum B_{fl} \tag{5}$$

结点深度:遇到左括号,结点的深度加 1,遇到右括号,结点的深度减 1;如果结点中没有右括号,深度为前面左括号的个数减去右括号的个数;如果初始结点中含有右括号,深度要减去结点中右括号的个数。

值:第一个结点取到第一个左括号,最后一个结点取到第一个右括号,其余结点为两个分隔符之间去掉右括号的值。

父子关系:求父子关系可以有两种方式,根据括号求父子关系和根据结点深度求父子关系。左括号决定孩子结点的分支开始,与其左括号相匹配的右括号为孩子结点的终结。或者是从本结点开始向后到第一个与本结点深度相同的结点为止,所有深度比本结点深度大 1 的结点就为孩子结点。

```
for (int i =0;i < iNodeCount;i + + ) {
  node = (Node) list. get(i);
  List listChild = new java. util. ArrayList( );
  int childCount =0;
  for(int j = i + 1;j < iNodeCount;j + + ) {
    Node nodeChild = new Node( );
    nodeChild = (Node) list. get(j);
    if (nodeChild. getLevel( ) = = node. getLevel( )) {
      break;
    } else if(nodeChild. getLevel( ) = = (node. getLevel( ) + 1)) {
      childCount = childCount + 1;
      listChild. add(nodeChild);
    }
  }
  node. setChildNode(listChild);
  node. setChildNo(childCount);
}
```

2.2 功能结构线性表达的分析算法

功能结构线性表达的分析算法和成分结构线性表达的算法基本相似,只是标记不一样罢了。成分结构的线性表达用圆括号和逗号作为分隔符来标记结点,左括号和右括号来决定结点的深度和结点之间的层次关系,句子的自然表达,也就是词项是用尖括号标记出来的。功能结构的线性表达用中括号和分号作为分隔符标记结点,通过左中括号与右中括号来决定结点的深度与结点之间的关系,叶子结点的值通过单引号来标记。

上面介绍的只是标记形式的不同。其实功能结构表达与成分结构表达最大的不同就是功能结构表达采用属性值对,属性与值之间通过等号来连接,如“PRED = 'believe < SUBJ, COMP > '”,而成分结构表达采用的只是标记本身,如“NP”。

为了后面作图的方便,有几个值要在此求出。在结点类(entity. Node. java)里增加变量 iChildrenNo 表示多层孩子叶子结点的个数,也就是子孙叶子结点的个数,用于画功能结构竖线的高度。iChildrenLevel 表示子孙结点的辈数,也就是结点后面有几代,iChildrenMaxLength 表示子孙结点中字符数最长的长度,iChildrenLevel 和 iChildrenMaxLength 共同作用来决定功能结构右闭线的水平位置。

iChildrenNo 求结点的子孙结点个数,从右到左自底向上进行计算。每个结点的子孙结点个数等于所有子结点的子孙结点个数的和,是一个迭代计算。如果子结点的子孙结点个数不为零,那么子孙结点个数等于子孙结点个数加上子结点的子孙结点个数;否则如果子结点的子结点个数不为零,那么子孙结点个数等于子孙结点个数加上子结点的子结点个数,否则子孙结点个数直接加 1,对结点的每一个子结点,经过以上三种情况的判断,通过累加,就得到结点的子孙结点个数。

```
if (nodeChild. getChildrenNo( ) ! =0) {
  childCount = childCount + nodeChild. getChildrenNo( );
```

```
}else if(nodeChild.getChildNo()! =0){
  childCount = childCount + nodeChild.getChildNo();
}else{
  childCount = childCount + 1;
}
```

iChildrenLevel 求子孙结点的辈数,从右到左自底向上进行计算。如果子结点的子孙辈数不为零并且大于或等于本结点的子孙辈数时,本结点的辈数加 1。对结点的每一个子结点,进行以上判断处理,通过累加,就得到结点的子孙辈数。

```
if (nodeChild.getChildrenLevel() >0 &&
nodeChild.getChildrenLevel() > =node.getChildrenLevel()){
  childLevel = nodeChild.getChildrenLevel() +1;
}
```

iChildrenMaxLength 求子孙结点中最大子孙长度,从右到左自底向上进行计算。对于每一个子结点,如果子结点的子孙最大长度加上子结点的长度大于本结点的最大子孙长度,那么本结点最大子孙长度就等于子结点的最大子孙长度加上子结点的长度。这样计算出来的最大子孙长度,不仅仅是结点值的字符长度,还包括结点的深度累加值,也就是说如果我们把每一个结构的字符长度看成权重,那么求出来的是最长路径。对结点的每一个子结点,进行以上判断处理,通过累加,就得到结点的最大子孙长度。

```
if (nodeChild.getChildrenMaxLength() + nodeChild.getValue().
    length() >node.getChildrenMaxLength()){
childMaxLength = nodeChild.getChildrenMaxLength() + nodeChild.
    getValue().length();
}
```

