基于中文陈述句灵活语序的Lambek演算

刘冬宁 邓春国 滕少华 梁 路
(广东工业大学计算机学院 广州 510006)

摘 要 目前,自然语言处理已从句法、词法层面走向轻量级语义层面。针对中文陈述句的自然语言处理,传统Lambek演算无法解决中文陈述句灵活语序的问题,而现有的方法加入模态词、新连接词等后,又进一步增加了已经是NP-hard的Lambek演算的复杂性,因此并不适合计算机的相关处理。基于此,采用加标动词匹配的Lambek演算对中文陈述句灵活语序进行处理。加标动词匹配算法的时间复杂度低,使得计算机及其程序能有效地对中文陈述句灵活语序进行处理,并能通过Curry-Howard对应理论与λ-演算引入轻量级语义处理。

关键词 Lambek演算,中文陈述句,灵活语序,动词匹配
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.10.057

Lambek Calculus of Flexible Word Order of Chinese Based Statements

LIU Dong-ning DENG Chun-guo TENG Shao-hua LIANG Lu
(School of Computer, Guangdong university of technology, Guangzhou 510006, China)

Abstract Now natural language processing has shifted from syntactic/lexical level to lightweight semantic level. As for the natural language processing of Chinese narrative sentences, the traditional Lambek calculus cannot solve the problem of processing those Chinese statements with a flexible word order. And the present methods, such as adding modal words or new conjunctions, are not suitable for the relevant computer processing, because they will increase the complexity of the NP-hard Lambek calculus. In response, this paper used the Lambek calculus of marked verb matching to process the flexible word order of narrative sentences in Chinese. A low time complexity of the marked verb matching algorithm enables the computer programs to effectively process the flexible-word-ordered Chinese sentences, and also makes it possible to apply the lightweight semantic processing according to the corresponding Curry-Howard theory as well as lambda-calculus.

Keywords Lambek calculus, Chinese statements, Flexible word order, Match of verb

自然语言处理在日常生活中扮演着重要角色[1]。随着大数据时代的到来,半结构化的自然语言在计算机处理中有着更高的要求[2],如需要更快速、更准确地处理大量自然语言数据,并且需要涉及到自然语言中语义或轻量级语义的处理。传统的方法已难以满足大数据时代的要求,例如基于概率模型的统计方法难以准确地进行语义分析,而一些过于强调语义逻辑的方法又会使计算机进行处理的时间复杂度更高。与此同时,基于Lambek演算的自然语言处理有着许多优点,它是上下文无关的[3,4](context-free)、具有代数语义(Algebra)、关系语义(Relation)的模型,并能通过Curry-Howard对应理论与λ-演算引入轻量级语义[5]处理。而且由于Lambek演算没有收缩(contraction)、弱化(weakening)和交换律(exchange)这3条结构规则的特点[6],使得Lambek演算在自然语言处理中是可判定的。但同时,其因缺少交换律(exchange)规则而不能处理灵活语序。在前人研究中,为了处理灵活语序,分别在Lambek演算中加入了新模态词和新连接词。虽然这些方法在逻辑、数学和语言学范畴内形式完美,但其使原本已是NP-Hard的Lambek演算变得更为复杂,在程序处理中运行效率极低,因此并不可行。基于此,本文提出了加标动词匹配的Lambek演算,以解决中文陈述句灵活语序处理的问题,并且它有较高的执行效率。

1 Lambek演算与灵活语序分析

1.1 Lambek演算简介

Lambek演算由著名的加拿大学者Joachim Lambek于1958年提出,用于自然语言处理中的句法分析。Lambek演算是基于句法类型的演算,即句子片段中每一个词语都用一个类型表示,形成类型序列,然后通过类型序列来进行演算,从而判定句子的合法性。Lambek演算是上下文无关的(context-free),而且具有代数语义(Algebra)、关系语义(Relation)

到稿日期:2013-12-16 返修日期:2014-03-23 本文受国家自然科学基金项目(61272067,61104156,61370229),国家科技支撑计划课题(2013BAH72B01)资助。

刘冬宁(1979—),男,博士,副教授,主要研究方向为人工智能逻辑、数据库与协同计算,E-mail:liudn@gdut.edu.cn;邓春国(1988—),男,硕士生,主要研究方向为自然语言处理、数据挖掘;滕少华(1952—),男,博士,教授,主要研究方向为数据库与协同计算;梁 路(1980—),女,博士,副教授,主要研究方向为数据库与协同计算。

