现代汉语句法结构分析器及语言知识库调试环境

技术文档

詹卫东[*] 刘群[+]
[*] 北京大学中文系 北京大学中国语言学研究中心
[+] 中科院计算技术研究所

目 录

1 程序文件及界面

1.1 程序文件

现代汉语句法分析器及语言知识库调试环境（An **I**ntegrated **D**evelopment **E**nvironment for **U**nifcation-based **C**hinese **G**rammar，以下简称 IDE-UCG）运行时需要的文件分为 4 类：（1）主程序及参数配置文件；（2）语言知识库文件；（3）程序运行期间文件（为知识库调试的目的设置）；（4）输入输出文件。详细内容如下表所示：

表 1.1 – 1： ICGDE 程序文件列表

类别	文件名称	功能说明
（1）主程序及参数配置文件	TestParserDoc.exe	句法分析主程序
	TestParserDoc.ini	句法分析程序参数配置文件，必须与主程序在同一目录下
	Slex.dll	分词与词性标注程序模块（参见 2.1）
（2）语言知识库文件	Kbase\model*.*	语言模型文件 Model.txt 为可编辑的文本文件，Model.dat 为二进制文件（两文件在 Kbase\model 目录下）
	Kbase\dictn*.*	词典文件 Dictn.txt 为可编辑的文本文件，Dictn.TAT 为编译后的二进制文件，Dictn.TDX 为二进制索引文件（三文件在 Kbase\dictn 目录下）
	Kbase\dictnref*.*	扩充词典文件 Dictnref.txt 为可编辑的文本文件，Dictnref.TAT 为编译后的二进制文件，Dictnref.TDX 为二进制索引文件（三文件在 Kbase\dictnref 目录下）
	Kbase\prsrbase*.*	句法分析规则文件 Prsrbase.txt 为可编辑的文本文件， Prsrbase.TAT 为编译后的二进制文件（两文件在 Kbase\prsrbase 目录下）
	Kbase\lexicons*.*	分词与词性标注模块使用的词库文件（共 22 个文件在 Kbase\lexicons 目录下。参见 2.1 说明）
	Kbasse\babel.txt	短语结构库文件（babel.txt 文件在 Kbase 目录下）
	Kbasse\babelindex.txt	短语结构库的索引文件（babelindex.txt 文件在 Kbase 目录下）
（3）程序运行期间文件	log.txt	记录程序运行时间
	trace.txt	记录句法分析中间过程的文件（参见 4.1）
	spy.txt	设置断点文件，用于跟踪规则的合一约束工作状况（参见 4.1）
（4）输入输出文件	FileName.stx	输入句子文件(后缀.stx 缩写自 source_text)
	FileName.ttx	输出结果文件(后缀.ttx 缩写自 target_text)
	FileName.trn	关联输入（源文件）和输出（目标文件）的中间文件(后缀.ttx 缩写自 translation_text)

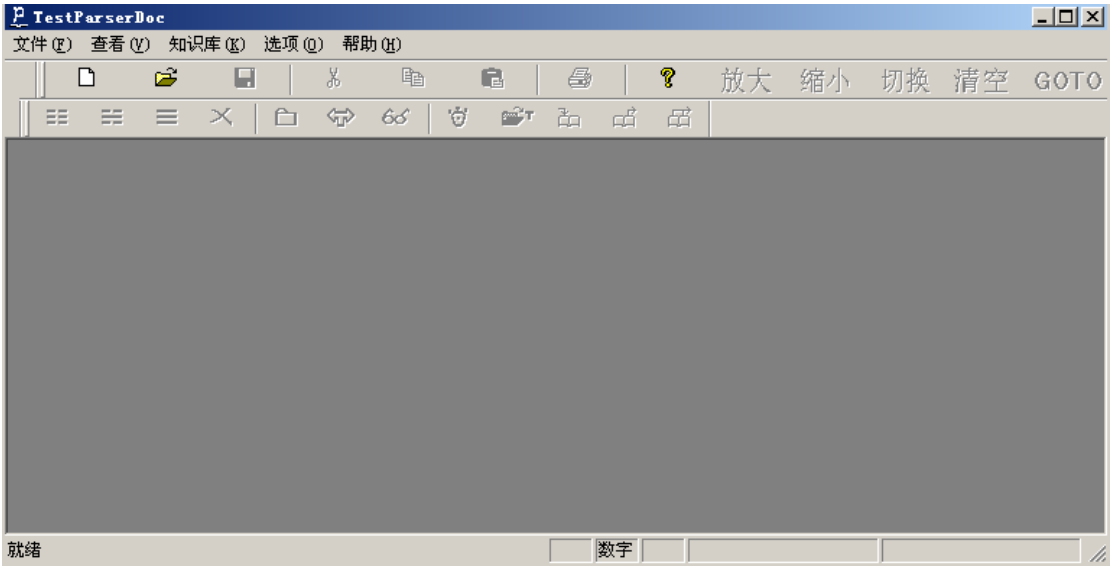
上述文件中第 1、2 类文件是程序运行必须的，如果缺少会造成程序无法运行，甚至无法正常启动。第 3、4 类文件不影响程序运行。第 3 类文件是为程序调试语言知识库服务的，只有选择相应的语言知识库调试功能时，才会用到这些文件。第 4 类文件是程序的输入输出文件。设当前分析的句子文件为 FileName.txt。IDE-UCG 程序将自动生成三个文件 FileName.stx，FileName.ttx 和 FileName.trn。三个文件的文件名部分相同，后缀不同。程序通过打开 FileName.trn 文件，来打开 FileName.stx 和 FileName.ttx 文件，将原始文件和结果文件分别显示在程序窗口的不同区域（参见 1.2 节图 4）。有关.txt、.stx、.ttx、.trn 四个文件间的关系，可参看 1.2 节有关“新建”“打开”“导入”文件操作的详细说明。

TestParserDoc.ini 是程序运行的参数配置文件（详见 2.4 节）。用户可以以文本方式打开该文件手工修改，也可以在程序界面的“选项”菜单下通过程序提供的对话框进行参数设置。（参见 2.4 节图 2.4-1）。所有文本文件均可以用第三方文本编辑器打开进行编辑修改，也可以在 IDE-UCG 程序界面中编辑修改（详见第 2 节和第 3 节的说明）。

1.2 程序界面

IDE-UCG 是基于 MFC（微软基础类库）的多窗口应用程序。程序启动后显示主窗口（Main Frame），如下图 所示。

图 1.2 – 1：程序启动后的初始界面



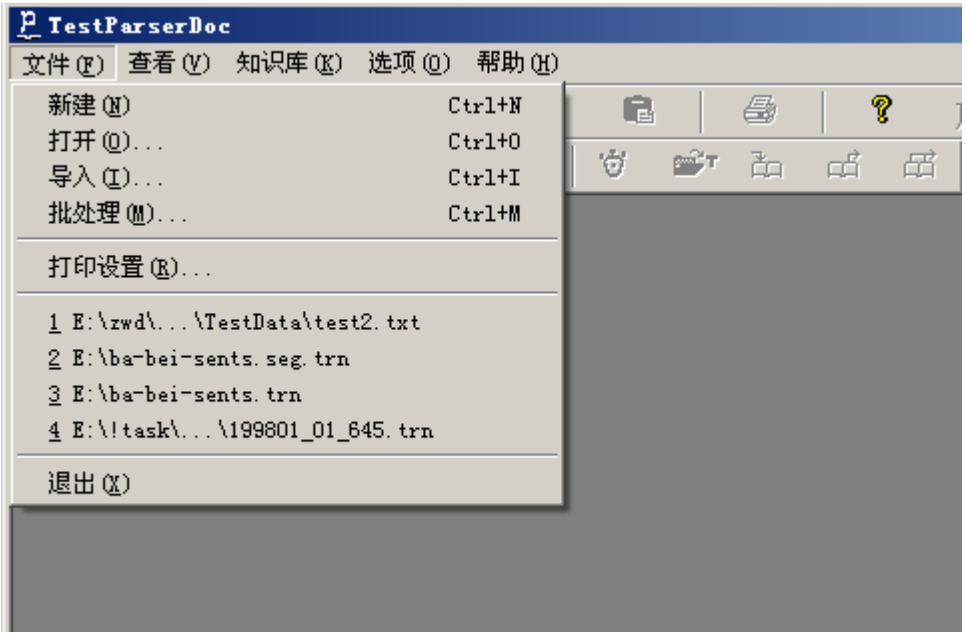
点击“知识库”会弹出知识库管理界面（详见第 3 节）

点击“选项”下的菜单项可以对系统参数进行配置（详见 2.4 节）

在未打开输入文件（待分析句子文件）时，主窗口中的工具条按钮除“新建” 、“打开” 和 “关于” 三个按钮外，都处于非活动状态（inactive），鼠标无法点击。

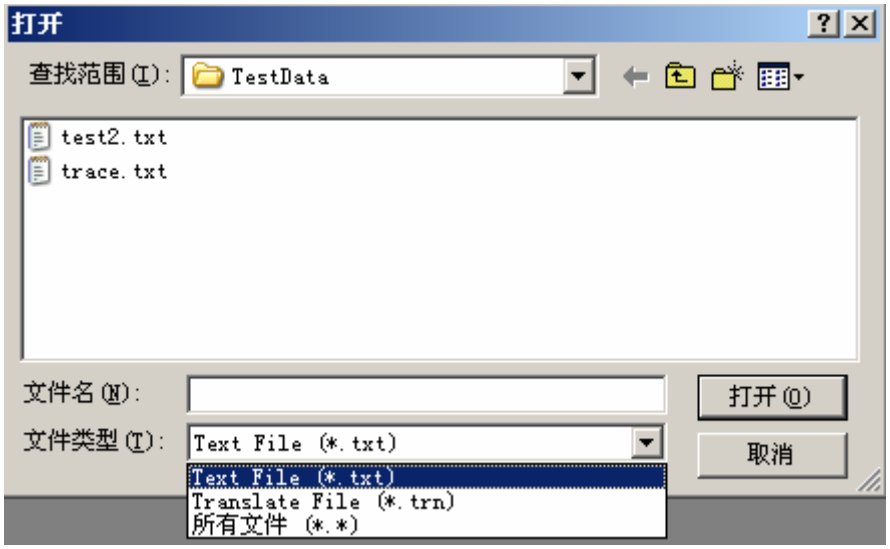
主窗口“文件”菜单下有四种方式可以选择文件进行句法分析，分别是“新建”“打开”“导入”“批处理”，如下图所示：

图 1.2 – 2：程序“文件”菜单项



点击“新建”和“打开”，用户都可以选择两种类型的文件，即普通的.txt 文本文件和关联输入输出文件的 .trn 文件。下图显示了点击“打开”时用户选择文件类型的方式。

图 1.2 – 3：打开文件对话框

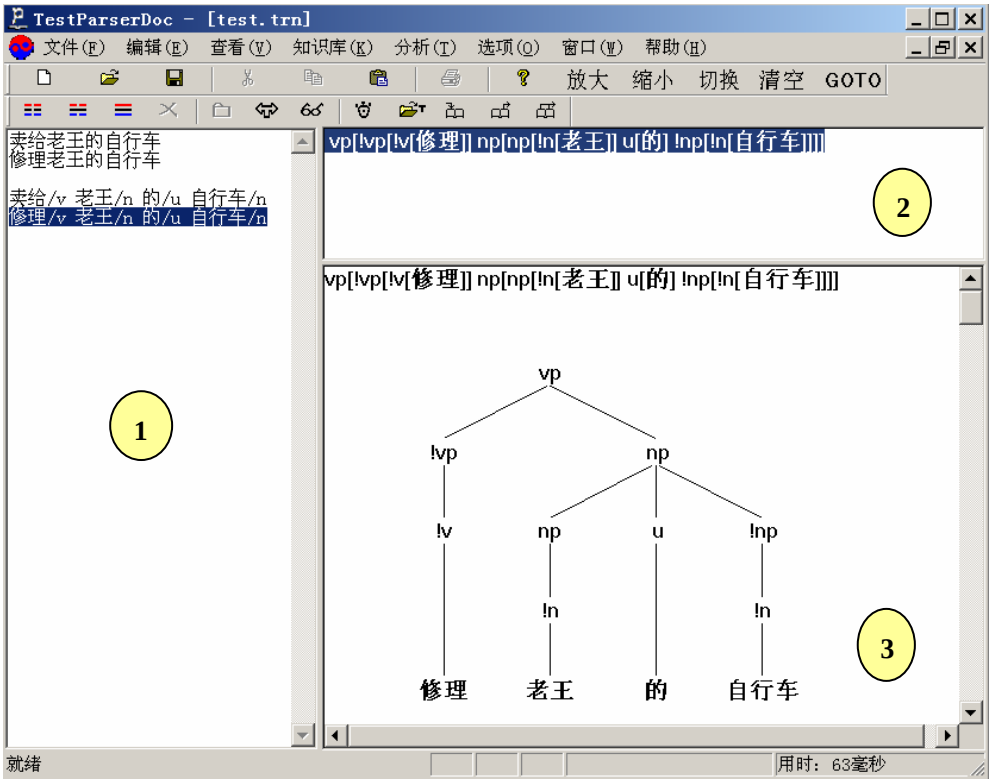


注意：如果选择.txt 文本文件，则只能进行文本编辑操作，不能进行句法分析。只有选择.trn 文件，才能进行句法分析。

点击“新建”和“打开”时用户选择文件类型的方式略有不同。以点击“打开”菜单项为例，在打开文件对话框的“文件类型”处，选择“Translate File (*.trn)”，就可以打开 .trn 文件。打开.trn 文件后，程序将窗口划分为 3 个子窗口，在左子窗口显示原始句子（待进行句法分析）文件(.stx)，在右上子窗口显示分析结果（句法结构）文件（.ttx）。在右下子窗口