3 规则与词典的提取算法

对于规则与词典,有无注释项,其分析算法不一样。不带注释的规则称为简单规则,不带注释项的词典称为简单词典。词典的注释项的提取要依赖于具体的 LFG 范例,因此本文没有对其进行提取。

3.1 简单词典的提取

同样,通过调用线性句子的分析算法,得到句子的结点序列。对于每一个叶子结点,尖括号内的串正是单词的值,左尖括号左边的值为单词的种类。把这两项信息提取出来,插入数据库。

```
sLexicon = node.getValue();
sLexiconSpec = sLexicon.substring(0,sLexicon.indexOf("<"));
sLexiconValue = sLexicon.substring(sLexicon.indexOf("<") +1,
    sLexicon.indexOf(">"));
```

如果该词已经存在,则更新词频和句子序列,词频加 1,句子序列加上句子的序号;如果词不存在,则直接插入,词频设为 1,句子序列设为该句子的序号。

如果没有重复,对于任意一个句子 S,则从每个句子中抽出词典 L 的最大条数为 N,N 为句子表达的叶子结点个数。∀S 有∑L < =N。

3.2 简单规则的提取

通过线性句子表达的分析之后,就得到句子的所有结点序列。提取规则并插入到数据库中,对于含有叶子结点的成分要把尖括号以及尖括号内的单词滤掉。如果结点值中含有尖括号,那就把尖括号内的内容给滤掉,这是句子的自然表达结点。如果结点值中没有尖括号,说明不是叶子结点,提出它作为规则的左部;然后对该结点的所有子结点进行分析,如果是叶子结点,就滤掉尖括号的内容,添加到规则的右部,生成 SQL 语句,插入到数据库。每个结点的子结点个数就等于规则右部项的个数。

```
sSqlRule = ((Node)listRule.get(0)).getValue();
if (sSqlRule.indexOf("<") >0){
  sSqlRule = sSqlRule.substring(0,sSqlRule.indexOf("<"));
}
for (int k =1;k <listRule.size();k + +){
  sChildNode = ((Node)listRule.get(k)).getValue();
  if (sChildNode.indexOf("<") >0){
    sChildNode = sChildNode.substring(0,sChildNode.indexOf("
        <"));
  }
  sSqlRule = sSqlRule + "," + sChildNode;
}
```

如果规则已经存在,则更新规则的频率和句子序列,出现频率加 1,句子序列添加上该句子的序号;如果不存在,则将其插入到数据库里,出现频率设为 1,句子序列设为句子的序号。

如果没有重复,对于任意的一个句子 S,则从每个句子中抽出规则 R 的最大条数为 M,M 为有两个及以上子结点的结点总数。∀S 有∑R < =M。

3.3 规则注释的提取

规则的注释比较复杂,但一般来讲,对于非叶子结点,其注释一般只有一项,通常是描述句子结点的关系。对于树状结构图来讲,层次结构树可以反映结点在句子中的位置,每一个结点值只是反映一个离散的结点信息,结点与结点之间的连线可以反映结点与结点之间有关系,但有什么关系并反映不出来。注释的功能就是表达这种关系。某一 NP 的注释为(↑SUBJ) = ↓,其意思是说 NP 的父结点的主语是本结点,换句话说,NP 这一结点是上一结点的主语。

如果子结点个数大于零,就需要为结点抽取注释信息,当然根结点除外。

```
if (childCount >0 && i >0){
  node. setNote("(↑" + node. getValue( ). substring( node. getVal-
    ue( ). indexOf( ":") + 1, node. getValue ( ). length ( ) - 1 )
    +") = ↓");
}
```

