一种基于汉语隐喻依存句法树的嵌入式树匹配算法

李剑锋[1], 杨 芸[1], 周昌乐[1,2*]
(1. 厦门大学信息科学与技术学院, 福建 厦门 361005;
2. 浙江大学语言与认知中心, 浙江 杭州 310028)

摘要: 提出了一种基于汉语隐喻依存句法树的嵌入式树匹配算法, 旨在发掘给定语句中所有可能存在隐喻关系的句法依存结构模式, 即从隐喻依存模式库中寻找出所有能够嵌入目标依存句法树的模式, 同时记录下对应的节点匹配结果. 模式库由预先存入的从大规模隐喻句中抽象出来的标准化依存句法树组成. 算法采用自上而下和自下而上回溯修正相结合的办法, 实现了依存模式的精确匹配. 测试结果表明, 本算法能够准确无误的找出依存模式库中能够嵌入目标树的所有规则树, 实现精确匹配的同时准确记录了每个节点的对应匹配结果, 算法保证了较高的运行效率.

关键词: 树匹配; 依存树; 规则匹配; 隐喻信息处理
中图分类号: TP 391 文献标识码: A 文章编号: 0438-0479(2008)04-0500-05

树型结构的模式匹配广泛运用于各个领域中, 国内外的研究人员做了许多有益的研究工作, 形成了一些理论和模型, 特别是用于 XML 检索的无序树近似匹配方面[1-5]. 然而已有的关于树的嵌入匹配算法并不直接适用于本文的研究. 本文所进行的研究是对表示成依存句法树(简称依存树)结构的汉语语句所可能含有的隐喻成分间的依存结构进行挖掘匹配. 主要过程是首先构建了一个含有一定数量的依存树模式库, 然后对于给定的目标语句寻找依存模式库中所有可嵌入到目标树中的依存模式, 要求进行精确匹配, 并且在完成匹配的同时还自动进行对匹配结果的标记工作.

本文在研究依存树结构特点的基础上, 借鉴无序树匹配的有关理论, 提出了一种采用自上而下和自下而上回溯修正相结合的方法的依存树嵌入匹配算法, 能够实现依存模式的精确匹配, 满足了隐喻自动识别系统所提出的要求.

1 基于隐喻依存关系的树的匹配

1.1 依存句法与依存句法树

依存树是在依存句法分析的基础上产生的, 句法分析所用的工具为哈尔滨工业大学信息检索研究室的依存句法分析系统[6]. 句法分析生成带依存句法标记的语句, 语句依存关系表示成可视图形如图 1 所示, 词语下方为词语的词性标记, 词语之间的依存关系用带

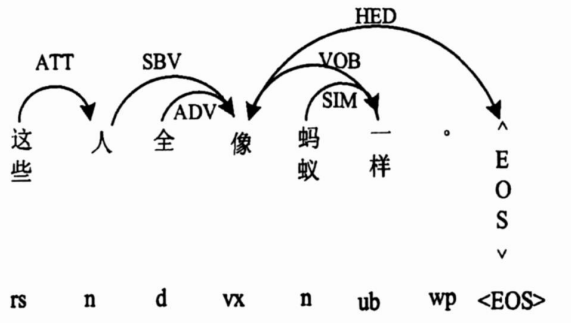

图 1 依存句法结构图
Fig. 1 Dependency structure

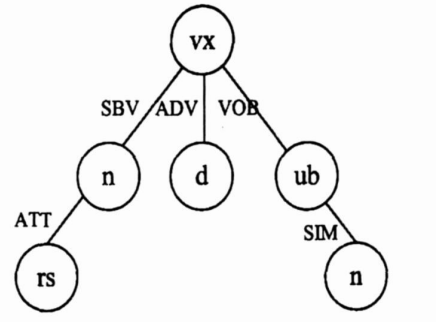

图 2 标准化依存句法树
Fig. 2 Dependency tree

属性的从被支配词指向中心词的有向弧表示.

本文将依存句法表示成以句子中心词为根节点的依存句法树结构, 并在此基础上进行相关的模式匹配. 由于本文所进行的模式匹配不涉及词义的分析, 仅关注句法结构, 因此本文对依存句法树进行了标准化处理, 只保留词性信息和依存关系, 并且由于中心词都是被支配词的父节点, 即图 1 中的有向弧由孩子节点指向父节点, 故在生成依存树的过程中去掉了箭头, 将每

收稿日期: 2007-09-14
基金项目: 国家自然科学基金(60373080)资助
* 通讯作者: dozero@xmu.edu.cn

一个待匹配的语句构造成如图 2 所示的依存树.