的模型，它的连接词只有 3 个，即积运算(product)“·”、左余运算(left residuation)“\”和右余运算(right residuation)“/”。Lambek 演算拥有 4 条规则(如表 1 所列)，并具有切割可消除性(cut-free)，这些优点使得 Lambek 演算在形式完美的同时运算简单，由此适用于程序的处理。

表 1　Lambek 演算规则

序号	规则
I	$(x/y)\cdot y\rightarrow x, y\cdot(y\backslash x)\rightarrow x$
II	$(x\backslash y)/z \Leftrightarrow x\backslash(y/z)$
III	$(x/y)\cdot(y/z)\rightarrow x/z, (x\backslash y)\cdot(y\backslash z)\rightarrow x\backslash z$
IV	$x\rightarrow y/(x\backslash y), x\rightarrow(y/x)\backslash y$

其中，规则 I 描述了类型消去演算，规则 II 描述了 Lambek 演算的结合性，规则 III 描述了 Lambek 演算的传递性，规则 IV 描述了演算中的类型是可以提升的，即从简单类型提升到复杂类型。

定义 1(剩余半群，Residuated Semigroup)[7]　一个剩余半群的结构为 $M=(M,\leqslant,\cdot,\rightarrow,\leftarrow)$，其中 $(M,\leqslant)$ 为偏序集，$(M,\cdot)$ 为一个半群，$\cdot$ 是 M 上的一个可结合的二元运算，$\rightarrow$ 和 $\leftarrow$ 是 M 上的二元运算，该结构满足：

$$a\cdot b\leqslant c \text{ iff } b\leqslant a\rightarrow c \text{ iff } a\leqslant c\leftarrow b \qquad \text{(RES)}$$

据(RES)，其推论如下：

推论 1

1) if $a\leqslant b$ then $ca\leqslant cb$ and $ac\leqslant bc$

2) if $a\leqslant b$ then $c\rightarrow a\leqslant c\rightarrow b$ and $a\leftarrow c\leqslant b\leftarrow c$

3) if $a\leqslant b$ then $b\rightarrow c\leqslant a\rightarrow c$ and $c\leftarrow b\leqslant c\leftarrow a$

设类型序列 $S=\{t_1,t_2,\cdots,t_n\}$，其中 t_k 为类型数组($k=1,2,\cdots,n$)。根据定义 1 和推论 1 以及表 1 中的规则，Lambek 演算的过程可归纳成以下 5 步：

Step1：列出句子或句子片段中每一个词语所有可能的类型，记于数组 t_k 中；

Step2：对类型序列中每一组类型进行组合，得到类型序列数组 $arrayS$；

Step3：选取类型序列数组中一个类型序列 $arrayS[i]=\{t_1[a],t_2[b],\cdots,t_n[c]\}$，其中 a、b、c 分别为类型数组 t_1、t_2、t_3 的一个取值；

Step4：对每个类型序列组合 $arrayS[i]$ 进行类型演算；

Step5：if 演算结果为 s，then 结束计算，else 跳到 Step3 直到所有类型序列组合都计算完。

例如句子“刘强爱看言情片”[8]，通过 Lambek 对其进行演算。首先对句子中每一个词列举出所有可能的类型，根据 Lambek 演算，名词“刘强”可以赋予类型 n，及物动词“爱看”可以赋予复合类型 $n\backslash s/n$，名词“言情片”可以赋予类型 n：

刘强　爱看　言情片

n　$n\backslash s/n$　n

由于每个词只有一个类型，因此只需对 $n\cdot n\backslash s/n\cdot n$ 进行 Lambek 演算。演算过程如下：

$$\dfrac{\dfrac{n\cdot n\backslash s/n}{s/n}(\mathrm{I})\cdot n}{s}(\mathrm{I})$$

经过演算最终得到类型 s，说明“刘强爱看言情片”是合法句子。

1.2　灵活语序分析

Lambek 演算的逻辑系统足够“弱”，即它只有有限的规则，而且没有收缩(contraction)、弱化(weakening)和交换律(exchange)这 3 条结构规则。在此基础上，Lambek 演算是可判定的，但其计算复杂度是 NP-hard 的[9]。

$$\frac{\Gamma,A,A\vdash B}{\Gamma,A\vdash B}\ \text{Contraction}$$

$$\frac{\Gamma\vdash B}{\Gamma,A\vdash B}\ \text{Weakening}$$

$$\frac{\Gamma,A,\Gamma',B,\Gamma''\vdash C}{\Gamma,B,\Gamma',A,\Gamma''\vdash C}\ \text{Exchange}$$