4 图形的生成算法

本平台是通过算法自动生成图形,通过创建 Image 对象,直接生成 JPEG 编码的图形文件,调用了 java. awt. image 和 com. sun. image. codec. jpeg 两个 Java 包。其过程是首先创建 Image 对象,然后创建 Graphics 对象,然后设置图形的底色,根据数据来计算图形的大小,即宽和高,最后填充矩形,这样 Image 对象就准备好了,可以在上进行图形的自动绘制了。当图形绘制完毕后,释放图 Graphics 的环境,从服务器返回输出流,进行 JPEG 编码,完成整个图形的创建过程。

从数据结构上讲,成分结构与功能结构都是标准的多叉树。从图形上讲,成分结构是标准的多叉树,是下端开放的;而功能结构却不是标准的多叉树,其末端是闭合的。

从图形表示来看,成分结构的结点之间用直线,叶子结点(词项)用虚线来表示。横向:叶子结点等间距排列,其它层中,结点位于最左孩子与最右孩子间距的中垂线上;纵向:层与层之间等间距。

4.1 成分结构树图的生成过程

成分结构树图的生成有以下三步:首先计算每个结点的水平位置和垂直位置,然后在相应的位置画出结点值,最后画出结点之间的连线。

(1) 计算每个结点的水平位置和垂直位置,也就是 left 与 top。从最右下角的结点(结点序号最大)开始,如果是叶子结点(孩子结点数为零),那么该结点的水平位置就是:

$$node.left = width - leafCount * hSpace + (hSpace - node.Width)/2 \quad (6)$$

其中 width 为图形宽度,hSpace 为两个相邻结点之间的水平宽度,hSpace/2 代表要留出图形边界,leafCount 为结点在所有叶子结点的序号,(hSpace - node. Width)/2 来确保结点居中显示。如果是自底向上方式,结点的垂直位置等于树的深度乘以垂直间距,即:

$$node.top = tree.Depth * vSpace \quad (7)$$

否则结点的垂直位置等于结点的深度乘以垂直间距,即:

$$node.top = node.Level * vSpace \quad (8)$$

如果不是叶子结点,那么结点的水平位置等于最左子树的水平位置加上最右子树与最左子树的水平位置差再减去结点长度的一半,即:

$$node.left = nodeLeft.left + (nodeRight.left - nodeLeft.left)/2 - node.width/2 \quad (9)$$

垂直位置等于结点的深度乘以垂直间距,同公式(8)。

(2) 在相应的位置输出结点值。由于位置都确定了,就可以直接用 drawString( )输出。

(3) 画出结点之间的连线。线的起始水平位置等于源结点的左位置加上源结点宽度的一半,起始垂直位置等于源结点的垂直位置加上源结点的高度;线的终止水平位置等于目标结点的左位置加上目标结点宽度的一半,终止垂直位置等于目标结点的垂直位置。

4.2 功能结构集图的生成过程

功能结构集图的生成过程有以下四步:首先计算每个结点的水平位置和垂直位置,然后在相应的位置画出结点值,接下来画出结点的左开线,最后画出结点的右闭线。

(1) 计算每个结点的水平位置和垂直位置。结点的水平位置等于结点的深度乘以水平间距,同公式(6);垂直位置等于前面叶子结点的个数加上一个垂直间距。

(2) 在相应的位置输出结点值。把所有的等号换成空格,输出结点值即可。

(3) 画出结点的左开线。只有子结点数不为零的结点才需要画线。画一条竖线和两条短横线,竖线的起始水平位置等于结点的左位置加上一个水平间距,起始垂直位置等于结点的垂直位置;竖线的终止水平位置与起始水平位置相同,终止垂直位置等于结点的垂直位置加上结点的所有叶子结点数乘以垂直间距。竖线的两端画两条短横线。

(4)画出结点的右闭线。左开线同右闭线是轴对称的,因此只要确定这个轴的水平位置就可以了。对称轴的水平位置等于结点的后续辈数减去 2 再乘以水平间距,再加上所有子孙结点中最宽的结点宽度(不仅仅是结点值的字符长度,还包括结点的深度累加值),最后加上左开线的水平位置就可以了。右闭线的水平位置等于对称轴的水平位置减去左开线的水平位置再加上对称轴的水平位置,即:

```
AxisLine. left = ( node. ChildrenLevel - 2 ) * hSpace + node. Children-
    MaxLength + leftLine. left;
RightLine. Left = Axis. Left - LeftLine. Left + Axis. Left;
```