1.2　依存句法树相关定义

定义 1 依存句法树

依存句法树用 T 表示, $T=(V, E, \mathrm{root}(T))$, 其中 V 表示一个有限的节点集; $\mathrm{root}(T)$ 表示依存句法树的根节点, 为语句的中心语; E 表示边集, 是 V 上的一个二元关系, 满足反自反, 反对称, 可传递性. 如果 $(u, v) \in E$, 则称 u 节点是 v 节点的父节点, 记为 $u=\mathrm{father}(v)$. 如果两个节点 v_1, v_2, 有 $(v_1, v_2) \in E^+$ (其中 E^+ 是 E 的传递闭包), 则称 v_1 是 v_2 的祖先, 记为 $v_1=\mathrm{ancestor}(v_2)$. 节点 v 所在的高度记为 $\mathrm{height}(v)$. 本文所定义的依存树是一种无序树, 兄弟节点的左右顺序是无意义的.

定义 2 节点标签

依存句法树的每一个节点 v 都设置了一个标签, 标签由该节点所代表的词性和与其父节点之间的依存关系两部分构成, 分别用 $\mathrm{label}(v)$ 和 $\mathrm{dependency}(v)$ 表示. 对于 v 为根节点的情况, $\mathrm{dependency}(v)=$ "HEAD", 其中"HEAD"是语句中心语的标志, 用来表示根节点的依存关系.

树的匹配类型有多种, 本文所涉及的匹配类型为树的嵌入匹配. 两棵依存树之间的匹配可以归结为两棵树的节点之间的对应关系. 文献[2]给出了关于树的节点映射及其匹配的相关概念, 但它不适合本文中节点对应关系存在着多对多的情况, 故本文结合依存树实际情况给出依存树的节点关系及嵌入匹配的相关定义.

定义 3 不同节点

在依存句法树 $T=(V, E, \mathrm{root}(T))$ 中, 节点 $v, u \in V$, 且 $\mathrm{height}(v)=\mathrm{height}(u)$, $\mathrm{label}(v)=\mathrm{label}(u)$, $\mathrm{dependency}(v)=\mathrm{dependency}(u)$, $\mathrm{father}(v)=\mathrm{father}(u)$, 有可能 $v \neq u$.

例如图 3 中依存树 T_2 的 ⑤ ⑥两个节点属于不同节点.

定义 4 对应节点

设 $T_1=(V, E, \mathrm{root}(T_1))$, $T_2=(W, F, \mathrm{root}(T_2))$ 为两棵依存树, $v \in V$, $w \in W$, 满足以下条件:

(1)如果 v 为根节点, $\mathrm{label}(v)=\mathrm{label}(w)$;

(2)如果 v 为非根节点, $\mathrm{label}(v)=\mathrm{label}(w)$ 且 $\mathrm{dependency}(v)=\mathrm{dependency}(w)$,

则称 w 为 v 在 T_2 中的对应节点.

由定义 3 和 4, 可以得出依存树 T_1 中一个节点在 T_2 存在多个对应节点的可能. 如图 3 中 T_1 节点 ①在 T_2 中有对应节点 ②和 ③, ③在 T_2 中有对应节点 ⑤和 ⑥.

满足该定义的对应节点还不一定属于最终满足嵌入匹配条件后的对应嵌入匹配节点, 定义 5 将给出嵌入匹配节点的定义.

定义 5 嵌入匹配, 嵌入匹配节点

设 $T_1=(V, E, \mathrm{root}(T_1))$, $T_2=(W, F, \mathrm{root}(T_2))$ 为两棵依存树, 如果满足如下条件:

(1) $\exists w_t \in W$ 且 w_t 是 $\mathrm{root}(T_1)$ 在 T_2 中的对应节点;

(2)对 V 中所有的非根节点的节点 v, $\exists w \in W$ 使得: w 是 v 在 T_2 中的对应节点, 且 $w_t=\mathrm{ancestor}(w)$, $\mathrm{height}(v)$ 等于 w 到 w_t 的路径长度, 且 $\mathrm{father}(w)$ 是 $\mathrm{father}(v)$ 在 T_2 中的对应节点;

(3)满足上述条件(2)的 w 的个数应该不少于 T_1 中与 v 的高度、标签、父节点都相同的不同节点个数,

则称存在依存树 T_1 到 T_2 的嵌入匹配, 称 T_1 为被嵌入树, T_2 为嵌入树. 称满足以上条件的 w_t 和 $\mathrm{root}(T_1)$, w 和 v 互为嵌入匹配节点.