显示分析结果的树形图。如下图所示。

图 1.2 – 4：打开.trn 文件时的界面



① 原始句子显示区　②分析结果显示区　③ 句法结构树图显示区

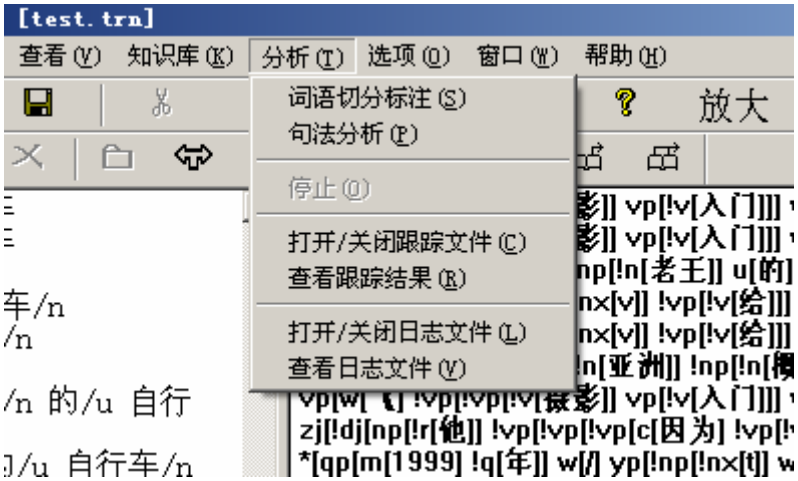
子窗口界面下工具条跟主窗口界面下的工具条不同。子窗口界面工具条中各按钮的功能描述如下表所示：

表 1.2 – 1 ：程序工具条按钮功能说明

图标	功能描述
	对原始句子进行分词和词性标注
	对已经分词和词性标注的句子进行句法结构分析
	对原始句子进行分词、词性标注、句法结构分析
	停止正在分析的进程
	设置规则断点
	打开跟踪分析过程开关
	打开跟踪记录文件（trace.txt）
	将原始句子输出到文件保存
	将分析结果输出到文件保存
	将原始句子和分析结果分别输出到文件保存
放大	放大（宽度）显示句法结构树图
缩小	缩小（宽度）显示句法结构树图
切换	②区显示模式在原始句子与带括号结构标记句子之间切换
清空	清空②区分析结果列表
GOTO	定位到分析结果列表中的某一行

子窗口界面菜单项增加了一个“分析”菜单，其下的各个菜单项如下图所示：

图 1.2 – 5：子窗口菜单项“分析”菜单



各菜单项具体功能为：

（1）词语切分标注：功能跟按钮相同。
（2）句法分析：功能跟按钮相同。（该菜单项不能对已经分词的句子进行句法分析）
（3）停止（分析）：中断当前正在进行的句法分析。
（4）打开/关闭跟踪文件：功能跟按钮相同。
（5）查看跟踪结果；功能跟按钮相同。
（6）打开/关闭日志文件：是否启用日志功能。
（7）查看日志文件：打开日志文件查看。

用户选择点击“文件”菜单下“导入”选项，则只能选择.txt 文件进行处理，假如用户选择导入“test.txt”文件，程序会自动在跟 test.txt 同目录下生成 test.stx，test.ttx，test.trn 三个文件，导入后程序界面跟打开.trn 文件时一样（参见图 1.2 – 4）。开始时 test.stx 文件中内容是从 test.txt 文件中复制而来的，因此内容完全一样。如果在程序运行期间对左子窗口中的文字进行了编辑修改，并保存（ 或 ），则 test.stx 文件内容发生改变，而这个编辑修改操作对 test.txt 文件是没有影响的。

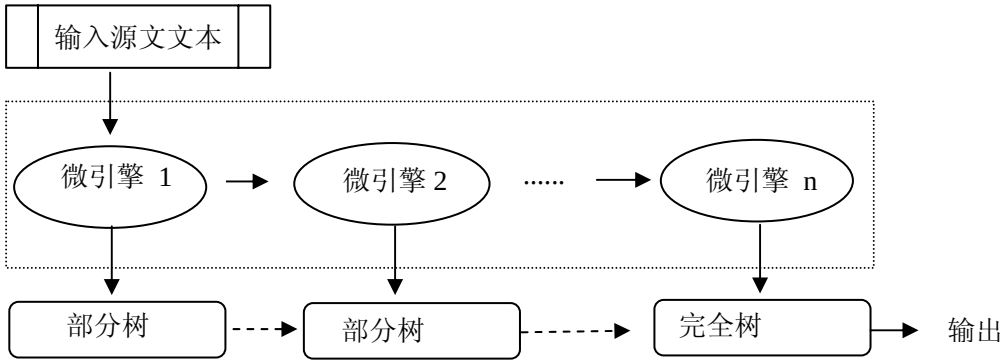
用户选择点击“文件”菜单下“批处理”选项，程序弹出“浏览文件夹”对话框。用户可以在对话框中指定一个目录，程序对该目录下所有文件进行句法分析。分析文件可以是未经分词和词性标注的原始文本，也可以是已经经过分词和词性标注的句子文件。在进行句法分析前，程序会弹出对话框请用户确定所分析文件的类型。分析结果文件以 .prs 后缀名在同目录下保存，文件名跟原始文件名相同。

2 句法分析模块

2.1 微引擎流水线分析模式

IDE-UCG 采用微引擎流水线模式进行句法分析。句子的分析过程被看作是一个树的构建过程：针对原始文本中的一行（句子），程序识别出其中的词语和短语（树节点），并把这些基本的语言成分搭建成一（多）棵完整的句法结构树。这个过程可图示如下：

图 2.1 – 1：微引擎分析模式示意图



每个微引擎由一个识别器（Recognizer）和一个选择器（Selector）组成。识别器识别源文字符串中所有可能的合法语言单位，选择器从识别器所识别的所有可能的候选语言单位中选择可能性大的单位作为中间结果（节点）输出。

IDE-UCG 目前设计了 6 个微引擎来完成句法分析任务，如下表所示：

表 2.1 – 1：IDE-UCG 微引擎名称及功能概要说明

序号	微引擎名称	功能说明	调用知识库/模块
1	SegTagUserRecognizer	全切分分词和词性标注微引擎	Kbase\dictn*.*
2	SegTagBDRecognizer	ICL 分词和词性标注微引擎	Slex.dll Kbase\lexicons*.*
3	PhraseRecognizer	基于短语库的短语识别引擎	Kbasse\babel.txt
4	DictnRecgonizer	基于扩充词典的分析引擎	Kbase\dictnref*.*
5	RuleBasedRecognizer	基于规则库的分析引擎	Kbase\dictn*.* Kbase\prsrbase*.*
6	FailSoftRecognizer	软失败分析引擎	---

SegTagUserRecognizer 对句子进行全切分分词及词性标注，分词词典是 Kbase\dictn 目录下的词典文件，规模较小，因而分词效果不理想，在后续的句法分析中可能造成较多错误。

SegtagBDRecognizer 调用北大计算语言所开发的分词和词性标注程序（Slex.dll）对句子进行分词和词性标注处理。该 Slex.dll 用到的词典知识源在 Kbase\lexicons\目录下，其中

有 6 个主要的分词词典：

UsrLex0　　基本词典， 收入北大计算语言所“现汉语法信息词典”的所有词条

UsrLex1　　用户词典 1，收入用户定义的人名(np)

UsrLex2　　用户词典 2，收入用户定义的地名(ns)

UsrLex3　　用户词典 3，收入用户定义的机构、团体名(nt)

UsrLex4　　用户词典 4，收入用户定义的其它所有专有名词(nz)

UsrLex5　　用户词典 5，收入用户定义的其它普通名词、缩略语、习用语等

程序设计者对各词典中收录词条的性质进行了划分，目的是便于人工维护这些词典，但程序在运行中并不对用户词典中的词条的性质加以区分，因此，实际上用户可以在上述各词典中添加自己需要的词条。词条的基本格式是“word　pos”，即词条跟词性标记以空格分隔开的形式，一个条目占一行。词典中可以有/* */ 或 // 形式的注释。

PhraseRcognizer 可以将句法分析的中间结果跟 babel.txt 中存储的短语条目进行匹配，如果是已经预存的短语，则形成的一个树节点输出。Babel.txt 的规模较小。效果不明显。

DictnRecgonizer 基于一个扩充词典对句中的词语进行识别。扩充词典规模较大，匹配词语成功的概率较高，但缺点是词条没有语法语义信息描述，因而在后续句法结构分析中无法进行有效的合一约束判断。

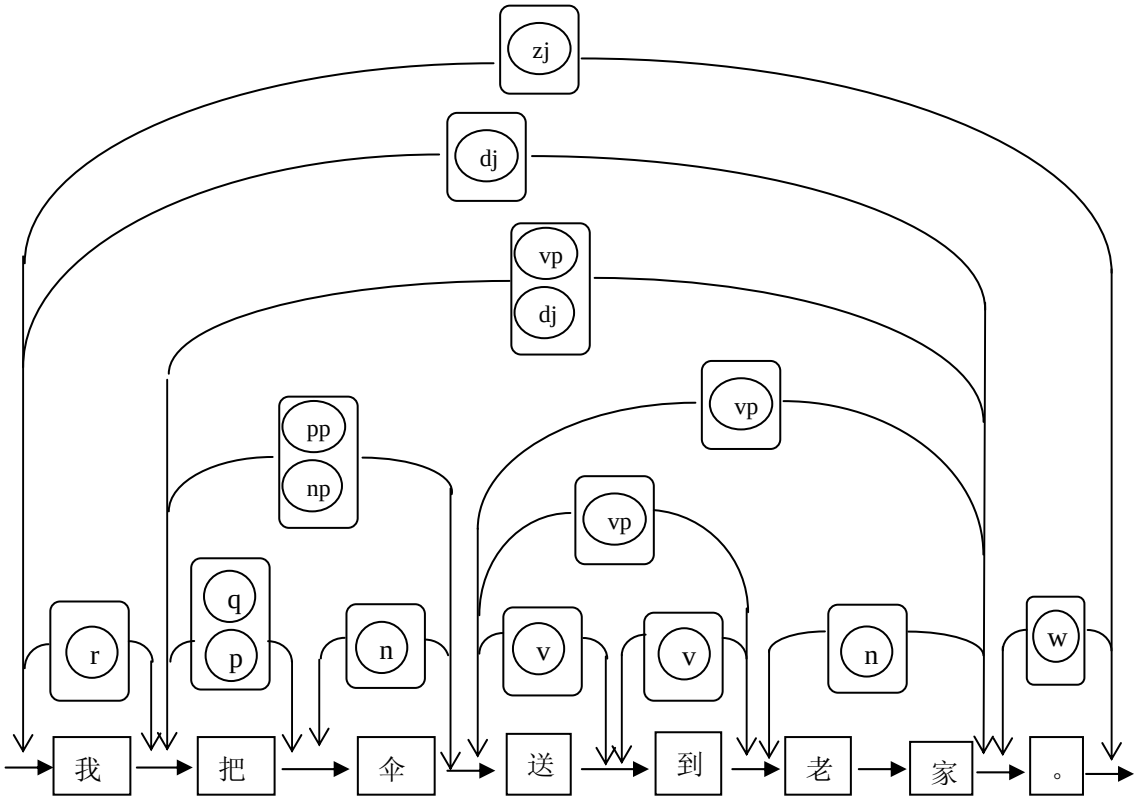
RuleBasedRecognizer 是句法分析的核心引擎，基于 Kbase\dictn\下的词典和 Kbase\prsrbase\下的分析规则对句子的结果进行分析（详见下文第 3 到第 6 节）。