在此 3 条规则中，没有了弱化(weakening)规则使得 Lambek 演算是非单调的，并且使得演算中每个公式至少用一次，没有收缩(contraction)规则会使得演算中每个公式至多只用一次，没有了交换律(exchange)规则会使得 Lambek 演算是非交换的。但是，也正因为没有交换律(exchange)规则，使得 Lambek 演算无法处理自然语言中的灵活语序，如汉语的倒装句“言情片刘强爱看”。针对中文陈述句灵活语序的处理，现有的方式是加入模态词(Modality)，例如邹崇理[8]提出的方法、Moortgat[10]提出的方法和刘冬宁[11]提出的方法。但是加入模态词的方法又引出了一个问题，即虽然在数学上、逻辑学上和语言学上形式完美，但在程序计算中却不可行，这是因为 Lambek 演算已经是 NP-hard 的，再加入模态词，使得时间复杂度更高。

针对这个问题，我们提出了基于加标动词匹配的方法，使得 Lambek 演算不仅可以对中文陈述句的灵活语序进行处理，而且还能使其演算在程序计算机中可行[12]。

2　Lambek 演算与灵活语序分析

由于中文具有比较灵活的语序，因此句子中的主语、谓语和宾语的位置在一定情况下可以进行交换，但交换后的语序并不改变句子的基本语义，反而在某些情况下起了强调作用，在某些场合下正是需要这种灵活语序。而传统 Lambek 演算并不能处理这些灵活语序，现有的加入模态词的 Lambek 方法由于效率低也不能适应计算机的处理。

根据 Lambek 演算，陈述句“刘强爱看言情片”的话题式倒装句“言情片刘强爱看”和焦点式倒装句“刘强言情片爱看”，其句法类型无法通过计算得出正确句子 s，计算过程如下：

$$\frac{n\cdot\dfrac{n\cdot n\backslash s/n}{s/n}(\mathrm{I})}{?}$$

从句子的类型序列($n\cdot n\cdot n\backslash s/n$)及其 Lambek 演算中可以发现，动词($n\backslash s/n$)左右两侧的名词($n$)个数不匹配，因此需要将名词的位置进行置换。由于左侧的名词(n)有两个，因此需要对其加以标记[13]。

定义 2　设语言 $l=\{T,C,S,R\}$，其中，$T=\{s,n\}$ 为类型集，s 表示句子，n 表示名词，$C=\{\cdot,\backslash,/\}$ 表示连接词集，“·”为可结合的二元运算，“\”为左余(left residuation)运算，“/”为右余(right residuation)运算，S 为类型序列(其元素是 $T\cup C$ 的元素)，R 是演算规则集，规则如前文表 1 所列。

定义 3　设函数集 $F=\{r=f(\delta)\}$，$r=f(\delta)$ 为函数集中的

元素，其中 δ 为函数的参数，r 为函数返回值。

加标动词匹配 Lambek 演算的函数定义如表 2 所列。

表 2　加标动词匹配 Lambek 演算定义

函数	说明
Mark(S)	加标函数，参数 S 为类型序列，返回类型序列中每一个名词和动词加标之后的类型序列
Move(S)	类型移动函数，参数 S 为类型序列，对类型序列中与动词不匹配的名词和进行移动
Lambek(R)	使用规则集 R 对 S 进行 Lambek 演算

根据定义 2 和定义 3，加标动词匹配的 Lambek 演算为二元组 $L=\{l,F\}$。

其中 Mark(S)函数的伪代码如图 1 所示。

```
Input:类型序列 S
Output:加标后的类型序列
1. for (k=0;k < S. Length;k++) {
2.   if (S[k]为主动名词 n) S[k] =n_i;
3.   else if (S[k]为被动名词 n) S[k]=n_p;
4.   else if (S[k]为动词 n\s/n) S[k]=n_i\s/n_p;
5.   else if (S[k]为动词 n\s) S[k]=n_i\s;
6. }
7. return S[k];
```

