自然语言基本时态句型的 Lambek 演算[1]

刘冬宁 聂文龙[2]
（中山大学逻辑与认知研究所，广州 510275）

摘要：本文提出了一种基于时态语法分析的 Lambek 演算，其在原以 Lambek 演算为核心的范畴语法基础上，引入了时态类型。由于时态类型区别于语言学中诸如动词、名词、副词等语言结构类型，我们提出了并发类型演算（语言结构类型与时态类型并发处理），在文中给出了该演算系统的代数模型，并证明了相应的可靠性与完全性。我们将此演算系统称为 LC 演算系统，在文末给出基于 LC 系统的范畴语法 LCCG，并对其语义转换做了相应处理，该处理使之更易于计算机编程实现。由于在此演算系统中，只讨论了时态演算中最本质的三种时态对象，即现在时，过去时和将来时，因此我们称之为“基本时态句型演算”。

关键词：时态句型 Lambek 演算 并发类型 范畴语法

中国分类号： B81 文献标识码： A

1 引言

在计算机对语言的处理中，其编译过程一般分为词法分析、语法分析、语义分析、中间代码生成、中间代码优化、目标代码生成六个主要阶段。其头三个阶段分别对应于语言（自然语言、人工语言）的文法、语法和语义的分析，是编译全过程中的核心阶段，后三个阶段属于代码转换的语言翻译阶段，在六个阶段中还贯穿着表格管理和错误处理两个控制部分。在不同的分析阶段，具有不同的分析任务与阶段目标，各阶段进行的操作在逻辑上是紧密结合的。但仍然有某些阶段可能组合在一起工作，例如在语法和语义分析阶段同时使用到的 LR 分析树等；同时也会针对同一项因素所衍生的不同问题在各阶段分别进行分析，例如本文中将要谈到的时态问题。在语法和语义分析阶段，都需要针对该问题进行分析，其中在语法分析阶段，主要进行的是时态句型分析，在语义分析阶段则需对时态语义函数进行计算。

传统的编译理论对于人工语言的处理已经取得了不少成果，例如对高级语言 BASIC、PASCAL、C/C++、JAVA 等的处理。但对应于自然语言，却存在着较大的不足与缺陷。这主要体现于两方面，一方面为自然语言相对于人工语言体现出语言更大的“柔性”，不利于传统的建模分析；另一方面，由于在传统的编译理论中，语法分析和语义分析两阶段中采用了不同的文法作为分析工具，也使得语言的分析不能顺利得从语法过渡在语义。在语法分析中，其采用的文法为承继上一阶段词法分析所使用的上下文无关文法与正则文法（乔姆斯基文法，Chomsky 文法，上下文无关文法又称为 2 型文法，正则文法又称为 3 型文法），而在语义分析阶段使用的则为属性文法，并采用语法制导的语义分析过程进行语义分析与翻译。由于文法上缺乏有效的衔接，也导致了传统编译理论对自然语言的处理能力显得十分薄弱。

因此，自 Joachim Lambek 在 1958 年提出了 Lambek 演算[1, 2]的词法分析工具与 Richard

[1] 收稿日期：2006－10－10

基金项目：国家社会科学基金项目“超内涵逻辑”和教育部哲学社会科学研究重大课题攻关项目“基于自然语言的知识表达和推理系统”（项目编号：04JZD0006）。

[2] 作者简介：刘冬宁（1979－），男，博士研究生，主要研究方向：语言逻辑，人工智能逻辑，电子邮件：liu_dong_ning@yahoo.com.cn；聂文龙（1963－），男，博士，副教授，硕士生导师，主要研究方向：语言逻辑及其在计算机科学中的应用，现代哲学逻辑和智能计算机原理。

Montague 在 1970 提出了 Montague 语法[3,4]的内涵语义分析工具后，从上世纪 70 年代末开始，语言学、逻辑学与计算机界的学者们开始将上述理论作为自然语言的主流分析工具，并使用“Curry－Howard”对应理论[5,6]，将 Lambek 演算与 Montague 语法利用 λ -lamba 演算对应衔接起来，采用逻辑的方法，使得语法与语义的分析采用了同样的文法（Lambek 演算）作为背景，并利用范畴语法与类型论进行处理。另外，Lambek 演算还被学者证明了，以其为背景的范畴语法与 Chomsky 的上下文无关语法在表达能力上是弱等价的（只与 3 型文法构成的语法等价），这为计算机界研发自然语言的编译工具提供了有力支撑[7,8]。