FailSoftRecognizer 在前面的引擎都无法得到完整分析树时调用，产生一个相对最优树。

2.2 公共数据结构及算法描述

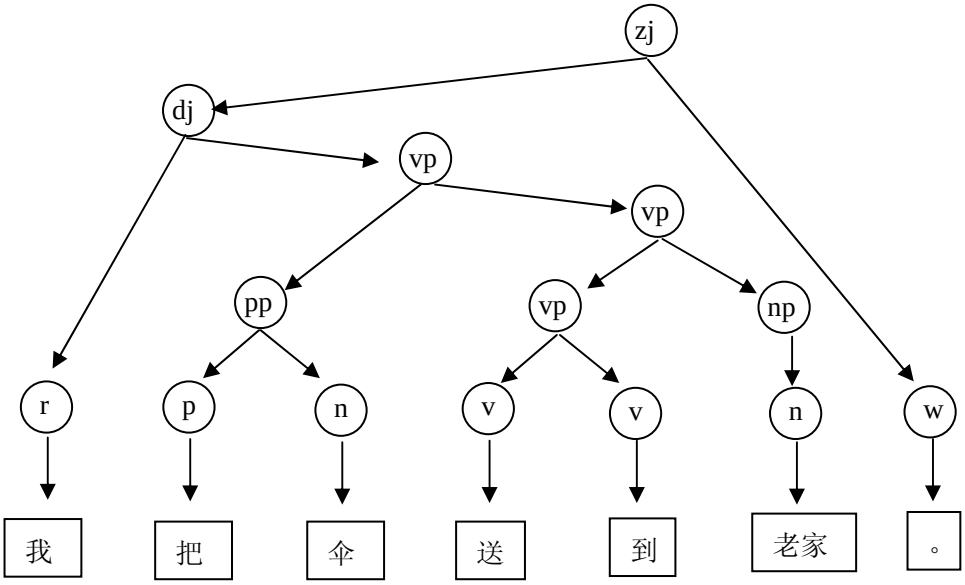
IDE-UCG 的公共数据结构是线图（chart）结构，如下图所示：

图 2.2 – 1：线图结构示意图

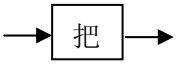
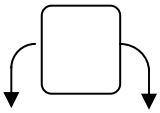


一个线图结构都对应着一个唯一的树（tree）结构，上面线图的树结构如下：

图 2.2 – 2：句法树结构示意图



线图分析中用到的主要数据结构简要描述如下：

符号	名称	说明
	SrcSection	用起始字符和结束字符下标定义一个分析片段。可以是基本单元（汉字、数字、英文单词等）或短语
	SrcNode	chart 中的弧（Arc）或 Tree 中的节点。 可以是一个词，也可以是一个短语或句子。 每一个 SrcNode 由首尾两个 SrcSection 确定其位置
	SrcNodeList	SrcNode 链表。存放句法分析的中间结果（部分树节点）和完整的分析结果（完全树节点）。

分析算法可简要描述如下：

```
BEGIN
    REPEAT 依次从分析流水线中取一个微引擎
        调用该识别器的初始化函数 recognizer->initialize_for_recognition()
        WHILE((TheSrcNode=recognizer->recognize())!=NULL)
            IF 识别出的结点覆盖整个输入文本
            THEN 返回成功，将该结点置为源文根结点
            ELSE 将识别出的结点加入的 Chart 结构中
            ENDIF
        ENDWHILE
    ENDREPEAT
    返回失败
END
```

2.3 节点评分机制

在分析过程中，各个微引擎都向 chart 中加入节点。每个节点都带有一个评分，该评分是选择器选择一个节点（相对得分高）或抛弃一个节点（相对得分低）的依据。

节点的评分定义为可信度(0 到 1 之间)的对数值。因此，评分的取值范围是 0 或者负数。可信度越低，评分值越小。分析程序在运行过程中，每产生一个节点，都将计算该节点的评分值。

一个节点通过构造函数产生的缺省的评分为 0（即可信度为 1）。

通过各微引擎所产生节点的评分方式如下：

（1）通过 SegtagBDRecognizer 词法分析微引擎得到的节点，评分为 0；

（2）通过 PhraseRecognizer 引擎（调用 Babel.txt 短语库）得到的节点，评分为 0；

（3)通过 DictnRecgonizer 引擎(调用 Dictnnref 目录下扩充词典)得到的节点,评分为 log(0.4)（即可信度为 0.4）；

（4）通过 RuleBasedRecognizer 规则引擎（调用 Prsrbase 目录下规则库）得到的节点评分为基本分和惩罚分两部分之和。这两部分评分的具体方式如下：

（A）基本分

节点基本分由两部分组成：所有子节点的评分之和 ＋ 产生该节点的规则的评分；

规则的评分公式为：

a. 如果规则的右侧只有一个节点，可信度为 1，评分为 0；

b. 不含具体词的规则，可信度为 0.5，评分值为 log(0.5)；

c. 含具体词的规则，可信度介于 0.5 到 1.0 之间，具体词越多，可信度越高。

对于上述情况 c，规则的评分分为 10 级，具体分值如下：

```
const float Score::ScaledRuleScore[10] =
{
    (float)log(1.0-1.0/2.0),
    (float)log(1.0-1.0/2.0/2.0),
    (float)log(1.0-1.0/2.0/2.0/2.0),
    (float)log(1.0-1.0/2.0/2.0/2.0/2.0),
    (float)log(1.0-1.0/2.0/2.0/2.0/2.0/2.0),
    (float)log(1.0-1.0/2.0/2.0/2.0/2.0/2.0/2.0),
    (float)log(1.0-1.0/2.0/2.0/2.0/2.0/2.0/2.0/2.0),
    (float)log(1.0-1.0/2.0/2.0/2.0/2.0/2.0/2.0/2.0/2.0),
    (float)log(1.0-1.0/2.0/2.0/2.0/2.0/2.0/2.0/2.0/2.0/2.0),
    (float)log(1.0-1.0/2.0/2.0/2.0/2.0/2.0/2.0/2.0/2.0/2.0/2.0)
};
```

以下是一些规则的示例（规则格式规范参见第 5 节），说明规则的具体程度的差异：

全局规则	局部规则
pp->!p np	pp -> !p<从> tp v<起|开始>
ap->!ap c ap	tp -> mcp !q<点> mcp q<分>
np->mp !np	ap -> m<一> q<年|天> p<比> m<一> q<年|天> !ap
...	...
	vp -> pp(!p<把> np) !vp(!vp vp(!vp<<到>> np))
	…

全局规则指不含终结符的规则。完全由短语类标记或词类标记构成。局部规则指包含终结符（即具体词语）的规则。终结符个数越多，则该规则的可靠性越高。

有的句子既可以通过全局规则得到分析结果，也可以通过局部规则得到分析结果，但评分有差异，通过局部规则得到的分析结果评分高。比如：

“把书运到了图书馆”这个句子通过局部规则得到的分析结果评分为 -2.10694，通过全局规则得到的分析结果评分为-2.79127。

通过局部规则得到的分析结果

vp -> pp(!p<把> np) !vp(!vp vp(!vp<<到>> np))

```
vp -2.10694{vpdao1}
====pp -0.694124{pp3}
    ====p<把> 0{}
    ====np -0.00097704{np00}
        ====n<书> 0{}
====vp -1.40497{vpyundao}
    ====vp -0.00097704{vp00}
        ====v<运> 0{}
    ====vp -0.71085{vpsb1}
        ====vp -0.0167254{vp1}
            ====vp -0.00097704{vp00}
                ====v<到> 0{}
            ====u<了> 0{}
        ====np -0.00097704{np00}
            ====n<图书馆> 0{}
```

通过全局规则得到的分析结果

vp->pp !vp

```
vp -2.79127{vpzz1}
====pp -0.694124{pp3}
    ====p<把> 0{}
    ====np -0.00097704{np00}
        ====n<书> 0{}
====vp -1.404{vpsb1}
    ====vp -0.709873{vp1}
        ====vp -0.694124{vpsbu1}
            ====vp -0.00097704{vp00}
                ====v<运> 0{}
            ====v<到> 0{}
        ====u<了> 0{}
    ====np -0.00097704{np00}
        ====n<图书馆> 0{}
```

（B） 惩罚分

一个节点的罚分为以下 5 部分之和。每个部分都设置了惩罚因子，可在参数配置中由用户指定（参见下文 2.4.1）。

a.如果该结点内含标点、单侧非标点，则加上相应的惩罚因子 InsidePunct1（对数值）。
b.如果该结点内含标点、双侧非标点，则加上相应的惩罚因子 InsidePunct2（对数值）。
c.已有同区域同标记结点数 × 惩罚因子 SameLabel（对数值）
d.已有同区域同标记同规则结点数 × 惩罚因子 SameRule（对数值）
e.已有同区域同标记同规则同核心结点数 × 惩罚因子 SameKernal（对数值）

概括来说，有两种原因要导致对一个分析节点进行罚分：

原因一： 生成的一个短语中间有一个标点符号，而短语一侧（或两侧）没有标点符号。上面 a，b 罚分因子针对的就是这种情况；

原因二： 在同一个位置上生成大量的同类型短语结点。上面 c，d，e 罚分因子针对就是这类情况。

第一种情况是体现了标点符号对短语的约束作用，在一般情况下，标点符号应是短语的边界，而不应在一个短语的内部，但由于实际语料中也存在短语内部包含标点符号的情况，因此在允许短语内部包含标点符号的前提下，对其他条件完全相同的中间分析结果，将包含标点符号的短语分析结果进行罚分处理；

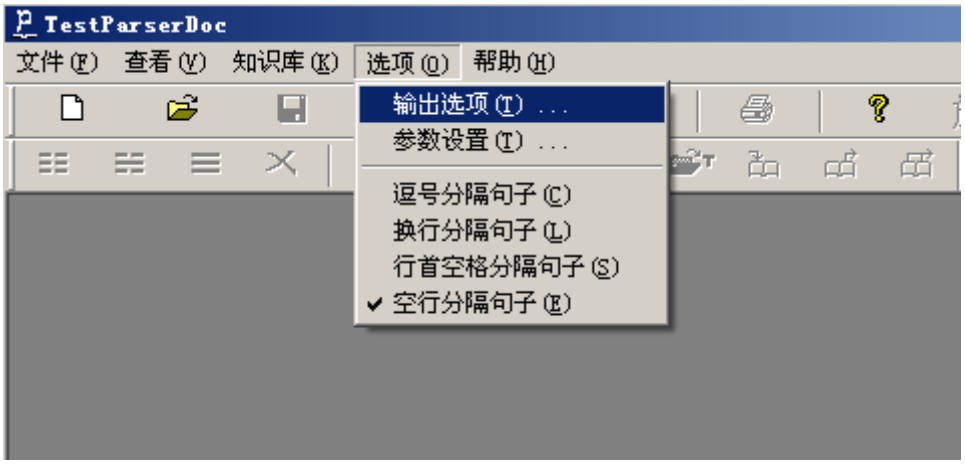
第二种情况是避免在同一个位置上反复生成大量同类短语，尽早在分析过程中剪枝，以避免大量的中间结果累积，造成分析树的组合爆炸问题，影响分析效率。如果两个节点覆盖的语言片段相同，并且根节点标签相同，或是同一条规则的分析结果，或其中心节点相同，则对符合这些情况的节点进行罚分；惩罚因子 SameLabel、Samerule、SameKernal 对应的节点中相同成分逐渐增多。因此一般来说，罚分应该逐渐加重。

（5）通过软失败引擎（FailSoftRecognizer）得到的节点给一个非常低的评分（缺省为-1000）。

2.4 参数设置

上文 1.1 节中已经提到，IDE-UCG 程序的.ini 文件（TestParserDoc.ini）中可以设置程序运行的参数。程序启动后，在“选项”菜单下可以进行参数设置。

图 2.4 – 1：程序参数配置菜单选项



上图中“输出选项”用于确定分析结果的格式（详见下面 2.4.1 的说明）。“参数设置”用于指定分析过程中对分析结果会产生影响的多项参数（详见下面 2.4.2 的说明）。“逗号分隔句子”“换行分隔句子”“行首空格分隔句子”“空行分隔句子”等 4 个菜单项是开关型菜单，选中的菜单项前面会出现一个√符号。这 4 个菜单项是由用户指定句子的判别标准。句子是 IDE-UCG 程序进行分析的基本单位。在未选定语言片段的情况下，程序默认对文件（*.stx）中的全部句子进行句法分析。如何分隔文件字符流中的句子，是由用户指定分隔标记的。程序给出的 4 个候选分隔标记为：（1）逗号；（2）换行符；（3）行首空格；（4）空行。