在功能结构集图中,对于不同深度的结点,对称轴的水平位置是不同的,对于深度相同的结点,其对称轴也不相同。图形为了美观,还需要对其具体位置做一些微调。在成分结构的树图中,线与文本可以紧密相连,而功能结构中的线与文本不能紧密相连,要留有适当的空隙。

对于带标注的成分结构树图,主要是在相应的结点下面画出结点注释,对于非叶子结点一般只有一个标注,保证位置居中,从本结点到子结点的连线的起始位置向

下移一个行距就可以了;对于叶子结点,注释一般都会有好几项,按居中方式往下排就可以了。

对于成分结构与功能结构映射图,由于成分结构线性表达与功能结构线性表达中含有映射函数,因此,分别画成分结构树图与功能结构集图,在有映射函数的两个图的结点之间画一条弧线进行映射。

5 成分结构与功能结构的转换算法

功能结构用"属性 - 值"的有序对来表示,同一结点下的属性是无序的。属性与值有一对一的关系,属性的值可以是子 f - 结构,但不可能是简单符号和子 f - 结构的集合。在进行计算时,不需要考虑结点内容,只要考虑子结点个数就行了。这样才能保证系统对数据的抽象处理能力,也就是保证平台的普遍适应性。

5.1 由简单成分结构与功能结构生成带标注的成分结构

提取规则注释,添加到成分结构树图非叶子结点的下面;提取词典注释,添加到成分结构叶子结点的下面,就构成了一个带标注的成分结构树图。因为平台不依赖于 LFG 范例,因此,一个功能结点下的属性值对的个数是不确定的,属性值的顺序也是不确定的,平台本身与结点值是无关的,对于词典注释的抽取是困难的。当实施具体的语法工程时,针对具体的范例就可以轻松地抽取词典注释了。

例如 OBJ = [SPEC = A; NUM = SG; MODI = 'satisfying'; LEX = 'answer'],如果不依赖于具体的 LFG 范例的话,我们是没法确定 NUM = SG 既可以是词 A 的注释,又可以是词 answer 的注释,而不是词 satisfying 的注释。生成带标注的成分结构如图 1 所示。