图 1　Mark(S)函数伪代码

Move(S)函数算法思路如下：

Step1：从类型序列 S 中查找动词的位置，记为 pos；

Step2：if 动词左侧存在名词类型 n_i，then 将 n_i 移动并插入 pos 位置；

Step3：if 动词右侧存在名词类型 n_p，then 将 n_p 移动并插入 $pos+1$ 位置；

Step4：移动结束。

基于加标记的动词匹配 Lambek 演算 L 的流程如图 2 所示。

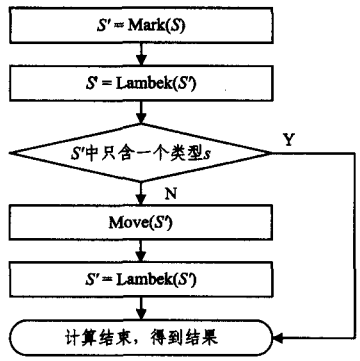


图 2　加标记的动词匹配 Lambek 演算的流程图

话题句“言情片刘强爱看”的演算过程如下：

$$\cfrac{\cfrac{\cfrac{n\cdot n\cdot n\backslash s/n}{n_p\cdot n_i\cdot n_i\backslash s/n_p}(\text{Mark})}{\cfrac{n_p\cdot s/n_p}{s/n_p\cdot n_p}(\text{I})}(\text{Move})}{\cfrac{}{s}(\text{I})}$$

同理，焦点句“刘强言情片爱看”的演算过程如下：

$$\cfrac{\cfrac{\cfrac{n\cdot n\cdot n\backslash s/n}{n_i\cdot n_p\cdot n_i\backslash s/n_p}(\text{Mark})}{\cfrac{n_i\cdot n_i\backslash s/n_p\cdot n_p}{s/n_p\cdot n_p}(\text{I})}(\text{Move})}{\cfrac{}{s}(\text{I})}$$

由此可见，此方法可以识别中文陈述句中的灵活语序。

3　加标动词匹配的 Lambek 演算的应用

加标动词匹配的 Lambek 演算不仅可以处理灵活语序如话题句“言情片刘强爱看”和焦点句“刘强言情片爱看”，而且还可以处理其他倒装句。现代汉语中倒装句的常见类型有：主谓倒置、宾语前置、定语后置和状语后置。中文陈述句中的倒装句分类如表 3 所列。

表 3　中文陈述句倒装句分类

倒装句类型	说明	举例	句法类型
主谓倒置	主语放在谓语后面	爱看言情片刘强	n\s/n·n·n
宾语前置	宾语放在主语前面	言情片刘强爱看	n·n·n\s/n
定语后置	定语形容词放在句子后面	一个姑娘走过来，漂亮的	n/n·n·n\s·n/n
状语后置	副词状语放在句子的后面	她听着音乐，静静地	s/n\s·n\s/n·n·n\s/n/(n\s/n)

对于句子“爱看言情片刘强($n\backslash s/n\cdot n\cdot n$)”，根据基于加标记的动词匹配 Lambek 演算，有：

$$\cfrac{\cfrac{\cfrac{n\backslash s/n\cdot n\cdot n}{n_i\backslash s/n_p\cdot n_p\cdot n_i}(\text{Mark})}{\cfrac{n_i\cdot n_i\backslash s/n_p\cdot n_p}{s/n_p\cdot n_p}(\text{I})}(\text{Move})}{\cfrac{}{s}(\text{I})}$$

因此句子“爱看言情片刘强”是合法的。

对于句子“爱看刘强言情片”，根据基于加标记的动词匹配 Lambek 演算，同有：

$$\cfrac{\cfrac{\cfrac{n\backslash s/n\cdot n\cdot n}{n_i\backslash s/n_p\cdot n_i\cdot n_p}(\text{Mark})}{\cfrac{n_i\cdot n_i\backslash s/n_p\cdot n_p}{s/n_p\cdot n_p}(\text{I})}(\text{Move})}{\cfrac{}{s}(\text{I})}$$

因此句子“爱看刘强言情片”也是合法的。但是，我们可以发现这个句子在语义上是不合法的，因为动词“爱看”紧接着名词“刘强”，所以我们认为句子的语意是“言情片爱看刘强”。因此，Move 函数并不能适应各种倒装句，所以我们需要修改 Move 函数。在移动时，应该根据倒装句的句式进行移动，而不是简单地匹配动词。

定义 4　$RM=\{rm\}$ 为规则集，其中 $r=(from,to)$ 为规则集 RM 中的规则，$(from,to)$ 是一个二元参数组，其中 $from$ 表示名词 n 在类型序列中的位置，to 表示移动到动词的相对位置。to 值为 -1 时表示将该类型删除[14]。