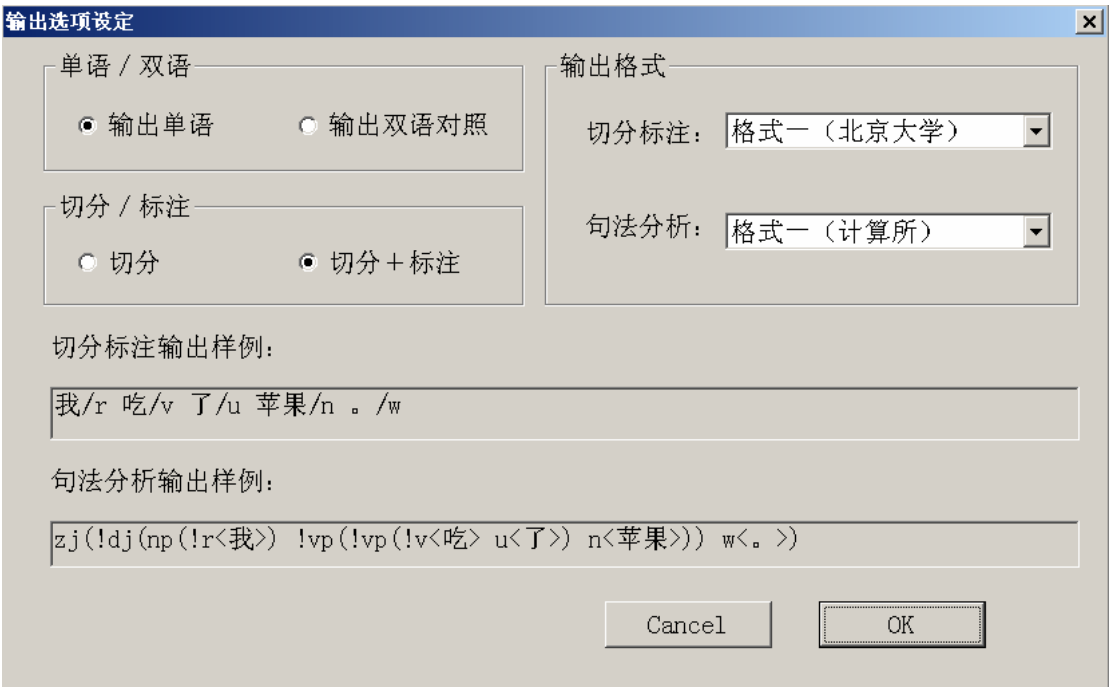
这部分设置对应在 TestParserDoc.ini 文件中的项目为：

```
[Sentence Segmentation]
Comma=0
NewLine=0
SpaceLine=0
EmptyLine=1
WordBySpace=0
```

2.4.1 输出选项：影响分析结果的输出形式

IDE-UCG 对分词和词性标注以及句法结构的分析结果，均提供了不同标记格式的支持。用户可选择、指定不同输出形式。在下图“输出格式”处，可以通过下拉菜单选择不同单位的结果文件格式。对话框下半部分文本框中显示不同格式的样例。

图 2.4 – 2：输出参数设置对话框



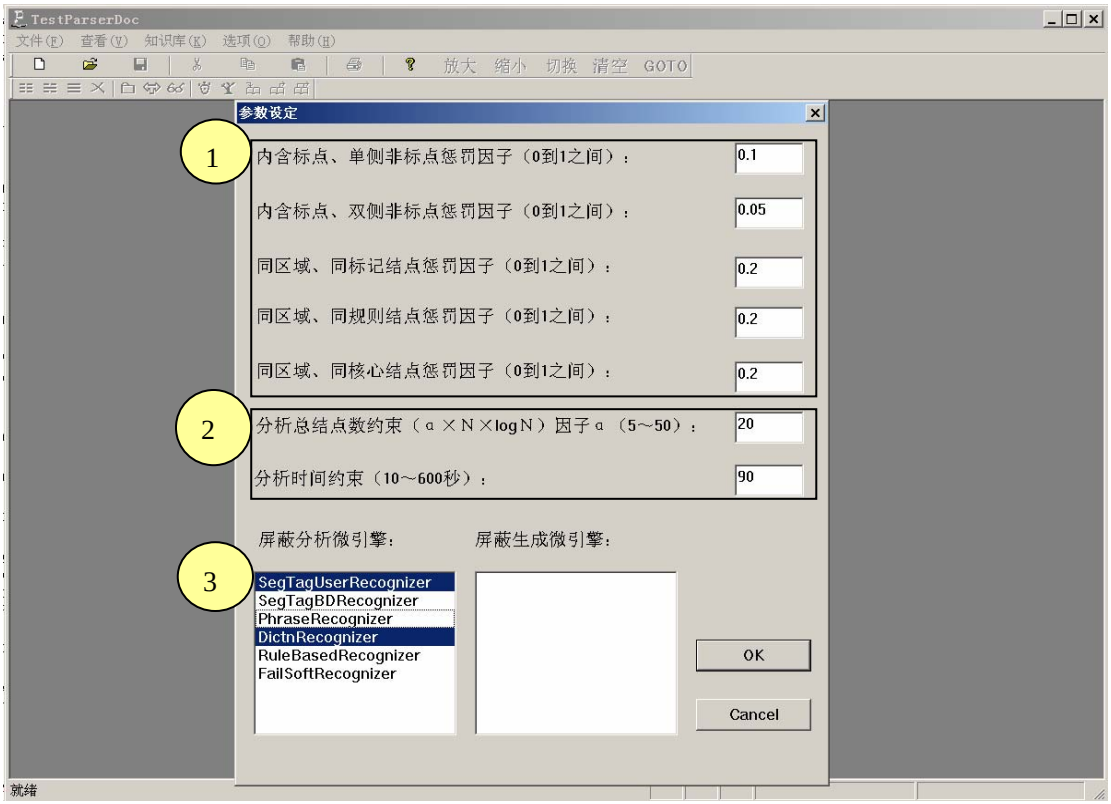
这部分参数设置后在 TestParserDoc.ini 文件对应的项目如下：

```
[Translation Output]
Bilingual=0
POSTag=1
SegTagFormat=1
ParseFormat=1
```

2.4.2 参数设置：影响句法分析的结果内容

点击“选项”下“参数设置”后，弹出下图所示的对话框。

图 2.4 – 3 ： 参数设置对话框



①┌ 关于分析树节点惩罚因子参数的说明

文本编辑框中的数值范围在 0 到 1 之间。值越大，罚分越小。如设置为 1，则表示该惩罚因子在程序分析过程中不起作用（1 取对数后为 0，在程序运算时，该罚分不计入）。用户输入的参数值将记录在 TestParserDoc.ini 文件中。

上图所示的罚分因子设置及其在 TestParserDoc.ini 文件中保存的对应值为：

内含标点、单侧非标点惩罚因子 0.1　　ini 文件中记录：InsidePunct1 = 100（0.1 * 1000）
内含标点、双侧非标点惩罚因子 0.05　ini 文件中记录：InsidePunct1 = 50（0.05 * 1000）
同区域、同标记节点惩罚因子 0.2　　ini 文件中记录：SameLabel = 200（0.2 * 1000）
同区域、同规则节点惩罚因子 0.2　　ini 文件中记录：SameRule = 200（0.2 * 1000）
同区域、同核心节点惩罚因子 0.2　　ini 文件中记录：SameKernal = 200（0.2 * 1000）

用户在参数设置对话框中设定的参数值乘以 1000 后，记录在 TestParserDoc.ini 文件中，程序中在计算罚分时实际上取的是罚分因子的 log 值，即 log(0.1), log(0.05) 等。TestParserDoc.ini 文件中反映罚分参数设置的项目为：

[Punishment]
SameLabel=200
SameRule=200
SameKernal=200
InsidePunct1=100
InsidePunct2=50

② 关于分析过程时间开销和空间开销约束参数的说明

为了避免分析时间太长，强制句法分析在一定条件下结束。

分析总节点数约束是对分析过程中产生的 SrcNode 总数的限制。总节点个数不能超过句子中的 SrcSection 个数（N）乘以 LogN 再乘以节点数约束因子 α 。

分析时间约束指分析时间如超过设定的秒数则分析进程结束。

这部分参数设置在 ini 文件中对应的项目为：

[Limitation]

NodeNumber=50 （节点数约束因子）

Time=30 （分析时间约束因子）

③ 关于屏蔽分析微引擎的说明

用户可以打开/关闭某个微引擎。在分析微引擎列表框中选中某个微引擎（蓝色背景），则该微引擎被关闭。否则，该微引擎处于打开状态。

序号	微引擎名称	功能说明
1	SegTagUserRecognizer	全切分分词和词性标注微引擎
2	SegTagBDRecognizer	北大计算语言所分词和词性标注程序模块
3	PhraseRecognizer	基于短语库的短语识别引擎
4	DictnRecgonizer	基于扩充词典的分析引擎
5	RuleBasedRecognizer	基于规则库的分析引擎
6	FailSoftRecognizer	软失败分析引擎

用户屏蔽微引擎的参数设置在 TestParserDoc.ini 文件中对应的项目为：

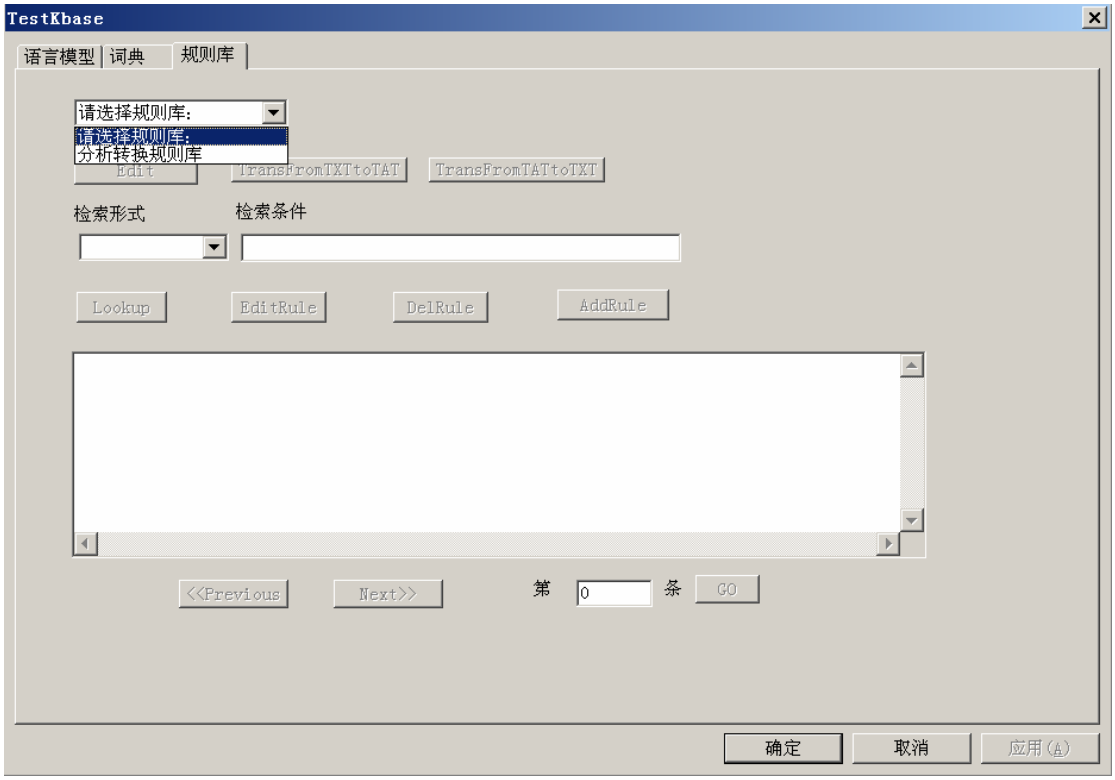
[AnalyzerMaskTable]