其中条件(1)保证了 T_1 的根节点在目标树中有对应节点 w_t. 条件(2)中 $w_t=\mathrm{ancestor}(w)$ 和 $\mathrm{height}(v)$ 等于 w 到 w_t 的路径长度, 保证了 w 节点在以 w_t 为根节点的子树中且它在子树中的高度和 v 节点在 T_1 所在的高度相同. 但是仅这些条件并不能完全保证这两个节点就是最终的嵌入匹配节点, 如图 4 中 T_1 中节点③和 T_2 中的节点 ⑤并不匹配, 故条件(2)中用条件 $\mathrm{father}(w)$ 是 $\mathrm{father}(v)$ 在 T_2 中的对应节点保证了匹配的正确性. 这样条件(1)和(2)保证了 T_1 中每个节点在 T_2 中都有相应的嵌入匹配节点. 由于有可能出现 T_1 中的多个节点都和 T_2 中同一个的节点相匹配, 导致 T_2 中相匹配的节点的数目还少于 T_1 中对应节点的数目, 如图 5 所示, 故加上条件(3)排除了这种情况. 图 3 即为依存树 T_1 到 T_2 的嵌入匹配情况, 其中 T_1 中节点 ①, ②, ③在 T_2 中的嵌入匹配节点分别为 ②, ④和 ⑤⑥.

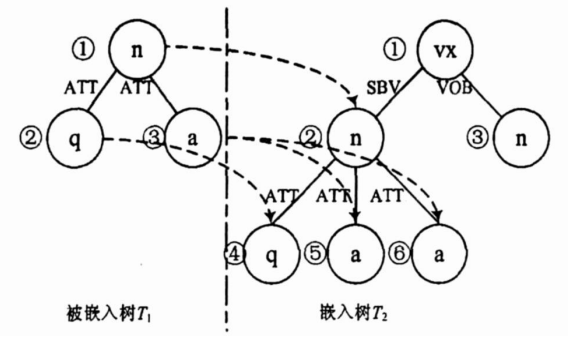

图 3　映射和嵌入匹配示例

Fig. 3　Examples of embedded matching and embedded matching nodes

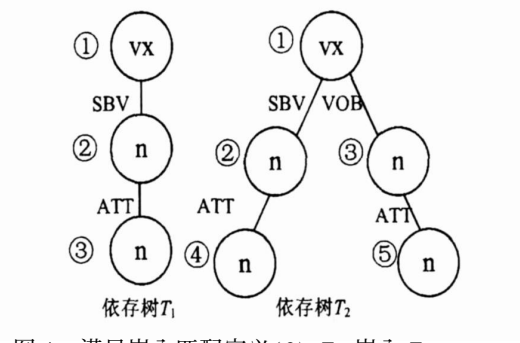

图 4 满足嵌入匹配定义(2), T_1 嵌入 T_2

Fig. 4 T_1 is embedded in T_2

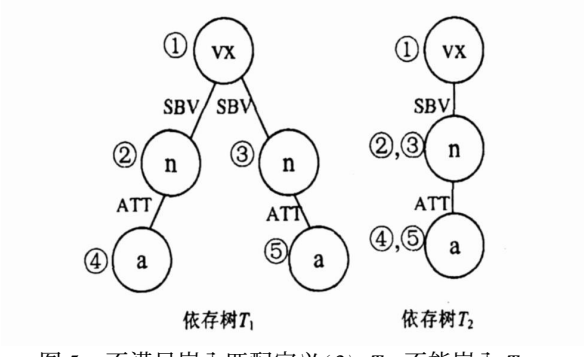

图 5 不满足嵌入匹配定义(3), T_1 不能嵌入 T_2

Fig. 5 T_1 is not embedded in T_2

1.3 隐喻句法依存模式库

依存句法模式匹配所用到的依存模式库为厦门大学艺术认知与计算实验室隐喻计算研究小组分析整理的不同类型隐喻依存句法树组成. 每一棵依存树即为一个隐喻依存模式, 这些模式都是最基本的隐喻依存关系, 模式之间不存在嵌套或者重复的关系. 根据我们的模式匹配思想, 可以存在模式库中的多个模式同时和一个目标语句相匹配. 目前使用的依存模式库规模为 32 个基本模式.