然而，在语法分析方面，Lambek 演算与范畴语法的结合仍存在不足，就传统的 Lambek 演算而言，其只能处理一些基本的语法问题。目前，学者们均利用该系统的变形处理大量的语言学问题，如照应和省略语问题等[9]，又如本文重点论述的时态问题。

在时态处理中，通过传统的 Lambek 演算，我们知道：“John works.”和“John worked yesterday.”都是合法的句子，其中 John 的类型为 n，works 的类型为 n\s，yesterday 的类型为 s\s。但通过同样的类型演算，不难发现“John works yesterday.”也是合法的句子，而这是反事实的，其并非一个正确的句子，与语法不符，我们一般将它的错误称为“时态语法错误”。基于此，本文提出了一种基于时态语法分析的 Lambek 演算，其在原以 Lambek 演算为核心的范畴语法基础上，引入了时态类型。由于时态类型区别于语言学中诸如动词、名词、副词等语言结构类型，我们提出了并发类型演算（语言结构类型与时态类型并发处理），在文中给出了该演算系统的代数模型，并证明了相应的可靠性与完全性。我们将此演算系统称为 **LC** 演算系统，在文末给出基于 **LC** 系统的范畴语法 **LCCG**，并对其语义转换做了相应处理，该处理使之更易于计算机编程实现。

由于在此演算系统中，只讨论了时态演算中最本质的三种时态对象，即现在时，过去时和将来时，因此我们称之为“基本时态句型演算”。对诸如过去完成时、现在进行时、现在完成时等时态句型，在本演算系统中不予讨论。更特殊地，我们只对现在时和过去时进行了处理和对比，对于将来时与过去时的处理，原理上可以预测它们是大致相同的。这样的原因有两点，首先，尽管我们研究的对象语言是英语，但我们仍希望研究的是时态语句中最本质的运算，例如过去完成时、现在进行时、现在完成时等时态句型仅仅是作为特例在英语中出现；其次，这样的研究也是结合逻辑编程语言 Prolog 展开的，在 Prolog 语言中，同样只针对了现在时和过去时进行程序处理[10]，但其并没有将这些处理并入到句型演算中，而只是作为简单的参数作为处理，这也是传统 Prolog 语言处理时态问题的不足之处。

2 时态类型演算及各解决方案分析

2.1 时态句柄

基于传统 Lambek 演算，要解决基本时态句型的演算并不复杂。可行的一种方案是上在句子类型 s 上添加适当的时态标记，即形成时态句柄。例如我们用 s_{-1} 表示过去时的句子类型，用 s_1 表示将来时的句子类型，并仍然用 s 来表示现在时的句子类型，然后通过将这些句子类型添加入动词和副词相应的位置中，即能解决时态的问题。例如

（1）John　　worked　　yesterday.
　　n　　$n\backslash s_{-1}$　　$s_{-1}\backslash s$
（2）John　　works　　yesterday.
　　n　　$n\backslash s$　　$s_{-1}\backslash s$

通过句型演算，易知（1）是一个合法的句子，但（2）不是，这是因为 $n\backslash s^*\ s_{-1}\backslash s$ 无法做相应的运算，导致我们不能推出（2）是一个正确的句子。

时态句柄的解决方法是相当直观的，而且处理简便、轻易，然而一方面它属于绕过特征问题的本身对问题进行解决的方法，是 ad hoc 的，另一方面也没体现句型演算的相关特质（参见文献[1]中 Lambek 对代词的相似的分析方法）。但是其针对句子类型 s 所添加的时态标记的方法仍有一定可借鉴之处，由此衍生了我们的第二套解决方案。

2.2 时间副词的单一时态类型

根据时态句柄的解决方案，衍生了新的解决方法。在新的解决方法中，我们添加了表示时间副词的原始类型：t_{-1}、t_0 和 t_1。其中 t_{-1} 表示过去的时间，t_0 表示现在的时间（用于现在时），t_1 表示将来的时间，其时间轴如下所示：

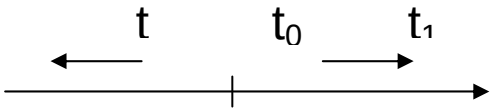

图 1 时间轴

针对此，我们可以把表示时间的副词表示为时间类型，例如 yesterday 可表示为 t_{-1} 类型，tomorrow 可表示为 t_1 类型，today 则表示成三种类型均可，而动词的过去式例如 worked 可表示为$(n\backslash s)/t_{-1}$。根据 Ajdukiewicz、Bar-Hillel 和 Buszkovski 的表示方法[11, 12, 13]，$(n\backslash s)/t_{-1}$ 又可表示为“$(n\rightarrow s)\leftarrow t_{-1}$)”，通过这样的表示方法，我们强调的是词项在时态句型演算中，动词由时态驱动的特性，在文末的举例中，我们将通过语义的解释阐述其合理性。此外，这种表示方法的优势还体现在赋予了动词“生存周期”，这将体现在语义分析执行阶段，在后面我们将进一步做阐述。