AnalyzerMaskTable=1 0 0 1 0 0 （其中 0 表示微引擎未被屏蔽，1 表示屏蔽）

IDE-UCG 的主要设计目标是研究汉语句法分析规则知识的形式表达。因此分析器运行时，通常选择屏蔽 1，3，4 等几个微引擎。

3 语言知识库管理模块

点击主菜单项“知识库”，弹出下面的知识库管理界面对话框：



用户通过上述界面可编辑的知识库包括“语言模型”“词典”（核心词典+扩充词典）“规则库”（句法分析规则库）。关于语言知识库的格式规范参见下文第 5 节。

3.1 语言模型

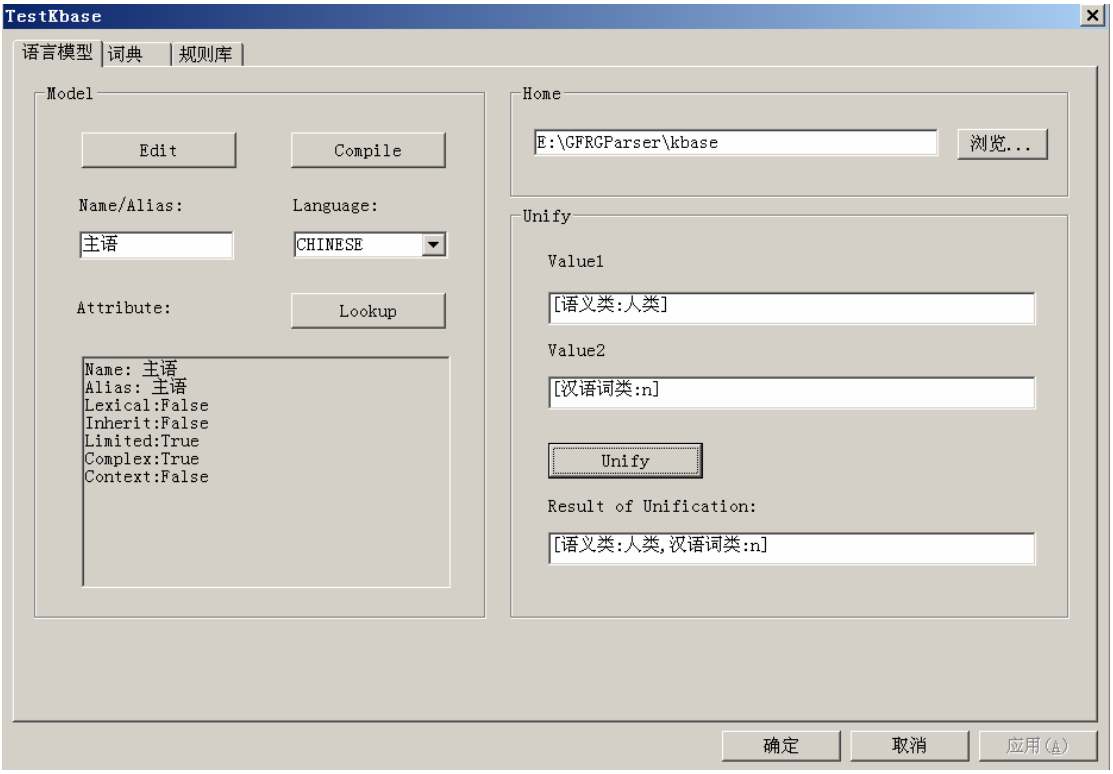
（1）点击“Edit”按钮，打开 kbase\model\model.txt 文件（默认打开程序为 notepad/记事本）。用户可直接编辑该文件。编辑完成保存后，点击 Compile，程序将 model.txt 转存为 model.dat 文件。程序在启动时读取 model.dat，作为检查知识库文件（dictn.txt、prsrbase.txt 等）中的表达形式是否符合规范的依据（参见下文第 5 节说明）。

（2）Home 区域指定知识库所在的文件夹。并将设定结果保存在 ini 文件中：

[Kbase]

Home=..\..\kbase　　（Home 中可以 TestParserDoc.exe 所在路径为基准，指定相对路径）

（3）在知识库的“语言模型”管理页面，用户可以测试两个特征值合一的结果。



下面是一些特征值合一的例子：

特征名（Attr）	取值类型	Value1	Value2	合一结果
语义类	层次型	事物	人类	人类
语义类	层次型	~具体事物	人类	合一失败
汉语词类	符号型	n	r	合一失败
汉语词类	符号型	n	~r	n
汉语词类	符号型	~n	~r	~n ~r
汉语词类	符号型	n|r	~n	r
体谓准	符号型	体|谓	体|准	体
体谓准	符号型	体|谓|准	体|准	体|准
主语	特征结构	[语义类:人类]	[汉语词类:n]	[语义类:人类,汉语词类:n]
主语	特征结构	{主体:[语义类:人,汉语词类:n]}	[语义类:植物]	[语义类:植物]{主体:[语义类:人,汉语词类:n]}

3.2 词典

（1）点击“Edit”按钮，打开 kbase\dictn\dictn.txt 文件进行编辑。

（2）点击“TransFromTXTtoTAT”按钮，将 dictn.txt 转换为 dictn.tat 和 dictn.tdx 文件。
（3）点击“TransFromTATtoTXT”按钮，将 dictn.tat 和 dictn.tdx 转换为 dictn.txt 文件。
（4）在词语输入文本框输入一个词语 W，点击“lookup”按钮，在词典中查找 W。并将查找结果显示在下面的文本框供用户编辑。

— 用户编辑完成后，点击“EditDictn”按钮，将内存中维护的 W 词条信息更新，但不写入 dictn.dat 文件。程序结束时，系统提示用户是否将修改后的词条写入到磁盘文件 dictn.dat。
— 点击“DelDictn”按钮，删除当前查到的词条（只是删除内存中的该词条，并未删除磁盘文件 dictn.dat 中的词条）
— 如果没有查到 W，用户可以将 W 作为一个新词条进行编辑，编辑结束后，点击“AddDictn”按钮，将 W 加入到内存词库数据结构中，但不写入 dictn.tat 文件。程序结束时，系统提示用户是否将修改后的词条写入到磁盘文件 dictn.tat。

对扩充词典（Dictnref.txt）的操作方式与对核心词典（Dictn.txt）的操作方式完全一样。

3.3 规则库

（1）对规则库的检索有两种模式。“规则名检索”模式：输入规则名，点击“lookup”按钮，检索到唯一一条规则（参见下文第 5 节对规则格式的说明）。“全文检索”模式：输入任意字符串，点击“lookup”按钮，对内存中的规则数据进行字符串匹配操作。命中结果可顺序显示在下面的文本框中，供用户编辑。点击 Previous 和 Next 按钮，可定位到不同的命中记录（规则）。

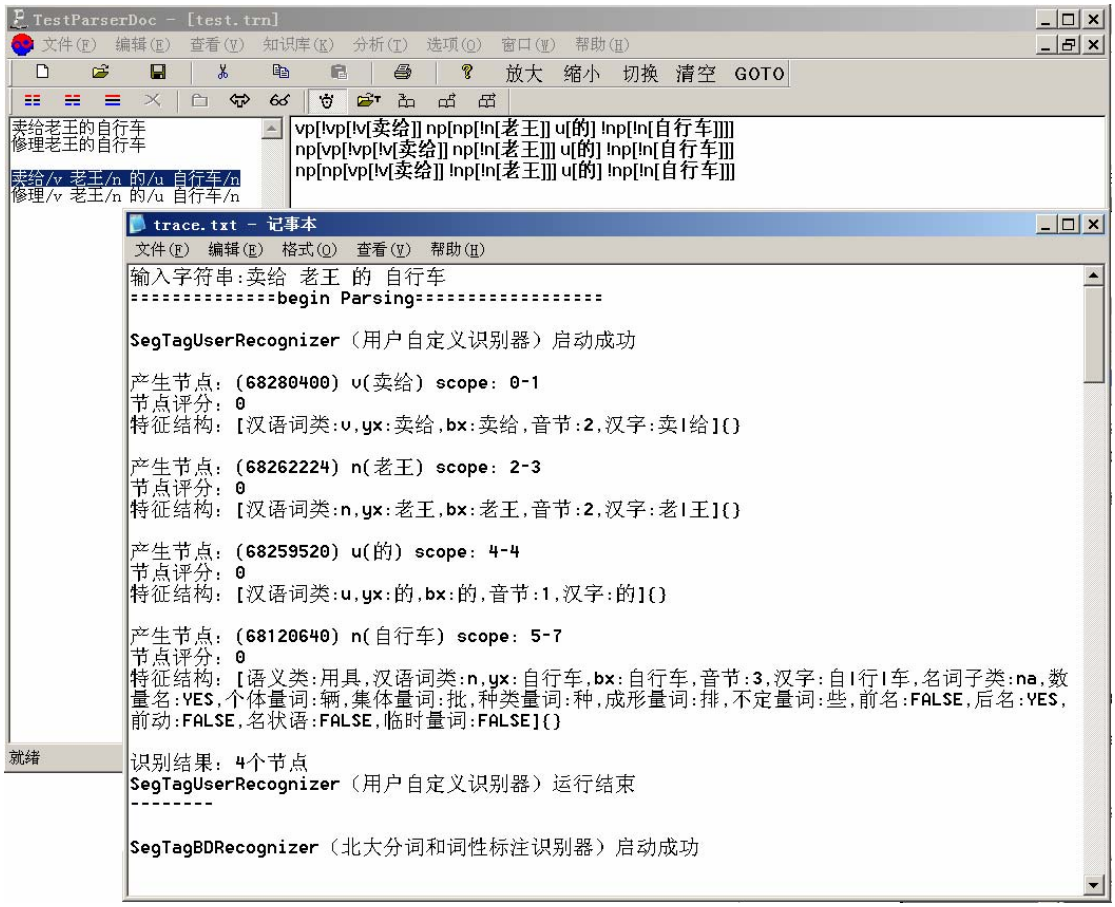
在“lookup”之后，可执行下面三种操作（类似上面对词库的操作）

（2）EditRule：对查到的规则进行编辑，并将结果写入内存中的规则数据结构，但不直接将结果写入磁盘文件 prsrbase.TAT。程序退出前，系统会提示是否写入。
（3）DelRule：删除一条规则。但不改变磁盘文件 prsrbase.TAT。
（4）AddRule：增加一条新规则。但不改变磁盘文件 prsrbase.TAT。
（5）以上是针对单条（多条）规则的编辑操作。也可直接点击“Edit”按钮，打开 prsrbase.txt 文件，直接进行规则的编辑。编辑完成后，要点击“TransFromTXTtoTAT”按钮，将 prsrbase.txt 转换为 prsrbase.TAT 文件。

4 句法分析过程跟踪模块

4.1 记录句法分析的中间过程

点击工具条上的 图标，可打开跟踪开关。程序将分析的中间过程输出到 trace.txt 文件中（除此之外，程序还将输出全部的句法分析结果。这跟在没有按图标时，程序只输出一个句法分析结果是不同的）。分析结束后，点击 图标，系统调用 notepad 程序打开 trace.txt 文件。



按下 图标后，程序分析“卖给/v 老王/n 的/u 自行车/n”，得到 3 个句法分析结果。分析过程记录在 trace.txt 文件中。该文件按照调用微引擎的顺序，显示每个引擎启动、运行、结束的过程。中间的分析结果包括节点及其特征结构信息。

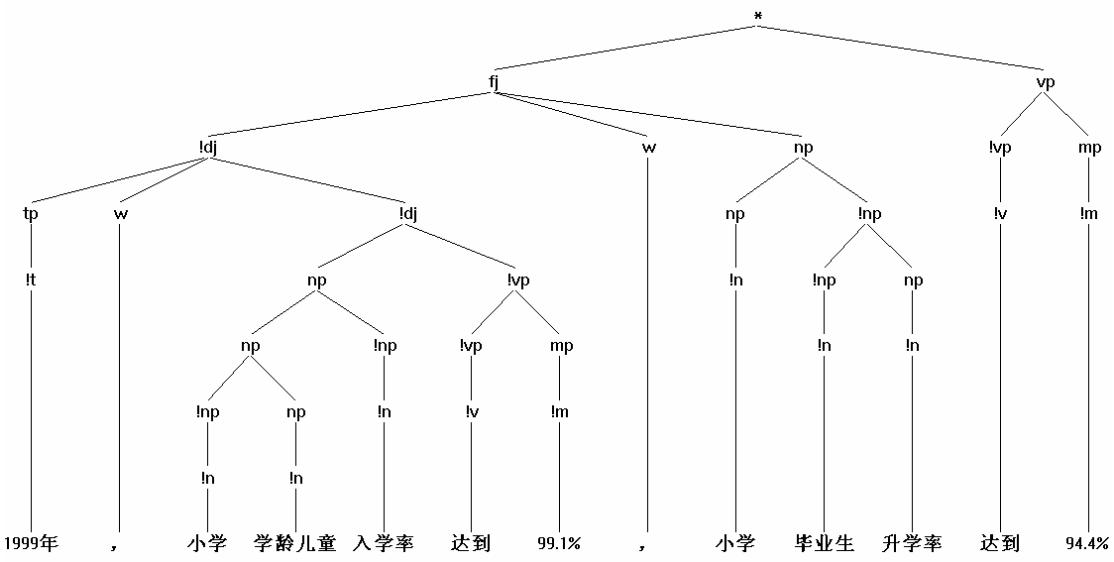
4.2 设置断点监测规则合一失败的原因

点击工具条上的  图标，会弹出设置规则“断点”的对话框。所谓规则“断点”（Breakpoint），是类似调试程序时在程序的某一行设置断点，以跟踪程序在执行到该行时内存中变量的取值状况。设置规则“断点”后，程序在进行句法分析过程中，将记录调用该规则时合一约束失败的情况，并把结果输出到 trace.txt 文件中（可查找“合一失败原因”定位到该记录），以便用户了解一条规则为何在分析过程中没有被调用，即查明合一失败的原因。