2 依存模式嵌入匹配算法

2.1 算法描述

为了便于后面的说明, 我们将依存模式库中的依存树称为规则树, 将要进行嵌入匹配的依存句法树称为目标树.

算法的目标是要找出所有依存模式库中规则树嵌入到目标树的情况, 并且需要记录每一种嵌入匹配情况下目标树中的嵌入匹配节点对应于依存模式库中某棵规则树中节点的标号, 用于进行匹配后的其他计算. 例如图 3 所示, 该规则树已经属于嵌入目标树的情况, 需要记录目标树节点 ②对应于规则树节点 ①, ④对应 ②, ⑤对应 ③, ⑥对应 ③.

算法的基本思想是对于目标树中以每一个节点为根的子树都要和依存模式库中的所有规则树进行是否发生嵌入匹配的判断, 过程如下: 对依存模式库中的规则树的节点从上至下进行层次遍历, 判断该规则树中的所有节点在这棵目标子树中是否都有对应匹配节点, 有则在一个临时结果记录单元中记录下目标树中每个节点对应于该规则树中节点的标号; 然后再按规则树中的叶子节点起从下往上进行回溯修正结果记录单元中记录的标记; 最后再对结果记录单元的结果进行整体判断是否满足嵌入要求.

2.2 算法的实现

算法相关数据结构如下: 依存树采用数组存放节点, 依存关系记录于孩子节点中, 这样数组中每一个节点包含以下数据: 词性标号 label, 依存关系 dependency, 父亲节点标号 father. 其中根节点中记录的依存关系为"HEAD", 父亲节点标号为 -1. 字符型结果标记记录数组: tempResult 和 modifiedResult, 它们的数组大小为目标树中节点的个数, 用于按照目标树中节点顺序存放该目标节点对应的嵌入匹配节点的所有标号, 即如果目标树节点 1 对应于某棵规则树中的节点 3 和 4, 则 tempResult[1] = "3, 4".

算法流程如下:

```
for 目标树的每一个节点 targetNode 为根节点的子树, 修改
    节点 targetNode 的依存关系为"HEAD", 并记录下
    targetNode 节点原依存关系;
    for 依存模式库中的每一棵规则树 ruleTree
        flag ◇ = 1, 用于判断规则树 ruleTree 中每个节点是否
          都在目标树中有匹配节点;
        for 规则树 ruleTree 中的按广度优先遍历的每一个节
            点 ruleNode
            判断是否有含有嵌入匹配节点模块;
            /* 判断节点 ruleNode 在目标树中以节点
              targetNode 为根的子树中是否有嵌入匹配节
              点, 有则在 tempResult 中记录对应匹配情况,
              返回真并继续; 无则 flag ◇ = 0, 返回假, 退出;
            */
        end for
        if (flag = 1) then
          回溯修正结果记录单元模块;
          /* 从规则树 ruleTree 的叶子节点起向上回溯,
            修正 tempResult 中的记录, 新的结果记录于
            modifiedResult 中; */
          最终判断是否发生嵌入匹配模块;
          /* 对 modifiedResult 中的记录按照规则树嵌入的
            标准判断是否发生嵌入匹配. 是则将 targetNode
            和规则树 ruleTree 在依存模式库中的编号和
            modifiedResult 作为最终结果值之一记录; */
        end if
```

end for

恢复 targetNode 节点原本依存关系;

end for

2.2.1 判断是否有嵌入匹配节点模块

该模块采用从上自下判断的方法确定规则树 ruleTree 中的节点 ruleNode 在目标树中以节点 targetNode 为根的子树中是否有嵌入匹配节点(实现规则树定义 5 之条件(2)的判断),有则设该嵌入匹配节点为 node,在 tempResult[node]中记录下 ruleNode 的标号. 由于是采用广度优先遍历,则如果规则树中某非根节点在目标树中有嵌入匹配节点,那么 tempResult 中必定记录了该非根节点的父节点的嵌入匹配情况,否则该节点就不可能有匹配节点. 实现步骤大致如下:

(1) 在目标树中以 targetNode 为根节点的子树中寻找到和规则树 ruleTree 中 ruleNode 节点所在层次,且词性标号和依存关系完全一样的节点 node;

(2) 如果 ruleNode 是根节点,则在 tempResult 中添加记录: 令 tempResult[node] 等于 ruleNode 的标号;

(3) 如果 ruleNode 不是根节点,则再判断 tempResult[node- > father]是否包含了 ruleNode- > father 的标号,是则在 tempResult[node]中添加上 ruleNode 的标号;

(4) 跳至步骤(1),直至找不到这种节点;

(5) 如果存在至少一个和规则树 ruleTree 节点 ruleNode 相匹配的节点则返回真,否则 flag ◇= 0,返回假.

2.2.2 回溯修正结果记录单元模块

tempResult 记录了目标树中的节点对应的规则树 ruleTree 中嵌入匹配节点的标号,但是这些记录存在着错误的情况,需要在本模块中从规则树的叶子节点起向上回溯加以修正,新的结果记录于 modifiedResult 中. 实现步骤大致如下:

(1) 在规则树 ruleTree 中寻找到一个叶子节点 leaf;

(2) 在 tempResult 中找到一个包含 leaf 标号的单元 resultNode, 令 modifiedResult[resultNode] 添加 leaf 标号,然后令 resultNode 等于目标树中 resultNode 号节点的父节点,leaf 也等于规则树 ruleTree 中它的父节点,重复上面的添加工作,直至到达规则树的根节点;

(3) 规则树 ruleTree 中是否还有叶子节点,有则跳至步骤(1),无则返回.

为了形象的表示此回溯过程,此处通过具体的例子来诠释此模块的功能. 如图 6 所示,经过本模块的修正去掉了 tempResult 中匹配错误的标号(其中目标树中所示标号为目标树中节点对应于规则树中的匹配节点的标号). 图 6(b)为目标树经过从上至下标注后 tempResult 中的结果,图 6(c)为目标树经过从下至上回溯,对 tempResult 中结果修正后 modifiedResult 中的结果.

2.2.3 最终判断是否发生嵌入匹配模块

有了前面两个模块实现的基础,还应该满足如下条件才最终满足嵌入的发生:

如果 modifiedResult 中某个单元 m 内记录的不同标记个数不止一个,设为 n 个,则 modifiedResult 中和 m 单元记录内容完全相同的单元个数必须大于等于为 n,即满足定义 5 中的条件(3).

如图 5(其中目标树中所示标号为目标树中节点对应于规则树中的匹配节点的标号)所示就不满足此条件,不属于树的嵌入匹配.

如果以上条件满足,则说明规则树嵌入了目标树中. 把目标树中子树根节点 targetNode 的标号、规则树的编号和 modifiedResult 作为结果之一记录下来,作为依存模式匹配之后的其他相关计算的依据.