区别于原 Lambek 句型演算中 yesterday 的类型 s\s，我们将这样的表达方法称为“时间副词的单一时态类型”方法。这样的解决方法也能处理一些时态上的问题，例如

（3）John worked yesterday.
n $(n\backslash s)/t_{-1}$ t_{-1}

（4）John works yesterday.
n $(n\backslash s)/t_0$ t_{-1}

通过句型演算，我们仍然可以得知（3）是一个合法的句子，而（4）不是。

“时间副词的单一时态类型”的解决方法同样简单明了，但却衍生了一些新的问题，例如

（5）John played yesterday basketball.
n $(n\backslash s/n)/t_{-1}$ t_{-1} n

通过上述例子，按照 Lambek 的句型演算系统，我们得到了（5）是一个合法句子的结论，而这恰好与事实相反。

为什么会出现诸如（5）的问题呢？究其原因，我们发现由于时间副词使用了单一的时间类型，因此丧失了反映句子或作为句子组成部分的一些特性。因此，这样的方法也是不可取的。

2.3 时间副词的并发类型

根据上述两种解决方法的优劣，我们可以对时间副词的类型处理做总结如下：

1. 时间副词的类型必须体现时态的特点，即时态性；
2. 时间副词在体现时态性的同时，仍需保持其作为句子组成部分的副词特性；

3. 时间副词的时态性与副词性需同时体现在其此项上，并反映于句型演算中，其处理是并发的（注意“并发”与“并行”的区别）。

根据上述三点总结，衍生了第三套解决方案，我们称之为“时间副词的并发类型”方法，这也是我们最终采用的方法。在这种方法中，时间副词的类型表示为“t|(s\s)”（或(s\s)|t)）。其中，连接词“|”表示“并发”的意义，在“|”符号两侧的类型对于词项而言是“并发”拥有的。例如，对于时间副词 yesterday，其类型为“t_{-1}|(s/s)”，表示其不仅有时间类型“t_{-1}”，同时保持了副词类型“s\s”，同时在句型时，两种类型必须“并发”处理。

鉴于新连接词“|”的引入，传统 Lambek 演算必须进行相应转换才能对其进行处理。因此在介绍“时间副词的并发类型”的演算实例之前，我们先对进行该演算的 Lambek 演算的变形系统进行演算。相对于传统 Lambek 演算的 **L** 系统，我们将新演算系统称为 **LC** 系统（C for Concurrence）。

3 LC 演算系统

在本节中，我们将介绍基于经典 Lambek 演算 **L** 系统的变形系统 **LC** 系统。

在 **LC** 系统中，该系统的类型由原子类型 p_1、p_2…等构成，所有原子类型的集合表示为 Tp。类型的有穷矢列集合（有穷非空矢列集合）写作 Tp*（写作 Tp^{+}）。此外，符号Λ为空类型矢列， 大写字母 A，B，…表示类型。大写希腊字母表示有穷（可能为空）的类型矢列。在LC 系统中包含四个二元连接词 • 、\、/和|。为描述方便，我们使用 Ajdukiewicz、Bar-Hillel 和 Buszkovski 的表示方法[11, 12, 13]，将“\”表示为“→”，将“/”表示为“←”。

LC 演算的矢列形如Γ⊢A，在此Γ为非空类型矢列。

该演算系统的自然演绎系统如下所示：

$$\frac{}{A\,|-A}\ (\mathrm{id})$$

$$\frac{X\,|-A/B \quad Y\,|-B}{X,Y\,|-A}\ (/\mathrm{E}) \qquad \frac{X,A\,|-B}{X\,|-B/A}\ (/\mathrm{I})$$

$$\frac{X\,|-B \quad Y\,|-B\backslash A}{X,Y\,|-A}\ (\backslash\mathrm{E}) \qquad \frac{A,X\,|-B}{X\,|-A\backslash B}\ (\backslash\mathrm{I})$$