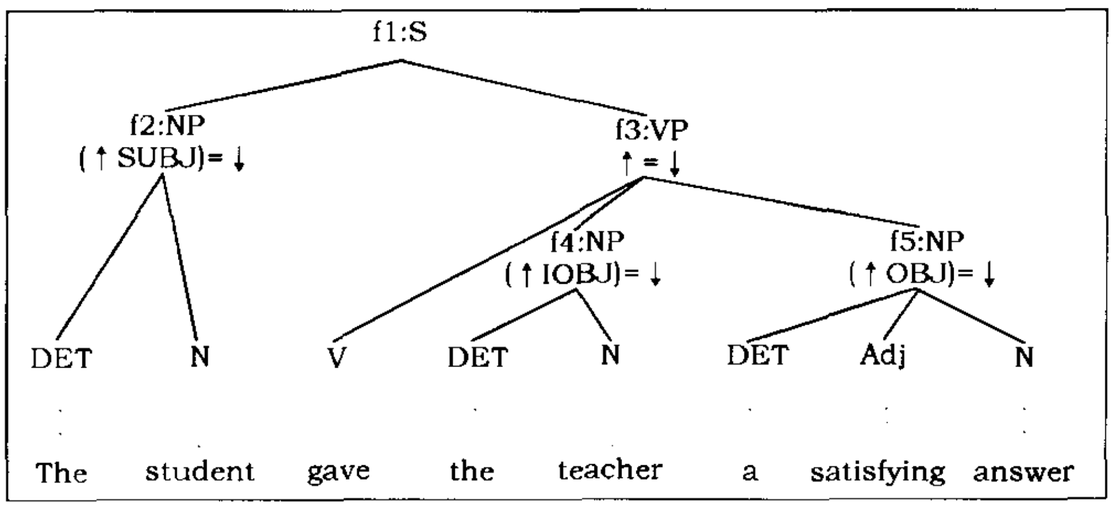

图 1 带标注的成分结构例图

5.2 由带标注的成分结构生成功能结构

首先由带标注的成分结构生成功能描述,如表 1 所示。

然后对相同的描述进行合一,就得到功能结构并生成框图,如图 2 所示。

表 1 功能描述例表

SenNo	DespNo	SrceNode	NoteCont	DestNode
S0001	01	f1	SUBJ	f2
S0001	02	f1	-	f3
S0001	03	f3	OBJ2	f4
S0001	04	f3	OBJ	F5
S0001	05	f2	SPEC	The
S0001	06	f2	NUM	SG
S0001	07	f2	NUM	SG
S0001	08	f2	LEX	'student'
S0001	09	f3	TENSE	PAST
S0001	10	f3	PRED	'gave < (↑ SUBJ) (↑ IOBJ) (↑ OBJ) >'
S0001	11	f4	SPEC	THE
S0001	12	f4	NUM	SG
S0001	13	f4	LEX	'teacher'
S0001	14	f5	SPEC	A
S0001	15	f5	NUM	SG
S0001	16	f5	NUM	SG
S0001	17	f5	MODI	satisfying
S0001	18	f5	LEX	'answer'

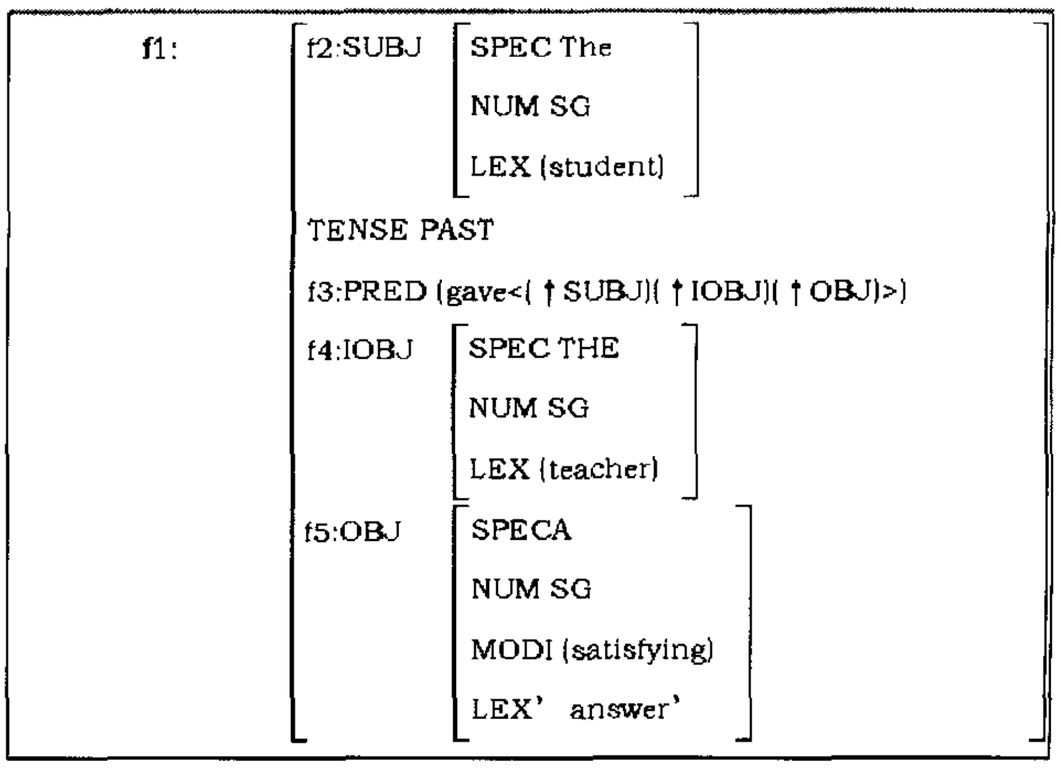

图 2 生成的功能结构例图

如果利用通用的抽象 LFG 模型而不依赖于具体的范例,从带标注的成分结构生成功能结构是可能的,而由功能结构生成带标注的成分结构是有困难的。特别是叶子结点,从底向上合一是容易的,而从上向下分解是困难的,主要原因是没法确定属性与值的归属,到底属于哪个词这样的信息是没有办法获取的。

6 测试数据的选取及测试结果

语法开发平台的语法提取子系统,因为没有正确率的问题,所以只需要测试,不需要验证。为了测试语法提取子系统,本文挑选了 17 个中文句子和 19 个英文句子,每个句子代表不同的句型。对这 36 个句子经过人工分

析,得出句子成分结构与功能结构的线性表达。把线性表达通过语法编辑器输入到系统内,然后自动进入语法提取子系统,进行语法提取,分别提取规则与词典。我们通过语法查询模块查看提取结果,一共提取了 171 条规则与 276 个词条,滤掉重复记录后,还有 94 条规则和 250 个词条。按照规则频率和词频倒排序可以看出规则与词典的分布情况。其具体情况如表 2 和表 3 所示。