3 结论及展望

本算法的时间复杂度为: O(TreeBankNum *

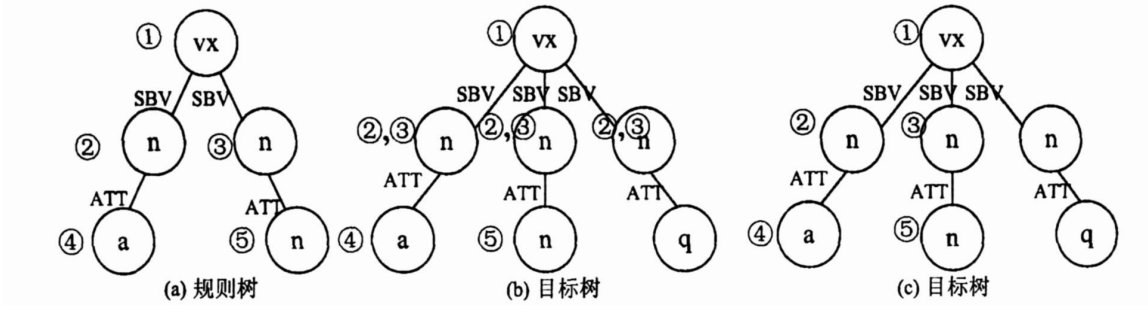

图 6　回溯修正结果记录单元模块功能示例

Fig. 6　Up-down finding and bottom-up amending

MaxNodeNum * MyTreeNodeNum^2), 其中 TreeBankNum 为依存模式库中规则树的数目, MaxNodeNum 为依存模式库中规则树所含的最大节点数, MyTreeNodeNum 为目标树中节点的数目. 依存模式库中规则树的数目目前为 32 个左右, 规则树最大节点数不超过 20, 目标句子中词语个数限制在 50 个以内, 所以算法保证了较高的运行效率. 经过对大量给定的目标语句进行测试, 本算法能够准确无误的找出依存模式库中能够嵌入目标树的所有规则树, 实现精确匹配的同时准确记录了每个节点的对应匹配结果.

本算法是一种精确匹配算法, 也可以运用到其他嵌入式树匹配相关的研究当中. 如果加以改造可以扩大适用范围, 比如可以根据节点的重要性赋予对应的权值, 改造成一种近似的嵌入式树匹配算法. 本算法存在着时间复杂度受目标树节点数目影响较大的局限性, 算法的时间复杂度还有待进一步的提高.

本文所开展的依存句法模式匹配算法研究是汉语隐喻计算研究[7-9]的一个初步环节, 即给定一个经过依存句法分析后的汉语语句, 在依存模式库中进行搜索找出所有与之满足嵌入匹配的模式, 并且记录该模式匹配情况下相应的匹配节点的对应情况. 这样的嵌入匹配寻找过程实际上就是将句子中部分成分及它们之间的依存关系独立出来, 作为单独处理单元, 过渡到下一步隐喻判定的环节当中, 进一步的研究工作我们正在开展过程当中.

参考文献:

[1] Kilpelainen P. Trees matching problems with applications to structured text database [D]. Helsinki: Department of Computer Science, University of Helsinki, 1992.

[2] Schlieder T. ApproXQL: design and implementation of an approximate pattern matching language for XML [D]. Berlin: Freie University, 2001.

[3] 徐如志, 钱乐秋, 程建平, 等. 基于 XML 的软件构件查询匹配算法研究[J]. 软件学报, 2003, 14(7): 1195- 1202.

[4] 贾晓辉, 陈德华, 严梅, 等. 基于刻面描述的构件查询匹配模型及算法研究[J]. 计算机研究与发展, 2004, 41(10): 1634- 1638.

[5] 郑仕辉, 周傲英, 张龙, 等. XML 文档的相似测度和结构索引研究[J]. 计算机学报, 2003, 26(9): 1116- 1122.

[6] 刘挺, 马金山, 李生, 等. 基于词汇支配度的汉语依存分析模型[J]. 软件学报, 2006, 17(9): 1876- 1883.

[7] Zhou C L, Yang Y, Huang X X. Computational mechanisms for metaphor in languages: a survey [J]. Journal of Computer Science and Technology, 2007, 22(2): 308- 319.

[8] 杨芸, 周昌乐, 王雪梅, 等. 基于机器理解的汉语隐喻分类研究初步[J]. 中文信息学报, 2004, 18(4): 31- 36.

[9] 李剑锋, 杨芸, 周昌乐. 面向隐喻计算的语料库研究和建设[J]. 心智与计算, 2007, 1(1): 142- 146.

An Embedded Tree Matching Algorithm Based on Metaphorical Dependency Structure

LI Jian-feng[1], YANG Yun[1], ZHOU Chang-le[1,2*]

(1. School of Information Science and Technology, Xiamen University, Xiamen 361005, China;
2. Center for the Study of Language and Cognition, Zhejiang University, Hangzhou 310028, China)

Abstract: This paper proposed an embedded dependency tree matching algorithm oriented to Chinese metaphor processing, aimed to find in a given sentence all the dependency relations that most probably occur in metaphors, that was to find out all the rule trees embedded in the given dependency tree, and at the same time record the matching results of corresponding nodes. The metaphoric dependency relations were derived from a large structured metaphor corpus, formalized as dependency trees and saved in a tree bank. The main process was top-down searching and bottom-up amending, and the test results showed that the algorithm could expectantly and efficiently find out accurate dependency relations and record the matching results.

Key words: tree matching; dependency relation; rule matching; metaphor processing