$$\frac{X\,|-A\bullet B \quad Y,A,B,Z\,|-C}{Y,X,Z\,|-C}\ (\cdot\mathrm{E}) \qquad \frac{X\,|-A \quad Y\,|-B}{X,Y\,|-\mathrm{A}\bullet\mathrm{B}}\ (\cdot\mathrm{I})$$

$$\frac{X\,|-A\leftarrow B \quad Y\,|-B\,|\,C}{X,Y\,|-A\bullet C}\ (|\,\mathrm{E}^{\mathrm{l}}) \qquad \frac{X\,|-A\,|\,B \quad Y\,|-B\rightarrow C}{X,Y\,|-A\bullet C}\ (|\,\mathrm{E}^{\mathrm{r}})$$

$$\frac{X\,|-A \quad X\,|-B}{X\,|-A\,|\,B}\ (|\,\mathrm{I})$$

显然，除(|E^l)、(|E^r)、(|I)三条规则外，其余规则是经典 Lambek 演算 **L** 系统的全部规则，它们组成原 **L** 系统。(|E^l)、(|E^r)、(|I)三条规则是为新添加的连接符“|”加入的消去和引入规则。在消去规则中，(|E^l)为左消去规则，(|E^r)为右消去规则。(|I)为引入规则。

定义 1：如果矢列Γ→A 在 **LC** 演算中是可推导的，则写作Γ| $_{\mathbf{LC}}$ -→A。

在不至于混淆的地方，我们将省略下标 **LC**。

定义 2：一个类型的长度被定义为类型中原始类型的出现的总次数。

$$\|p_i\|=1$$

$$\|A\cdot B\|=\|A\|+\|B\| \qquad \|A|B\|=\|A\|+\|B\|$$

$$\|A\backslash B\|=\|A\|+\|B\| \qquad \|A/B\|=\|A\|+\|B\|$$

相似的，对类型序列：$\|A_1\cdots A_n\|=\|A_1\|+\cdots+\|A_n\|$。

定义 3：原始类型的集合在一个类型中的出现定义如下：

$$Var(p_i)=\{p_i\}$$

$$Var(A\cdot B)=Var(A)\cup Var(B) \qquad Var(A|B)=Var(A)\cup Var(B)$$

$$Var(A\backslash B)=Var(A)\cup Var(B) \qquad Var(A/B)=Var(A)\cup Var(B)$$

4 LC 系统模型

与 **L** 系统大致相同，**LC** 系统的模型也为余格（residuated algbra）代数模型[14]。对于 LC 系统的余格代数、余格半群（residuated semigroup）和余格运算（residuated operation）描述如下。

定义 4：**LC** 的余格半群的结构为 $\mathtt{M}=(M, \circ, \rightarrow, \leftarrow, |, \leqslant)$，使得$(M, \circ)$是一个半群，$\leqslant$ 是 M 上的偏序关系，且→，←，|是 M 上的余格运算，并对所有 a，b，c，d∈M 满足以下关系：

（i）$b\leqslant a\rightarrow c$ iff $a\circ b\leqslant c$ iff $a\leqslant c\leftarrow b$

(ii) $a\leqslant b|c$ iff $(a\leqslant b \ \&\ a\leqslant c)$

(iii) $a\leftarrow b\circ(c|d)\leqslant a\circ d$ iff $(b\leqslant c \ \&\ c\leqslant b)$

(iv) $(a|b)\circ c\rightarrow d\leqslant a\circ d$ iff $(b\leqslant c \ \&\ c\leqslant b)$

该余格半群的力迫集（powerset）描述如下：

定义 5：$\mathtt{A}=(A, \bullet)$为半群，力迫集 P (A)中的运算○，→，←，| 定义如下：

$$X\circ Y=\{a\bullet b:\ a\in X,\ b\in Y\},$$

$$X\rightarrow Y=\{c\in A:(\forall a\in X)\ a\bullet c\in Y\},$$

$$X\leftarrow Y=\{c\in A:(\forall a\in Y)\ c\bullet a\in X\}$$

$$X\mid Y=\{a\in X\}\cap\{b\in Y\}$$

从上一节的$(\mid E^l)$、$(\mid E^r)$规 则，我们看到“|”与“•”在运算上有一定的相似之处。但从“|”连接符的力迫集看到，其力迫集是两个集合的交组成的一个集合，与“○”的双元序对力迫集相区别，这主要是因为(| I)规则引起的。同样，其定义与经典逻辑中的“∧”也相类似，它们都满足交换律，但是又相区别，因为“|”并不满足与以下相类似的规则：

$$\frac{X \mid -A \wedge B \qquad Y, A, B, Z \mid -C}{Y, X, Z \mid -C}(\wedge E)$$