表 2 高频规则与高频词表

规则	频率	词条	频率
S - > NP, VP	30	the	11
VP - > V, NP	22	a	10
NP - > DET, N	11	的、is、not	6
PP - > PREP, NP	9	You	5
NP - > DET, ADJ, N、VP - > V, V	7		
VP - > V, VP	6		
合计	85		32

表 3 低频规则与低频词统计表

出现频率	规则条数	百分比	词条数	百分比
1	63	67.0%	196	78.4%
2	18	19.1%	35	14.0%
3	2	2.1%	11	4.4%
4	3	3.2%	2	0.8%
合计	86	91.4%	244	97.6%

平均每个句子提取 4.75 条规则,其中 2.61 条是新规则,新规则率为 54.9%;每个句子提取 7.67 个词条,其中 6.94 条是新词,新词率为 90.5%。低频规则有 86 条,占 91.4%;低频词有 244 条,占 97.6%。这说明句子的选取很有覆盖性。

对于这 36 个句子,跟人工提取比较,结果完全一样,没有出现异常。因为不存在智能问题,所以只要程序没有 Bug,又能充分考虑到各种边界情况,理论上应该 100% 的正确,而实际也正是 100%。

语法开发平台是自然语言处理的一项基础性工作,它同样具有自然语言处理的特点:学科跨度大,开发周期长,开发难度高。语法开发平台也属于智能信息系统的范畴,它也具有智能信息系统的特点:理论难度大,系统准确性需要验证,除一般的数据库外,还需要各种模型库、方法库、知识库,涉及面比较宽广。作者会在以后的研究工作中沿着文中的设计思路继续进行后两个子系统的设计与实现,为应用系统的构建提供合适的平台。

参考文献:

1 Ronald M. Kaplan and Tracy Holloway King and John T. Maxwell III. Adapting ExistingGrammars: The XLE Experience. http://www2.parc.com/istl/members/thking/refs.html (Accessed Jun. 10, 2003)

2 Lionel Clément, Alexandra Kinyon. XLFG - an LFG parsing scheme for French. http://csli - publications. stanford. edu/LFG/6/lfg01clementkinyon.pdf (Accessed Aug. 24, 2003)

3 Christine Doran, Beth Ann Hockey, Anoop Sarkar, B. Srinivas and Fei Xia. Evolution of theXTAG System. http://www.sfu.ca/~anoop/papers/pdf/csli - version.pdf (Accessed Aug. 21, 2003)

4 Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas and Martin Zaidel. XTAG System - A Wide Coverage Grammar for English. http://acl.ldc.upenn.edu/C/C94/C94 - 2149.pdf (Accessed Sep. 25,2003)

5 Ann Copestake, Dan Flickinger. An open source grammar development environment andbroad - coverage English grammar using HPSG. http://www.cl.cam.ac.uk/~aac10/papers/lrec2000.pdf (Accessed Aug. 22,2003)

6 Ann Copestake The (new) LKB systemVersion 5.2 August 2000. http://www.linguist.jussieu.fr/~jtseng/lg035/manual.pdf (Accessed Sep. 26,2003)

7 Hisami Suzuki. A Development Environment for Large - scale Multi - lingual Parsing. http://acl.ldc.upenn.edu/coling2002/workshops/data/w06/w06 - 10.pdf (Accessed Aug. 22,2003)

8 Martin Volk Michael Jung, Dirk Richarz. GTU - A workbench for the development of naturallanguage grammars. http://www.ifi.unizh.ch/groups/CL/CLpublications.html (Accessed Aug. 22,2003)

9 Martin Volk, Dirk Richarz. Experiences with the GTU grammar development environment. http://www.ifi.unizh.ch/CL/CLpublications/Madrid97Workshop.pdf (Accessed Aug. 22,2003)

10 The XHPSG System. http://www - tsujii.is.s.u - tokyo.ac.jp/rental/ (Accessed Aug. 21,2003)

11 Naoki Yoshinaga, Yusuke Miyao. Grammar conversion from LTAG to HPSG. http://www - tsujii.is.s.u - tokyo.ac.jp/~yoshinag/papers/yoshinag - websls.pdf (Accessed Aug. 25,2003)

(作者 E - mail:huabolin@hotmail.com)