对于 4 种倒装句，其移动规则如表 4 所列。

表 4　Lambek 演算移动规则

规则	说明
$r_1=$(S. Length−1,Pos(v))	对应主谓倒置，移动时，将类型序列中位于类型序列末的主语名词移至谓语动词左侧，其中 S. Length−1 表示类型序列末的位置，Pos(v)表示动词 v 的位置
$r_2=$(0,Pos(v)+1)	对应宾语前置，移动时，将类型序列中位于类型序列前的宾语名词移至谓语动词右侧，其中 0 表示类型序列第 1 个位置
$r_3=$(S. Length−1,−1)	对应定语后置，移动时，直接将位于类型序列末的形容词定语删除，其中 −1 表示直接将定语类型删除
$r_4=$(S. Length−1,−1)	对应状语后置，移动时，直接将位于类型序列末的副词状语删除

根据定义 4，移动函数可改为 Move(S, $from$, to)，算法如下：

Step1：将 $S[from]$的类型插入至 $S[to]$的位置；

Step2：删除 from 处的类型 $S[from]$；

Step3：移动结束。

对于加入了移动规则的加标动词匹配的 Lambek 演算 L'，其演算流程如图 3 所示。

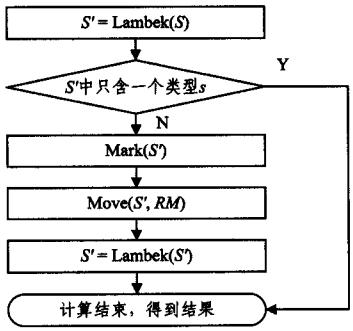


图 3　修改移动规则的加标记的动词匹配 Lambek 演算的流程图

对于句子“言情片刘强爱看”，根据修改移动规则的加标记的动词匹配 Lambek 演算，有：

$$\cfrac{\cfrac{\cfrac{\cfrac{n \cdot n \cdot n\backslash s/n}{n \cdot s/n}(\mathrm{I})}{n_i\backslash s \cdot n_p}(\mathrm{Mark})}{n_p \cdot n_i/s}(r_2)}{s}(\mathrm{I})$$

因此句子“言情片刘强爱看”是合法的。

基于加标动词匹配的 Lambek 演算不仅能对合法句子进行演算，而且对于不合法句子，依然能得出演算结果。对于句子“爱看刘强言情片”，根据加标记的动词匹配 Lambek 演算，有：

$$\cfrac{\cfrac{\cfrac{\cfrac{n\backslash s/n \cdot n \cdot n}{n\backslash s \cdot n}(\mathrm{I})}{n_i\backslash s \cdot n_p}(\mathrm{Mark})}{n_p \cdot n_i\backslash s}(r_1)}{?}$$

句子“非常刘强爱看言情片”的 Lambek 演算如下：

$$\cfrac{\cfrac{\cfrac{n\backslash s/n/(n\backslash s/n) \cdot n \cdot n\backslash s/n \cdot n}{n\backslash s/n/(n\backslash s/n) \cdot s/n \cdot n}(\mathrm{I})}{n\backslash s/n/(n\backslash s/n) \cdot s}(\mathrm{I})}{n\backslash s/n/(n\backslash s/n) \cdot s}(\mathrm{Mark})}{?}$$

由上述两个句子的演算过程可以看出，不合法句子在演算过程中，第一次 Lambek 演算不能得到句法类型 s，并且经过加标之后的类型序列不符合规则集中的规则，则该句子的类型序列不能进行移动操作，因此不能通过 Lambek 演算得到句法类型 s，即该句子是不合法的句子。由此可知，加标动词匹配 Lambek 演算不仅能对合法句子进行演算，而且也能对不合法句子进行演算，给出演算结果（不合法句子）。加标动词匹配 Lambek 演算的优点可以总结为以下 3 点：

（1）加标动词匹配的 Lambek 演算的计算机判定是可行的。传统的 Lambek 演算无法判定中文的灵活语序[16]，而采用加入了模态词的 Lambek 演算又因为其算法时间复杂度高而不能被计算机实际应用。基于加标记动词匹配的 Lambek 演算不采用模态词而是采用动词匹配的方法，其核心在于对类型序列进行移动操作，根据算法步骤可以发现，只要遍历一遍类型序列即可进行移动，假设移动规则集中的规则为有限条，则移动操作的时间复杂度为 $O(n)$，这并不增加原已是 NP-hard 的 Lambek 演算的复杂性。而传统的方法加入模态词的 Lambek 演算后在灵活语序的处理上，依然是基于 Lambek 演算，因此使得原已是 NP-hard 计算的复杂性变得更大，时间复杂度依然是 NP-hard。因此，基于动词匹配的 Lambek 演算在计算机实现上是更有优势的。除此之外，加标动词匹配算法还能处理灵活语序问题，因此加标动词匹配的 Lambek 演算在计算机实际应用中是可行的。