与上述规则相类似的规则和演算，在 Lambek 演算中，由“•”连接符的“•E”规则来执行。实际上，“|”在当前演算中是一种对“∧”进行补充的连接符和演算，和“•”连接符类似，它们都具有“∧”的部分功能和特点。

定义 6：LC 系统的代数模型为一序对（M，μ），使得 M 是一个余格半群，μ为 M 上的类型赋值。序列 $A_1, \ldots, A_n \vdash A$ 在模型中是真的，如果$\mu(A_1) \circ \cdots \circ \mu(A_n) \leqslant \mu(A)$。$A_1, \ldots, A_n \vdash A$ 在模型中是有效的，如果它在所有（M，μ）都是真的。

定理 1（可靠性定理）：LC 演算系统关于余格模型（M，μ）是可靠的。

要证明可靠性定理，只需证明在该模型中即证明($|E^l$)、($|E^r$)、($|I$)是可靠的，其余规则均在文献[10]中有所说明。现证明($|E^l$)、($|E^r$)、($|I$)如下所示：

证明($|I$)：

设 X⊢A 且 X⊢B，

∴对∀x∈X, ∀a∈A, ∀b∈B, x⩽a 且 x⩽b，据定义 4－(ii)，x⩽a|b

∴据定义 4－(ii)，x⩽a|b

∴X⊢A|B

证明($|E^l$)：

设 X|-A←B 且 Y→B|C，

∴对∀x∈X，∀a∈A，∀b∈B，据 4－(i)有，x⩽a←b

对∀y∈Y, ∀b∈B, ∀c∈C, y⩽b|c

∴据定义 4－(iii), x • y⩽(a←b) • b|c ⩽a • c

据传递性 x • y ⩽ a • c

∴X, Y⊢A • C

证明($|E^r$)：

使用定义 4－(iv)，与证明($|E^l$)类似。

故可靠性定理得证。◄

定理 2（完全性定理－Product-free）：LC 演算系统关于余格模型（M，μ）是完全的。

文献[13，15]中给出了关于经典 Lambek 演算的完全性证明，其中文献[15]已证明 Lambek 演算是完全的。因为 **LC** 系统是在经典 Lambek 演算上添加了“|”连接符和相关规则，所以我们只需证明“|”的完全性。因此，我们采用的是文献[13]中 product-free 的证明方法，其证明方法类似于典范模型的证明方法。现证明如下：

我们设定一个序列$\Gamma_0 \vdash A_0$，令 S 表示该序列中所有公式的子公式集合。显然，S 是一个

有穷集，S 上的语言为 S*的子集。定义赋值μ如下所示：

$$\mu(p)=\{\Gamma\in S^*:\ \Gamma\vdash_{LC} p\}$$

我们需要证明的是：

$$（☆）\ \mu(A)=\{\Gamma\in S^*:\ \Gamma\vdash_{LC} A\}$$

在此，我们只需要证明 A=B|C 的情况，其他情况的证明如文献[13]。在不置于混淆的情况下，符号“⊢”的下标“**LC**”符号将在下文中省略。

证明μ (B|C)＝{Γ∈ S*：Γ⊢ LC B|C }：

1）左⇒右：

设Γ∈μ (B|C)，∴Γ∈μ (B) ∩μ (C)，

∴Γ∈μ (B) 且Γ∈μ (C)，

据归纳假设，Γ⊢B 且Γ⊢C，

∴据(| I)，Γ⊢B|C。

2）右⇒左：

设Γ⊢B|C，∴Γ⊢B 且Γ⊢C，

据归纳假设，Γ∈μ (B) ∩μ (C)

∴Γ∈μ (B) ∩μ (C)，

∴Γ∈μ (B|C)。

故μ (B|C)＝{Γ∈ S*：Γ⊢ LC B|C }得证，即μ (A)＝{Γ∈ S*：Γ⊢ LC A }得证。

下证完全性：设$\Gamma_0\nvdash A_0$，据（☆），$\Gamma_0\notin f(A_0)$，又∵$\Gamma_0\vdash\Gamma_0$，再据（☆）$\Gamma_0\in f(\Gamma_0)$，∴$f(\Gamma_0)\nsubseteq f(A_0)$，∴$\Gamma\nvDash A_0$。