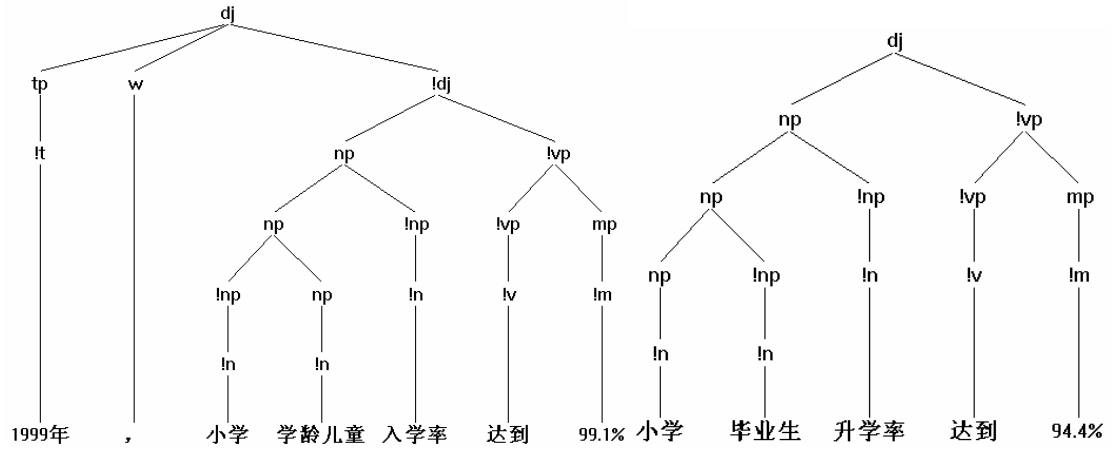
例如，分析下面的句子：

1999 年/t ，/w 小学/n 学龄儿童/n 入学率/n 达到/v 99.1%/m ，/w 小学/n 毕业生/n 升学率/n 达到/v 94.4%/m

结果是未能得到正确的句法结构树：

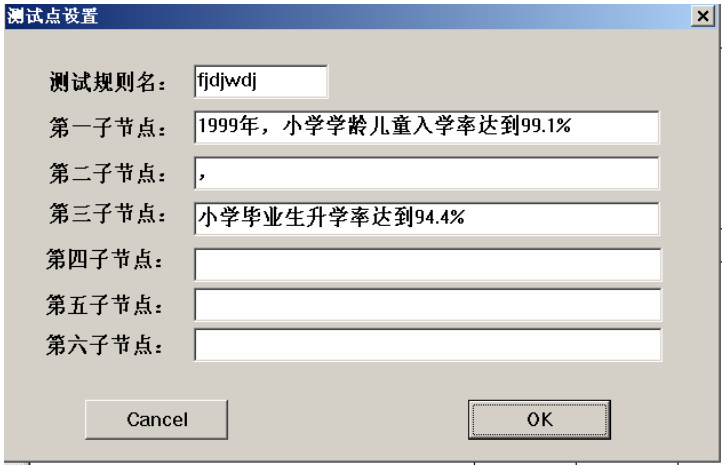


但分别分析“1999 年/t ，/w 小学/n 学龄儿童/n 入学率/n 达到/v 99.1%/m”和“小学/n 毕业生/n 升学率/n 达到/v 94.4%/m”两个部分，都能得到正确的句法结构树：



由此产生问题，为什么两个 dj（中间以逗号，分隔）不能组成一个 fj 呢。查 prsrbase.txt 规则库有可用的组合规则：&& {fjdjwdj} fj->dj w<","|"，"|"；"|";"|"…"|"……"|"："> !dj

现在想查该规则的合一约束条件为何失败，就可以点击 图标，设置规则断点如下：

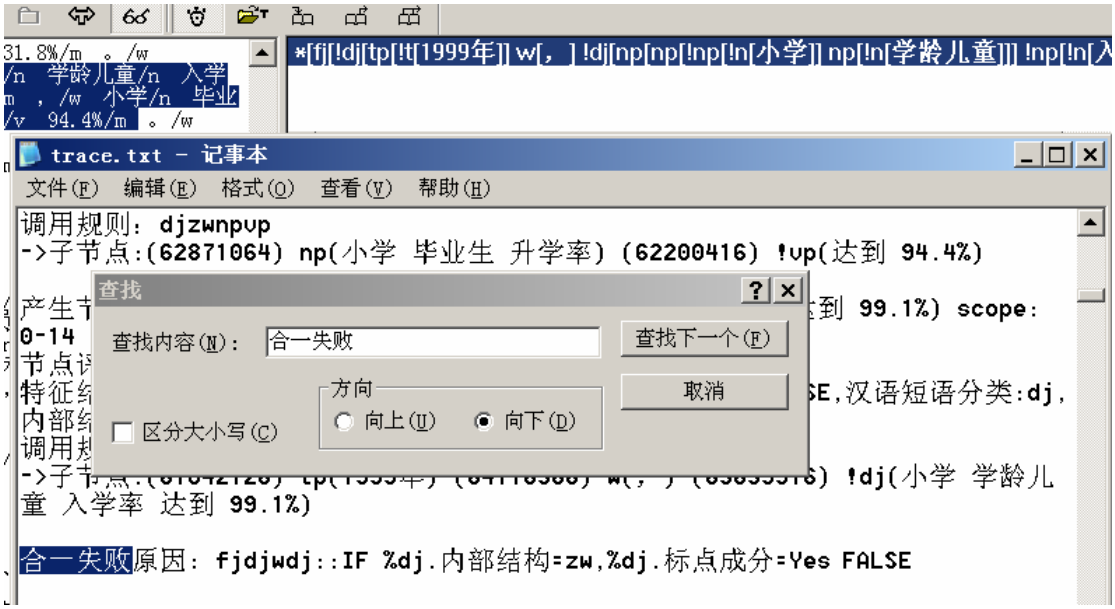


规则 fjdjwdj 箭头右部有 3 个子节点，因而在上面对话框中，要填写 3 个子节点分别覆盖的语言片段（SrcSection）。

设置断点完成后，点击 OK 按钮，上述断点设置信息将保存在 spy.txt 文件中。下次点击图标时，程序会从 spy.txt 文件读取上一次设置的断点信息。

要让断点设置起作用，还需要点击 图标，这样才会生成 trace.txt 文件，相应的合一失败原因信息才会记录在 trace.txt 文件中。

分析结束后，点击图标，打开 trace.txt 文件，查找“合一失败原因”，即可定位到相应记录出。了解规则的合一约束条件为何未通过。



在上面这个例子中，fjdjwdj 规则的合一约束中有这样一条：

IF %dj.内部结构=主谓,%dj.标点成分=Yes FALSE

这个约束条件要求 fj 的第一个 dj 不能是主谓结构，且包含标点符号。

上面例句中的第一个 dj 是“1999 年/t ，/w 小学/n 学龄儿童/n 入学率/n 达到/v 99.1%/m”，该 dj 是主谓结构，且包含标点符号，因此不符合这一条的要求，造成整个句子未通过这条规则的测试要求。

5 语言知识的形式化表达

5.1 关键字

语言知识库中使用了下列关键字：

SemModel	SynModel	EndModel	
SemCat	SemAtt	SemVal	
LexCat	LexAtt	LexVal	
PhrCat	PhrAtt	PhrVal	
SemAssAtt	SemAssVal	SynAssAtt	SynAssVal
BaseForm	VaryForm	Static	Dynamic
Limited	Unlimited	RelMode	Default
Hierar	Symbol	Number	Boolean
Field	Digit	Punct	Other
IF	TRUE	FALSE	
THEN	ELSE	ENDIF	
CONDITION	OTHERWISE	SELECT	

标识符[1]：

<Ident>:
    <NrmIdent>
    <StrIdent>
<NrmIdent>:
    <Alpha>
    <NrmIdent><Alpha>
    <NrmIdent><Digit>
<StrIdent>:
    "<String>"
<Alpha>:
    _
    <ChnChar>
    <LtnChar>
<LtnChar>:
    A-Z
    a-z
<ChnChar>:
    汉字

[1] 本文在对形式文法中所用到的术语进行定义时，遵循的体例是：被定义项都放在< > 中，后面紧跟一个冒号 ：然后在接下来的若干行中列举该术语的组成形式，每行一个形式，一行内的字符串是组合关系，每行间字符串是逻辑“或”的关系。

<Digit>:
0-9
<String>:
任意字符串，其中的双引号(")重复一次
数：
<Integ>:
<Digit>
<Integ><Digit>
分隔符：
. , ; : = | ~
( ) < > [ ] { }
^ && # ## $$ ** ! $ / & * + -> == != :: << >>
% %% %%% %%%%
注释：
位于 /* 和 */ 之间的字符串

除了以字母（包括汉字和下横线）开头的字母数字串以外，括在双引号中的任意字符串也可以作为标识符使用，用于表示标点符号等特殊符号。双引号内的字符串中如果又出现双引号（"）应重复一次，用两个双引号（""）表示。 除非在双引号内，否则标识符中不允许出现空白符号。两个标识符之间必须有空白或其它符号分隔。

5.2 描述语言知识的基本形式

本节介绍描述语言知识（包括规则、词典和短语、句子结构等）的基本形式。

5.2.1 原子

原子是最基本的描述单位，它不可以再分割，但可以用‘与’，‘或’，‘非’等逻辑符号相联结。原子之间的基本运算是合一（unification）。

<Atom>:
<Hierar>
<Symbol>
<Number>
<Boolean>
<Hierar>:
<Hierar_Or>
<Hierar_Or>:
<Hierar_And>
<Hierar_Not>
<Hierar_Or>|<Hierar_And>
<Hierar_And>:

```
            <Hierar_Smp>
            <Hierar_And>~<Hierar_Smp>
    <Hierar_Not>:
            ~<Hierar_Smp>
            <Hierar_Not>~<Hierar_Smp>
    <Hierar_Smp>:
            <Ident:Atom_Name>
            <Ident:Atom_Alias>
    <Symbol>:
            <Symbol_Or>
            <Symbol_Not>
    <Symbol_Or>:
            <Symbol_Smp>
            <Symbol_Or>|<Symbol_Smp>
    <Symbol_Not>:
            ~<Symbol_Smp>
            <Symbol_Not>~<Symbol_Smp>
    <Symbol_Smp>:
            <Ident:Atom_Name>
            <Ident:Atom_Alias>
    <Number>:
            <Number_Or>
            <Number_Not>
    <Number_Or>:
            <Number_Smp>
            <Number_Or>|<Number_Smp>
    <Number_Not>:
            ~<Number_Smp>
            <Number_Not>~<Number_Smp>
    <Number_Smp>:
            <Integ>
            <Ident:Atom_Alias>
    <Boolean>:
            <True>
            <False>
            <Ident:Atom_Alias>
```

原子<Atom>主要有四种不同的形式：层次型、符号型、数值型和布尔型。进行运算时，所有的别名 <Atom_Alias> 都要转换成其基本形式<Atom_Name>,<Integ>,<True>,<False>。

布尔型<Boolean>最简单，它只有<True>和<False>两个值，它不能组成与、或、非等复合形式，布尔值只能与它本身合一。

数值型<Number>可以取所有自然数或零，单个数值只能与其本身合一。数值可以组成两种复合形式：

（1） ~m~n...(非 m 且非 n...)