（2）加标动词匹配 Lambek 演算并不会因为句子的复杂性和灵活性而影响计算准确性，如口语化的文本、长句等。这是因为加标动词的方式类似于“照应语[15]”方式，而数据结构体现为堆栈，是前后照应的，如一个类型 n_p 只能和另一个 n_p 进行匹配。而且根据演算流程，先对句子进行一次不加标的 Lambek 演算，会使句子变短，复杂性降低，因此即使句子结构变得复杂，亦不影响其准确性。对于句子“舞台上漂亮的姑娘唱着一首首动人的歌曲，非常悦耳”，首先根据 Lambek 演算，为句子中每个词语赋予类型：“舞台上”、“漂亮的”、“动人的”、“悦耳”和“一首首”赋予形容词性类型 n/n，“姑娘”和“歌曲”赋予名词类型 n，“唱着”赋予及物动词类型 $n\backslash s/n$，“非常”赋予副词类型 $n/n/(n/n)$。因此其 Lambek 演算如下：

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{n/n \cdot n/n \cdot n \cdot n\backslash s/n \cdot n/n \cdot n/n \cdot n \cdot n/n/(n/n) \cdot n/n}{n/n \cdot n \cdot n\backslash s/n \cdot n/n \cdot n/n \cdot n \cdot n/n/(n/n) \cdot n/n}(\mathrm{I})}{n \cdot n\backslash s/n \cdot n/n \cdot n/n \cdot n \cdot n/n/(n/n) \cdot n/n}(\mathrm{I})}{s/n \cdot n/n \cdot n/n \cdot n \cdot n/n/(n/n) \cdot n/n}(\mathrm{I})}{s/n \cdot n/n \cdot n \cdot n/n/(n/n) \cdot n/n}(\mathrm{I})}{s/n \cdot n/n/(n/n) \cdot n/n}(\mathrm{I})}{s \cdot n/n/(n/n) \cdot n/n}(\mathrm{I})}{s \cdot n/n}(\mathrm{Mark})}{s \cdot n/n}(r_3)}{s}$$

因此该句子是合法的，由此可见，句子的复杂性、长短不会影响演算的准确性。

（3）加标动词匹配的 Lambek 演算不仅能够以比较快的速度处理中文灵活语序的 Lambek 演算，而且还可以处理普通陈述句的轻量级语义问题。例如使用传统的 Lambek 演算会将句子“言情片爱看刘强”判定为合法，而采用基于加标记动词匹配的 Lambek 演算则会将该句子判定为非法。其 Lambek 演算如下：

$$\cfrac{\cfrac{n \cdot n\backslash s/n \cdot n}{n_p \cdot n_i\backslash s/n_i \cdot n_i}(\mathrm{Mark})}{?}$$

因此“言情片爱看刘强”是不合法的句子。

4　实验验证

根据修改移动规则的加标记的动词匹配 Lambek 演算流程，先对待判定的句子中每一个词语进行类型赋值；然后对类型序列 S 进行 Lambek 演算，得到初次演算结果 S'，如果 S' 中只包含一个类型 s，则输出判定结果为合法句子，否则对 S' 进行加标操作；接着根据移动规则对加标后的 S' 进行移动操作；最后再进行 Lambek 演算，如果演算结果只包含一个类型 s，则输出判定结果为合法句子，否则输出非法句子。

其中程序中关键的演算过程的源代码如图 4 所示。

```
public void lambekCalCulus() { // 基于加标动词匹配的 Lambek 演算
    typeSeq.print(); // 输出
    lambek(); // Lammbek 演算
    // 如果序列不止含一个类型 s
    if (!(typeSeq.Size() == 1 && typeSeq.ToString().Equals("s"))) {
        typeSeq.Mark(); // 为类型序列加标
    }
    initRuleSet(); // 初始化规则集
    if (typeSeq.Get(0).types.Length == 1 && typeSeq.Get(0).types[0].GetCharText() == 'p') {
        move(ruleSet[1]); // 宾语前置
    }
    if (typeSeq.Get(typeSeq.Size() - 1).types.Length == 1
        && typeSeq.Get(typeSeq.Size() - 1).types[0].GetCharText() == 'i') {
        move(ruleSet[0]); // 主谓倒装
    }
    string lastType = typeSeq.GetTypeString();
    if (lastType.EndsWith("n/n")) {
        move(ruleSet[2]); // 定语后置
    }
    lambek(); // Lambek 演算
}
```