故完全性定理得证。◄

5 LC 范畴语法及语义表达

在本节中，将给出 **LC** 范畴语法，并使用“Curry－Howard”对应理论[5,6]，将 **LC** 演算与应用于语义计算 λ -lamba 演算对应衔接起来，采用逻辑的方法，使得语法与语义的分析采用了同样的文法（LC 演算）作为背景，并利用范畴语法与类型论对时态句型的演算进行处理。

首先，在不至于混淆的地方，我们用集合 **B** 表示原 **L** 演算中的基本范畴，例如表示名

词的 n 类型和表示句子的 s 类型等。用集合 **T** 表示时间范畴，包含了分别表示将来时、现在时和过去时的 t_{-1}、t_0 和 t_1。并参照文献[9]，给出 **LC** 演算系统的基本范畴定义，**LCCG** 范畴语法的定义等。

定义 7：LC 演算系统的基本范畴定义

集合 **Bc＝B∪T** 表示 **LC** 系统的基本范畴集合。**CAT（Bc）**为一最小集，使得

1. **Bc** ⊆ **CAT(Bc)**；
2. 如果 A, B ∈ **CAT(Bc)**，则 A/B∈ **CAT(Bc)**；
3. 如果 A, B ∈ **CAT(Bc)**，则 A\B∈ **CAT(Bc)**；
4. 如果 A, B ∈ **CAT(Bc)**，则 A • B∈ **CAT(Bc)**；
5. 如果 A, B ∈ **CAT(Bc)**，则 A | B∈ **CAT(Bc)**；
6. 此外，**CAT(Bc)**不在包含任何其他元素。

定义 8：LTCG 范畴语法的定义

给定字母表Σ。一个 **LCCG** 范畴语法 **G** 是一个三元组<**Bc，LEX，S**>。其中有穷集 **Bc** 定义如上，**LEX** 是一个$\Sigma^{+}\times$ **CAT(Bc)**上的一个有穷子关系，**S** 是 **CAT(Bc)**的一个有穷子集（指定元范畴，the sesignated categories）。

该语法上的语言定义如下。

定义 9：G＝<**Bc，LEX，S**>是字母表Σ上的一个 **LCCG** 语法。则对α∈L(**G**) iff 存在 a_1，…，$a_n\in\Sigma^{+}$，A_1，…，A_n ∈ **CAT(Bc)**，且 S∈**S** 使得：

1. $\alpha = a_1\cdots a_n$，
2. 对所有 i 使得 $1 \leqslant i \leqslant n$：<$a_i$，$A_i$>∈**LEX**，且
3. $\vdash A_1, \cdots, A_n \Rightarrow S$。

根据“Curry－Howard”对应理论，我们给出 LCCG 语法对应的语义演算系统的 **LC** 标号自然演绎系统如下所示：

$$\frac{}{x:A\,|\!-x:A}\quad(\text{id})$$

$$\frac{X\,|\!-M:A/B \quad Y\,|\!-N:B}{X,Y\,|\!-MN:A}\;(/\,\text{E})\qquad\qquad \frac{X,\mathrm{x}:A\,|\!-B}{X\,|\!-\lambda xM:B/A}\;(/\,\text{I})$$

$$\frac{X\,|\!-M:B \quad Y\,|\!-N:B\backslash A}{X,Y\,|\!-NM:A}\;(\backslash\,\text{E})\qquad\qquad \frac{x:A,X\,|\!-B}{X\,|\!-\lambda xM:A\backslash B}\;(\backslash\,\text{I})$$

$$\frac{X\,|\!-M:A\bullet B \quad Y,\mathrm{x}:A,\mathrm{y}:B,Z\,|\!-N:C}{Y,X,Z\,|\!-N[(M)_0/x][(M)_1/y]:C}\;(\cdot\,\text{E})$$

$$\frac{X \mid -M : A \quad Y \mid -N : B}{X, Y \mid - < M.N >: \mathrm{A} \bullet \mathrm{B}} \ (\cdot \mathrm{I})$$

$$\frac{X \mid -M : A \leftarrow B \quad Y \mid -N : B \mid C}{X, Y \mid -MN : A \bullet C} \ (\mid \mathrm{E}^{\mathrm{l}})$$

$$\frac{X \mid -M : A \mid B \quad Y \mid -N : B \rightarrow C}{X, Y \mid -NM : A \bullet C} \ (\mid \mathrm{E}^{\mathrm{r}})$$

$$\frac{X \mid -M : A \quad X \mid -N : B}{X \mid - < M, N >: A \mid B} \ (\mid \mathrm{I})$$

根据上述($\mid$ I)、($\mid E^{l}$)、($\mid E^{r}$)规则在的 **LC** 标号自然演绎系统中的表示方法，我们进一步看到“|”连接词与“•”和“∧”在标号系统中的的相似和区别之处。