（2） m|n|t...(m 或 n 或 t...)。

符号型<Symbol>可以取标识符为值，单个符号只能与其本身合一。其组合形式与数值型相同。

层次型<Hierar>比较复杂。假定有单个层次型值 α 和单个层次型值 β，当且仅当存在一个串 γ，能满足 α = β γ（或 β = α γ）时，α 和 β 才能够合一，合一的结果为 α（或 β）。这时我们称 α 为 β（或 β 为 α）的下位值，β 为 α（或 α 为 β）的上位值。它们反映了多级树状分类系统中上位范畴和下位范畴之间的关系。层次型有三种复合形式：

（1） ~A~B...(非 A 且非 B...)

（2） A~A1~A2...(A 且非 A1 且非 A2...)

（3） 上述两种形式的“或”(|)

对于第二种形式，要求 A1，A2 等必须是 A 的下位值。

5.2.2 特征结构

特征结构用于语言单位（如词语、短语、句子等）的内部结构，也用于描述规则和词典。它的基本形式是“属性——值”对的集合。

<Fest>:
[ ]
{ }
[ ] { }
[<IntFest>]
{ <ExtFest> }
[<IntFest>] { <ExtFest> }
<None>

<IntFest>:
<IntFeat>
<IntFest> , <IntFeat>
<IntFest> ; <IntFest>

<ExtFest>:
<ExtFeat>
<ExtFest> , <ExtFeat>

<IntFeat>:
<IntAtt> : <IntVal>

<ExtFeat>:
<ExtAtt> : <ExtVal>

<IntAtt>:
<Ident:LexAtt>
<Ident:PhrAtt>
<Ident:SemAtt>

<ExtAtt>:
<Ident:SemAss>

<IntVal>:
<Atom>

<ExtVal>:

<Fest>

<None>:

无

None

NONE

特征结构<Fest>由一些特征组成，每个特征是一个“属性——值”对。

值为原子的特征，称为内部特征<IntFeat> ，所有内部特征括在方括号中，组成内部特征集合<IntFest>。 在方括号中，逗号表示“与”，分号表示“或”，“与”的优先级高于“或”的优先级。内部属性<IntAtt>是一个表示属性的标识符。内部值<IntVal>是一个原子。

值为其它特征结构的特征，称为外部特征<ExtFeat>，所有外部特征括在花括号中，组成外部特征集合<ExtFest>。外部特征集中不允许出现表示“或”的分号。外部属性<ExtAtt>是一个表示关联的标识符。外部值<ExtVal>是另外一个特征结构。

<None>是一个特殊的特征结构，除了它自身以外，它不能与任何其它特征结构合一。

5.2.3 树和森林

〈Forest〉:

〈Tree〉

〈HeadTag〉〈Tree〉

〈Forest〉〈Forest〉

〈Tree〉:

〈LabelItem〉

〈LabelItem〉〈〈WordItem_Or〉〉

〈LabelItem〉(〈SubForest〉)

〈SubForest〉:

〈SubTree〉

〈HeadTag〉〈SubTree〉

〈SubForest〉〈SubForest〉

〈SubTree〉:

〈LabelItem〉

〈LabelItem〉〈〈WordItem_Or〉〉

〈LabelItem〉(〈SubForest〉)

〈LabelItem〉:

〈Ident:LexCat〉

〈Ident:PhrCat〉

〈WordItem_Or〉:

〈WordItem〉

〈WordItem_Or〉|〈WordItem〉

〈WordItem〉:

〈Word〉

〈Word〉:

<Ident>

<HeadTag>:

!

树与树林是语言知识表示中一种常用的形式，用于表示语言成分的内部结构。树的结点对应于句子中的语言成分。

标记变量由 n 个百分号加上一个标记项组成，表示对应树中的第 n 个该标记项的结点。

- 关于中心子结点：
  1. 一个父结点可以没有中心子结点，或者有一个中心子结点；
  2. 父结点和中心子结点共享该树所对应语言的相关属性(即 RelMode=1 的属性)，但这种共享关系可以被切断（参见约束<Bind>）；

5.2.4 约束

<Bind>:

<Equation>

<Test>

<Bind> , <Bind>

<Equation>:

<IntVar> = <Atom>

<IntVar> = <IntVar>

<ExtVar> = <Fest>

<ExtVar> = <ExtVar>

<ExtVar> = <True>

<ExtVar> = <False>

<ExtVar> == <ExtVar> <At>

<ExtVar> != <ExtVar> <At>

<Test>:

IF <Bind> TRUE

IF <Bind> FALSE

IF <Bind> THEN <Bind> ENDIF

IF <Bind> ELSE <Bind> ENDIF

IF <Bind> THEN <Bind> ELSE <Bind> ENDIF

<BoolFunction>

<NumberFunction> <RelOperator> <NumberFunction>

<NumberFunction> <RelOperator> <Number>

<StringFunction> <RelOperator> <StringFunction>

<StringFunction> <RelOperator> <Ident>

<ExtVar>:

<RootVar>

<LabelVar>

<ExtVar>.<ExtAtt>

<IntVar>:

```
          <ExtVar>.<IntAtt>
 <RootVar>:
          $
 <LabelVar>:
          <LabelVarTag> <LabelItem>
          %<NodeFunction>
 <LabelVarTag>:
          %
          %%
          %%%
          %%%%
 <LabelItem>:
          <Ident:LexCat>
          <Ident:PhrCat>
 <At>:
          @ <Ident:IntAtt>
          @ <Ident:ExtAtt>
          <At> <At>
<BoolFunction>:
          MatchPattern (<LabelVar>, <Ident>)
          Score (<Ident:Number>)
<NumberFunction>:
          NumOfChild (<LabelVar>)
          GetLength (<LabelVar>)
          FindWord(<LabelVar>,<Ident:Word>,<Ident:POS>)
          FindString(<LabelVar>, <Ident>)
<StringFunction>:
          GetString(<LabelVar>)
          SubString(<LabelVar>,<Number:Position>,<Number:Length>,
                                                  <Number:TokenType>)
<NodeFunction>:
          GetMostLeftNode(<LabelVar>)
          GetMostRightNode(<LabelVar>)
          GetLeafNode(<LabelVar>,<Number:Position>)
 <RelOperator>:
         >
         <
         ==
         <=
         >=
         !=
```

约束用于对一棵树中各个结点的特征结构进行限制；

约束<Bind>由多个约束等式和条件约束表达式组成；

约束等式<Equation>是一个表示合一运算的等式，这种等式有八种形式：

1. 一个内部变量(IntVar)与一个原子(Atom)合一；
2. 一个内部变量与另一个内部变量合一；
3. 一个外部变量(ExtVar)与一个特征结构(Fest)合一；
4. 一个外部变量与另一个外部变量合一；
5. 一个外部变量与一个布尔值（True）合一，表示该变量已指向句中的语言成分；
6. 一个外部变量与一个布尔值(False)合一，表示该变量未指向句中的语言成分；
7. 强制两个外部变量共享（==）某些属性；
8. 强制两个外部变量不共享（!=）某些属性。

上面第 8 种 Equation 一般用于切断中心词节点跟它的父节点之间的属性继承关系。

条件约束表达式<Test>用于指明在一定条件下才需要进行的约束；

约束中的各类函数的参数及其返回值的说明如下：

（1） Bool 型函数

MatchPattern (<LabelVar>, <Ident>)
参数：<LabelVar>为树节点；<Ident>为标识符（字符串）
返回值：bool 型
功能：测试 LabelVar 所覆盖的字符串是否是 Ident。

Score (<Ident:Number>)
参数：Number 取值范围在 -5 到 5 之间
返回值：永为真（true）
功能：给一条规则设置主观评分。-5 表示该规则最不可靠，5 表示最可靠。

（2） Int 型函数

NumOfChild (<LabelVar>)
参数：<LabelVar>为树节点
返回值：int 型
功能：返回 LabelVar 所代表的节点的儿子节点个数

GetLength (<LabelVar>)
参数：<LabelVar>为树节点
返回值：int 型
功能：返回 LabelVar 所覆盖的叶子结点（词语）的个数。
如果<LabelVar>为叶子节点，则返回 1

FindWord (<LabelVar>,<Ident:Word>,<Ident:POS>)
参数：<LabelVar>为树节点；<Ident:Word>为字符串（词）；<Ident:POS>为字符串（词性标记）
返回值：int 型

功能：如果 <词语,词性标记> 在 LabelVar 所覆盖的字符串中，则返回该词语所在的位置值（从左向右开始匹配，位置从 1 开始计数），如找不到，则返回 0。

FindString(<LabelVar>, <Ident>, <Number:Position>)

参数：<LabelVar>为树节点；<Ident >为任意字符串；<Number:Position>为查找的起始位置。取值为 1 表示从左往右开始查找，取值为-1 表示从右往左开始查找。

返回值：int 型

功能：如果 Ident(字符串) 在 LabelVar 所覆盖的字符串中，则返回 Ident 所在的位置（从左向右开始匹配，位置从 1 开始计数），如找不到，则返回-1。

（3） String 型函数

SubString(<LabelVar>,<Number:Position>,<Number:Length>,<Number:TokenType>)

参数：<LabelVar>为树节点；

<Number:Position>表示起始位置，1 为从左端第一个 Token(词或字)位置开始向右搜索，-1 为从右端第一个 Token(词或字)位置向左搜索（该参数也可以取 1 之外的其他整数，比如 2，-2，等等，含义依此类推）；

<Number:Length>表示目标子串的长度；

<Number:TokenType>取值为 1 时，表示按词搜索，即前面的<Number:Length>理解为词长，按词计算个数；取值为 0 时，表示按字搜索，即前面的<Number:Length>理解为汉字字符个数；

返回值：string 型

功能：返回 LabelVar 所覆盖的字符串中从位置 Position 开始，长度为 Length(词长或字长)的字符串。

<Number:Position>的值为 0，或其绝对值超过当前 Token 串的总长度时，均返回空串。

<Number:Length>加上<Number:Position>的绝对值大于当前 Token 串的总长度时，取出从<Number:Position>开始到当前 Token 串一端的全部字符串。

如果上述 3 个<Number>参数省略，则返回<LabelVar>所覆盖的全部字符串。

示例：

函数式	功能
SubString(%vp)	返回 vp 整个字符串
SubString(%vp,2,4,1)	从 vp 第 2 个叶子节点（词）开始，向右取 4 个词，如果 4 超出 vp 的长度，则取出第 2 个词之后全部的词串。
SubString(%vp,-2,4,0)	从 vp 所覆盖的字符串右端往左第 2 个汉字位置开始，向左取 4 个汉字，如果 4 超出 vp 的长度，则取出从右端第 2 个汉字往左的全部字符串。

（4） Node 型函数

GetMostNode(<LabelVar>)

参数：<LabelVar>为树节点

返回值：node 型，相当于树节点<LabelVar>。
功能：返回<LabelVar>所覆盖的儿子节点中最左边的儿子节点。

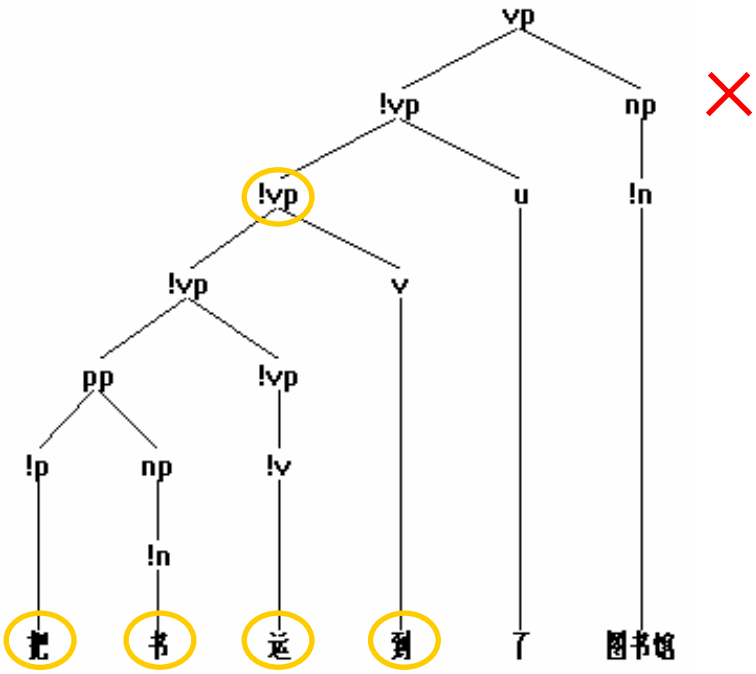
GetMostRightNode(<LabelVar>)
参数：<LabelVar>为树节点
返回值：node 型，相当于树节点<LabelVar>。
功能：返回 LabelVar 所覆盖的儿子节点中最右边的儿子节点。