图 4　Lambek 演算部分主要源代码

Move 函数的源代码如图 5 所示。

```
private void move(Rule rule) {
    int from = rule.from;
    int to = rule.to;
    if (from > to && to >= 0) {
        Word word = typeSeq.Get(from);
        typeSeq.Delete(from);
        typeSeq.Insert(to, word);
        print();
    } else if (from < to && to >= 0) {
        typeSeq.Insert(to, typeSeq.Get(from));
        typeSeq.Delete(from);
        print();
    } else if (to ==-1) {
        typeSeq.Delete(from);
        print();
    }
}
```

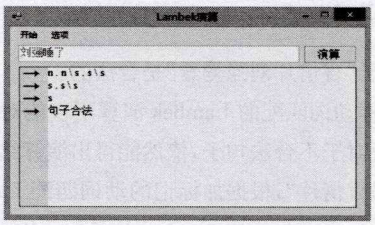
图 5　Move 函数的源代码

为了实现 Lambek 演算的程序判定，首先对实验所需的数据进行采集——选取汉语句子并对汉语句子进行分词，本实验采用了已经预处理的词汇表，即每个词语包含了词语本身以及对应的句法类型。为了使 Lambek 演算的实验更加全面，实验选取了合法句子“刘强睡了”、“言情片刘强爱看”以及“舞台上漂亮的姑娘唱着一首首动人的歌曲，非常悦耳”，第一句代表了汉语陈述句中最基本的主谓结构，第二句代表了汉语陈述句中常见的灵活语序，第三句代表了汉语中相对复杂、加了修饰的灵活语序句子，并且选取了反面例子（非法句子）“爱看刘强言情片”进行程序判定。表 5 中列出了部分词汇信息。

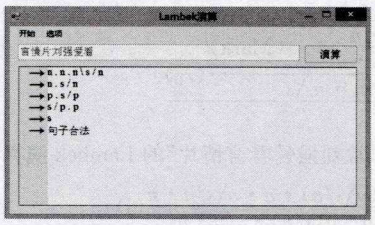
表 5　Lambek 演算词汇表

词语	类型	加标
刘强	n	n_i
言情片	n	n_p
爱看	n\s/n	$n_i\backslash s/n_p$
漂亮的	n/n	—
动人的	n/n	—
睡	n\s	—
非常	n\s/n/(n\s/n)	—
了	s\s	—

对句子“刘强睡了”、“言情片刘强爱看”、“舞台上漂亮的姑娘唱着一首首动人的歌曲，非常悦耳”和“爱看刘强言情片”进行程序判定，结果如图 6(a)－(d)所示。

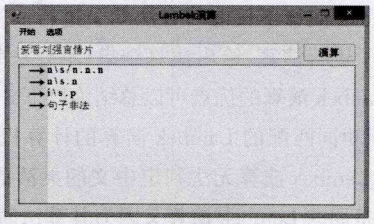


(a)



(b)

(c)



(d)

图 6　Lambek 演算程序判定结果图

由图 6(a)可以看出，正常语序的句子在演算的时候不需要对其进行加标和移动操作，直接通过一次 Lambek 演算即能得出结果。由图 6(b)可以看出，对于简单灵活语序，当第一次 Lambek 演算之后不能得出合法句子时，再对其进行加标和移动操作，然后再次进行 Lambek 演算，最终得出合法句子。由图 6(c)可以看出，对于复杂的灵活语序，首先通过第一次 Lambek 演算对句子中修饰成分进行演算，化简句子的类型序列，当演算无法得出合法句子时再对其进行加标和移动操作，从而再通过 Lambek 演算得出合法句子。由图 6(d)可以看出，对于不合法句子，第一次 Lambek 演算以及通过加标和移动操作后的第二次 Lambek 都无法得出合法句子时，输出该句子是非法的。由此可见，对于合法句子以及不合法句子，基于加标动词匹的 Lambek 演算在程序判定上均是可行的，而且有较高的执行效率。

结束语 基于 Lambek 演算的自然语言处理有着许多优点，它是上下文无关的、具有代数语义(Algebra)、关系语义(Relation)的模型，并能通过 Curry-Howard 对应理论与 λ-演算引入轻量级语义处理。但 Lambek 演算也存在一些限制，例如缺少交换律(exchange)规则，导致不能处理灵活语序。基于此，本文采用基于加标动词匹配的 Lambek 演算，其仅对 Lambek 演算中的类型序列加以标记，通过标记的类型进行匹配和移动，加标及移动操作的时间复杂度仅为 $O(n)$，因此能很好地被计算机应用。此外，基于加标记动词匹配的 Lambek 演算不仅能处理灵活语序，还能解决陈述句中基本语义错误的问题。在后续工作中，我们将针对中文陈述句的灵活语序引入轻量级语义演算，通过 Curry-Howard 对应理论，将 λ-演算作为语义分析工具引入到系统中，对应后的 Lambek 演算化为：