至此，我们为处理时态句型所做的准备工作已完成。下面将利用该标号自然演绎系统对时态句型进行语法和语义上的推导与演算。在该处理中，仍然以过去时为例进行处理，并分别以不及物动词和及物动词的处理进行示例。为简便起见，我们仍采用 Lambek 式的“\”和“/”的写法，同时在不至于混淆的地方，省略了演算所使用较简单的规则的书写。详例如下：

（6）John played yesterday.

n $(n\backslash s)/t_{-1}$ $t_{-1} \mid (s\backslash s)$

$$\frac{played}{(n \backslash s / t_{-1})} \qquad \frac{yesterday}{t_{-1} \mid (s \backslash s)} \mid \mathrm{I}$$

$$\frac{John}{n} \qquad \frac{played' \qquad yesterday'}{n \backslash s} \mid \mathrm{E}^{\mathrm{l}}$$

$$\frac{John' \qquad played'\ yesterday'}{s}$$

played'yesterday' John'

在上例中带符号“ ’ ”的，是对应词汇、短语和句子的语义解释，通过该解释，能使用 Montague 语法对其进行演算和知识表示，在语法分析和语义分析的转换过程中，起到了良好的过渡作用。在下例中，我 们会进一步看到该语义的应用。

（7）John played basketball yesterday.

n (n\s/n)/t_{-1} n t_{-1}| (s\s)

$$\frac{played}{(n\backslash s/n)/t_{-1}} \qquad \frac{\cdot}{t_{-1}}1$$

$$\frac{\dfrac{played' \qquad T}{n\backslash s/n}/E \qquad \dfrac{basketball}{n}}{\dfrac{played'T \qquad basketball'}{n\backslash s}}$$

$$\frac{played'Tbasketball'}{n\backslash s/t}/I,1 \qquad \frac{yesterday}{t_{-1}|(s\backslash s)}|\mathrm{I}$$

$$\frac{\dfrac{John}{n} \qquad \dfrac{\lambda T.played'Tbasketball' \qquad yesterday'}{n\backslash s}|\mathrm{E}^{l}}{\dfrac{John' \qquad played'yesterday'basketball'}{s}}$$

played'yesterday'basketball' John'

在上例中，我们重点标出了两个规则，分别是（/E）和（/I）。这是为了重点标出所使用的假设前提 1 的使用方法。假设前提的方法在类型逻辑语法的使用类似于乔姆斯基的 GB 系统[16]。假设前提的推理方式在类型逻辑语法中仅限于对高阶类型词项的运用，在本例中词汇“played”的类型为“(n\s/n)/t_{-1}”，其类型为高阶类型，详见文献[9]。

在上两例中，各分别在词汇 yesterday 上运用了一次| I 规则。这表明 yesterday 不仅有时间属性，也有副词属性。在运算中，两种属性是并发地进行运算的。

在上两例中，还各分别运用了一次| E^{l} 规则。例如在第一例的运算中，我们省略了一步：(n/s)/t_{-1} • t_{-1}|(s\s)⊢(n/s) • (s\s)，在这一步中，利用| E^{l} 规则消去了 t_{-1} 类型。其直观上的解释为：类型 t_{-1}|(s\s)拥有两个并发类型，在类型(n/s)/t_{-1} 的消去运算中，使用的是 t_{-1} 的类型属性。但因为类型 t_{-1}|(s\s)拥有两个并发类型，现在只用了一个，所以在演算中仍需要保留了另一个并发类型属性(s\s)，并运用于其后的演算。这样的演算是适用于时态句型演算这类既要考虑句子结构属性，又要考虑时态属性的并发运算的。

最后值得注意的是，在第二例演算的语义结果“*played'yesterday'basketball' John'*”中，动词的语义解释 played’后面紧跟着一个时间的语义解释 yesterday’。这是使用 λ-lamba 演算代入的结果。如果使用经典 Lambek 演算的 **L** 系统进行演算，得到的语义解释则为“*yesterday'played'basketball' John'*”。我们认为从程序实现的角度来看，前者的语义解释更为合理。因为 *played'yesterday'*表示在一个 played’的动作函数后面紧跟着它的执行周期（函数的生存周期）yesterday’，对于程序语义和程序的实现，比 *yesterday'played'*更直观与易于实现。而且 *played'yesterday'*的解释顺序也与我们的动词的类型(n→s)←t 的顺序相吻合。