GetLeafNode(<LabelVar>,<Number:Position>)
参数：<LabelVar>为树节点；<Number:Position>为<labelVar>节点所覆盖的叶子节点的序号（位置）
返回值：node 型，相当于树节点<LabelVar>。
功能：如果<Number>为正数，则返回 LabelVar 所覆盖的叶子节点中左边第<Number>个叶子节点，如果<Number>超出<LabelVar>叶子节点个数，返回最右边的节点；如果<Number>为负数，则返回右边第<Number>个叶子节点，如果<Number>超出<LabelVar>叶子节点个数，返回最左边的节点。

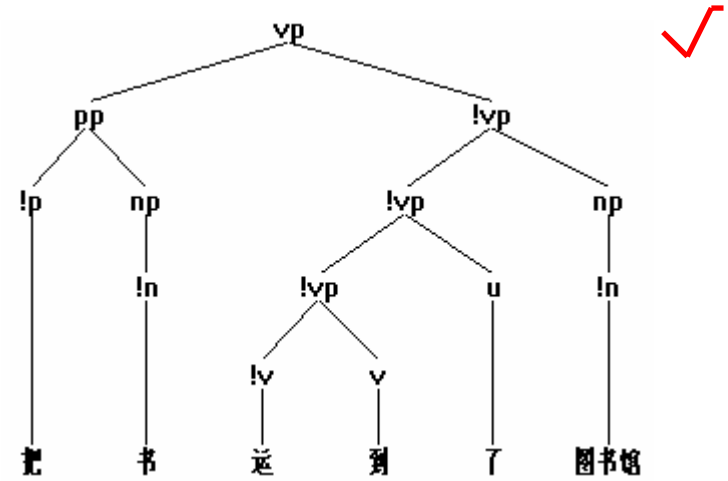
下面给出几个使用函数表达的<Bind>的实例。

例 1：“把书运到了图书馆”有下面两种句法结构分析结果。

甲图



乙图：

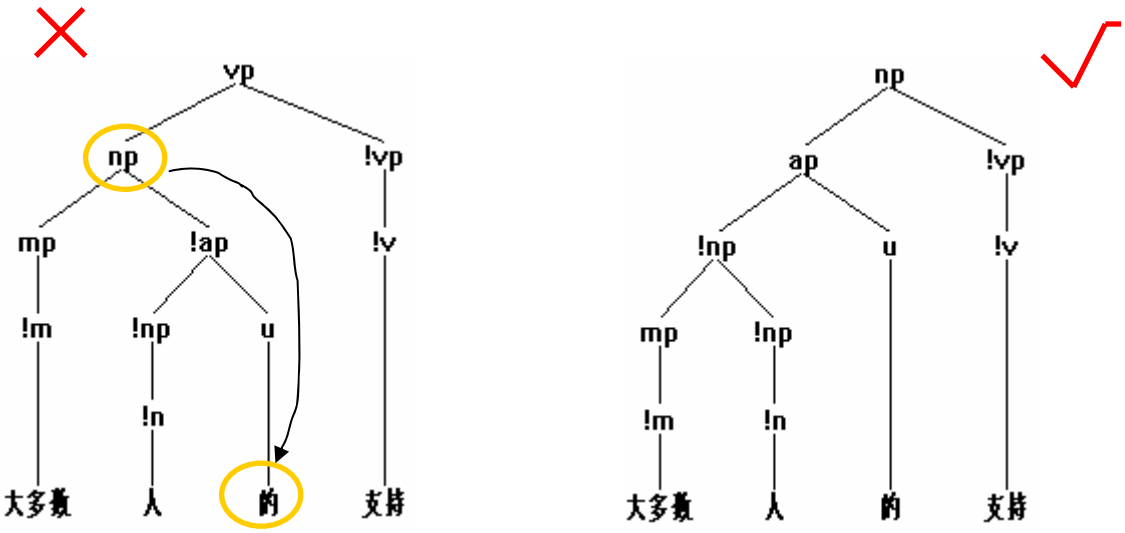


甲图是错误的结构分析，乙图是正确的。一般来说，“了”附着在较短的 vp 上，而不是附着在一个长 vp 上。因此，可以在规则中增加约束：

```
&& {vp1} vp -> !vp  u<了>  :: ..., IF #GetLength(%vp)>=4 FALSE, ...
```

在上面的规则中 IF 和 FALSE 之间有一个由函数式组成的 TEST。含义是如果 vp 覆盖的叶子节点个数大于等于 4，则这样的 vp 不能跟 u<了>组合。

例 2：“大多数人的支持”有下面两种句法结构分析结果。

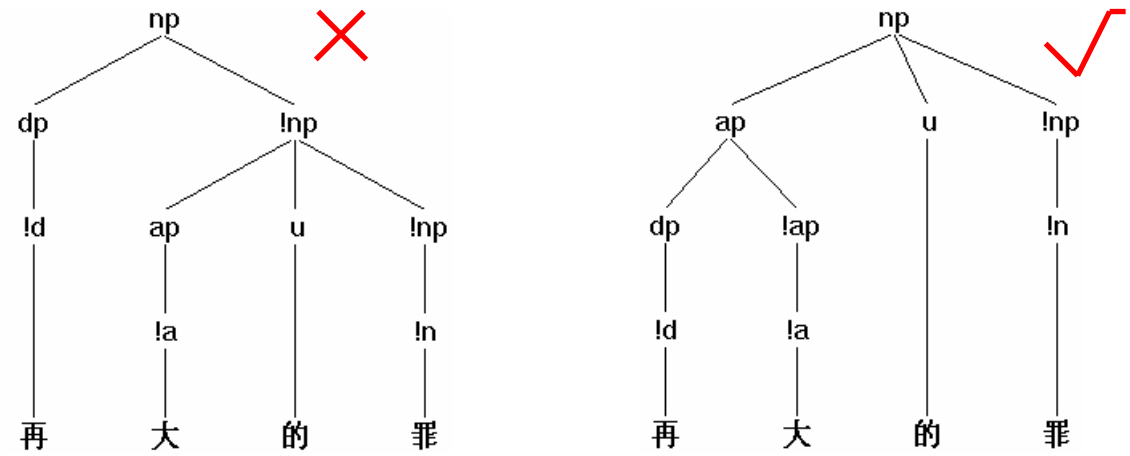


左边是错误的分析结果，右边是正确。由“的”字结尾的 np 一般不能作状语，跟 vp 构成一个更大 vp 短语。为此，可以在规则中增加如下的约束：

```
&& {vp2} vp -> np !vp  :: ..., IF %GetLeafNode(%np,-1).原形=的 FALSE, ...
```

上面规则中 IF 和 FALSE 之间调用了一个 NodeFunction。GetLeafNode(%np,-1)表示取 np 所覆盖的叶子节点右边第一个叶子节点。如果该叶子节点是“的”，则这样的 np 不能跟 vp 组成 vp。

例 3：“再大的罪”有下面两种分析结果。



尽管有些副词能直接跟名词组合（dp+np），比如“就两个人”，但显然 dp（“再”）跟 np（“大的罪”）的组合是不合语法的，为避免这种组合，同时又不排斥所有的 dp+np 组合，可以在规则中使用主观评分函数，来降低左边这种分析树的评分。

```
&& {np1} np->dp !np :: …, IF %dp.副名=否 THEN #Score(-5) ENDIF,…
```

上面规则中使用了 Score 函数来设置主观评分，即如果一个副词在词典中的“副名”属性取值为“否”（即该副词没有跟名词组合的能力），那么就赋给这条组合规则一个很低的分值（-5），表示这种组合是很不可靠的一种组合。

5.2.5 析句规则

缺省的析句规则文件名为 PRSRBASE （ Sentence-Parsing RuleBase ）。

```
<PrsRule>:
        && { <Title> } <PrsStruct>
        && { <Title> } <PrsStruct> <Note>
<Title>:
        <Ident>
<Note>:
        /* <Ident> */
```

```
<PrsStruct>:
        <PrsReductn> -> <PrsForest>
        <PrsReductn> -> <PrsForest> :: <PrsBind>
<PrsReductn>:
        <LabelItem>
<PrsForest>:
        <Forest>
        ^ <Forest>
<PrsBind>:
        <Bind>
        <Bind> <Note>
```

析句规则<PrsRule>主要由规则名<Title>和析句结构<PrsStruct>组成。
析句规则名<Title>是一个标识符，它在整个规则库中应唯一对应一条规则。
析句结构<PrsStruct> 由一个析句归约、一个析句树林与一个可选的析句约束组成。析句结构实际上也是一棵树，其根结点是析句归约，内部结构是析句树林，析句约束表示的是对析句结构这棵树的约束。
析句归约<PrsReductn>是一个标记项。
析句树林<PrsForest>由树林与可选的边界符(^)组成。析句树林前加 ^ 号，表示该树林组成的析句树不再参与树结构的组合。
注释<Note>一般在两条析句规则之间，或者放在两个<Bind>之间，以说明一条规则或约束适用的情况（实例）。

6 语言知识调试举例

6.1 词库知识调试举例

分析“爱打篮球的人一般个子比较高”其中划线部分的句法结构。
计算机产生的结果是：爱 + 打篮球的人
我们期望的结果是： 爱打篮球 + 的 + 人
打开 trace.txt 文件查看知道，计算机没有将 “爱打篮球”组合为一个 vp。
查看词库中“爱”条目：

```
$$ 爱
    **{v} v $=[谓词性主语:NO,系词:NO,助动词:NO,趋向动词:NO,补助动词:NO,形式动词:NO,准谓宾:NO,前名:NO,后名:NO,体谓准:体,双宾:NO,兼语句:NO,后动量词:动,后时量词:时,动趋:趋,趋向补语:~过,不:YES,没:YES,很:很,单作主语:是,单作谓语:是,单作补语:NO,动宾:动,形宾:形,小句宾:陈述,介宾的后:对,着了过:着|了|过,动介:在,与事宾语:与,格标 3:对,上:可,下去:是,起来:是,开:是,到:是,语义类:心理活动,配价数:2]{主体:[语义类:人],客体:[语义类:生物|集体|事理|事理|信息|空间]}
```

其中“体谓准”特征的取值为“体”，这意味着计算机当前的知识库中“爱”这个动词只能带体词性宾语（比如“人”），不能带谓词性宾语（比如“打篮球”），因此造成“爱打篮球”不能组合为一个 vp。
将词库中“爱”条目修改如下：

```
$$ 爱
    **{v} v $=[谓词性主语:NO,系词:NO,助动词:NO,趋向动词:NO,补助动词:NO,形式动词:NO,准谓宾:NO,前名:NO,后名:NO,体谓准:体|谓,双宾:NO,兼语句:NO,后动量词:动,后时量词:时,动趋:趋,趋向补语:~过,不:YES,没:YES,很:很,单作主语:是,单作谓语:是,单作补语:NO,动宾:动,形宾:形,小句宾:陈述,介宾的后:对,着了过:着|了|过,动介:在,与事宾语:与,格标 3:对,上:可,下去:是,起来:是,开:是,到:是,语义类:心理活动,配价数:2]{主体:[语义类:人],客体:[语义类:生物|集体|事理|事理|信息|空间]}
```

重新对上述例句进行分析，得到正确的句法分析结果。

6.2 规则库知识调试举例

分析“看机器的学生”的句法结构。
计算机分析得到的结果是：（1）vp(看机器) + u(的) + np(学生)
（2）vp(看) + np(np(机器) +u(的)+np(学生))
我们期望的结果是（1），而不是（2）。也就是说，计算机能分析得到正确的句法分析结果，但同时也产生了错误的结果。

打开 prsrbase.txt 规则库文件，查看相关规则：

```
&& {npdz14} np->ap !np :: $.内部结构=组合定中,$.定语=%ap,$.中心语=%np,
        %ap.内部结构=的字,      …
```

这条规则描述了 “ap(X 的) + np”短语组合的情况，对其中 X 部分语言单位的性质未做明确的限制，因而“看机器的”和“机器的”都可以跟“学生”组合，现在需要在规则中增加约束条件，只允许“看机器的”跟“学生”组合，而不允许“机器的”跟“学生”组合。

修改后的规则如下：

```
&& {npdz14} np->ap !np :: $.内部结构=组合定中,$.定语=%ap,$.中心语=%np,
        %ap.内部结构=的字,      …
        IF  %ap.中心语.cpcat=np,%np.语义类=人类  THEN  %ap.中心语.语义类=人类
ENDIF, …
```

在修改后的规则中，增加了对 ap 中 X（中心语）的约束，即在“X 的 Y”组合中，如果 Y 是指人名词，X 是名词，则要求 X 也是指人名词，这样，就排除了“机器的学生”这种错误的组合。

---　正文结束　---