$$\frac{}{x:A\Rightarrow x:A}(\mathrm{id})$$

$$\frac{x\Rightarrow M:A\leftarrow B \quad Y\Rightarrow N:B}{X:Y\Rightarrow MN:A}(\leftarrow\mathrm{E})$$

$$\frac{X,x:A\Rightarrow B}{X\Rightarrow\lambda xM:B\leftarrow A}(\leftarrow\mathrm{I})$$

$$\frac{x\Rightarrow M:B \quad Y\Rightarrow N:B\rightarrow A}{X:Y\Rightarrow NM:A}(\rightarrow\mathrm{E})$$

$$\frac{x:A,X\Rightarrow B}{X\Rightarrow:\lambda xM:A\rightarrow B}(\rightarrow\mathrm{I})$$

$$\frac{X\Rightarrow M:A\cdot B \quad Y,x\Rightarrow A,y:B,Z\rightarrow N:c}{X,Y,Z\Rightarrow N[(M)_0/x][(M)_1/y]:C}(\cdot\ \mathrm{E})$$

$$\frac{X\Rightarrow M:Y\Rightarrow N,B}{X,Y\Rightarrow\langle M,N\rangle:A\cdot B}(\cdot\ \mathrm{I})$$

引入 λ-演算后，加标记的名词 n_i 和 n_p 可相应形式化 λ 函数，如 $\lambda_{n_i}\cdot n_i$ 和 $\lambda_{n_p}\cdot n_p$。通过 λ-演算及相应函数语义，进一步地丰富系统的内涵与形式演算，由此实现自然语言轻量级语义分析。

参 考 文 献

[1] Atkinson J, Matamala J. Evolutionary Shallow Natural Language Parsing[J]. Computational Intelligence, 2012, 28(2): 156-175

[2] Elias H. The Big Data Challenge: How to Develop a Winning Strategy[J]. 中国制造业信息化, 2012, 47(14): 53-55

[3] Lambek J. The Mathematics of Sentence Structure [J]. The American Mathematical Monthly, 1958, 3(65): 153-169

[4] Kozak M. Cyclic Involutive Distributive Full Lambek Calculus is Decidable[J]. Journal of Logic and Computation, 2011, 21(2): 231-252

[5] 郝身刚，张丽. 轻量级语义 Web 服务发现模型的研究[J]. 计算机工程与设计, 2011, 32(2): 450-456

[6] Surarso B, Ono H. Cut elimination in noncommutative substructurallogics[J]. Reports on Mathematical Logic, 1996, 30: 13-29

[7] Buszkowski W, Lin Zhe. Pregroup Grammars with Letter Promotions[J]. Language and Automata Theory and Applications, 2010, 6031: 130-141

[8] 邹崇理. 多模态范畴逻辑研究[J]. 哲学研究, 2006(9): 115-121

[9] 刘冬宁，汤庸，滕少华，等. 基于时态数据库的极小子结构逻辑系统[J]. 计算机学报, 2013(8): 1592-1561

[10] Bernardi, Moortgat. Continuation semantics for the Lambek-Grishin calculus[J]. Information & Computation, 2010, 208(5): 397-416

[11] 刘冬宁，汤庸，黄昌勤，等. 基于时态查询语言的并发 Lambek 演算及范畴语法[J]. 智能系统学报, 2009(6): 245-250

[12] Bitter C, Elizondo D A, Yang Ying-jie. Natural language processing: a prolog perspective[J]. Artificial Intelligence Review, 2010, 33(1/2): 151-173

[13] Lin Zhe. Modal Nonassociative Lambek Calculus with Assumptions: Complexity and Context-Freeness[J]. Language and Automata Theory and Applications, 2010, 6031: 414-425

[14] Buszkowski W, Lin Zhe, Moroz K. Pregroup Grammars with Letter Promotions: Complexity and context-freeness[J]. Journal of Computer and System Science, 2012, 7351, 78(6): 1899-1909

[15] Jäger G. Anaphora and Type Logical Grammar[M]. UiL OTS Working Paper, September 2001: 7-110

[16] Lin Zhe. Distributive Full Nonassociative Lambek Calculus with S4-Modalities Is Context-Free[J]. Logical Aspects of Computational Linguistic, 2012: 161-172