至此，我们利用“时间副词的并发类型”的方法成功解决了时态句型的演算问题。在这种方法中，（2）、（4）和（5）这样的病句将不能得到成功的演算，而如（6）和（7）这样的句子则能得到成功的演算和满意的结果。

6 结论

本文提出了一种基于时态语法分析的 Lambek 演算，其在原以 Lambek 演算为核心的范畴语法基础上，引入了时态类型。由于时态类型区别于语言学中诸如动词、名词、副词等语

言结构类型，我们提出了并发类型演算（语言结构类型与时态类型并发处理），在文中给出了该演算系统的代数模型，并证明了相应的可靠性与完全性。我们将此演算系统称为 **LC** 演算系统，在文末给出基于 **LC** 系统的范畴语法 **LCCG**，并对其语义转换做了相应处理，该处理使之更易于计算机编程实现。由于在此演算系统中，只讨论了时态演算中最本质的三种时态对象，即现在时，过去时和将来时，因此我们称之为“基本时态句型演算”。

在后续研究中，我们将致力于研究该系统在切割消除、判定性、有穷模型性和有穷阅读性等方面的性质。并进一步扩充时间范畴，使其具有更强的解释能力与演算能力，得到更广泛的应用，例如建立与分析时态数据库的数据库操纵语言、创建时态语言分析器的中间件等方面。

本文是在鞠实儿教授的精心指导下完成，在此谨对其致以诚挚的感谢与敬意。

参考文献

[1] Lambek, J. (1958), The mathematics of sentence structure, Amer, Math, Monthly 65, 154-170.

[2] Lambek, J. (1961), On the calculus of syntactic types, Stucture of Languange and Its Mathematical Aspects, R. Jakobson, ed., AMS, Providence, RI.

[3] Montague, R. (1970), Universal grammar, Theoria 36, 373-398.

[4] Montague, R. (1973), The proper treatment of quantification in ordinary English, Approaches to Natural Language, J. Hintikka, J. Moravcsik and P. Suppes, eds, Reidel, dordrecht.

[5] Curry, H.B. (1961), Some logic aspects of grammatical structure, Stucture of Language and Its Mathematical Aspects, R. Jakobson, ed., AMS, Providence, RI.

[6] Howard, William A. (1969). The formulae-as-types notion of construction. Manuscript, published in Seldin and Hindley (1980).

[7] W. Buszkowski, The equivalence of unidirectional Lambek categorical grammars and context-free grammars, Z. Math. Logik Grundlagen Math. 31 (1985) 369-384.

[8] M. Pentus, Lambek grammars are context-free, in: Proc. 8th Ann. IEEE Symp. On Logic, in Computer Science, Montreal, 1993.

[9] Gerhard Jäger, Anaphora and Type Logical Grammar, UiL OTS Working Paper, September 2001.

[10] W. Buszkowski, 1997, Mathematical Linguistics and Proof Theory' In J. van Benthem & A. ter Meulen, eds., Handbook of Logic and Language, 638–738, Elsevier Science Publishers, Amsterdam.

[11] Ajdukiewicz, K. (1935), Die syntaktische Konnexität, Stud. Philos. 1, 1-27.

[12] Bar-Hillel, Y. (1953), A quasi-arithmetical notation for syntactic description, Language 29, 47-58.

[13] Buszkowski, W. (1986), Completeness resuLCs for Lambek syntactic calculus, Z. Math. Logik Grundlag. Math. 32, 13-28.

[14] W. Buszkowski, 1999,Algebraic structures in categorical grammar, Theoretical Computer Science 199(198) 5-24.

[15] M. Pentus, Lambek calculus is L-complete, Report LP 94-14, Institute for Logic, Language and Computation, University of Amsterdam, 1993.

[16] Chomsky, (1981), Lectures on government and Binding. Foris, Dordrecht.

Lambek Calculus of Basic Tense Syntax in Nature Language

Dongning Liu, Wenlong Nie

（Institute of Logic and Cognition，Sun Yat-sen University, Guangzhou，510275）

Abstract：This paper gives a kind of Lambek Calculus which based tense syntax. The **LCG**(Grammar Category of Lambek Calculus) is the core of this method, and it appends the temporal types to the type categories. Because the temporal types are different from the basic types in linguistics, such as verb, noun, adverb types etc., so we puts forward a kind of concurrence types calculus which deals with the temporal and basic types respectively, and we give the algebra model of the calculation system and prove the Soundness and Completeness Theory. This calculation system is called **LC**, it puts forward the category grammar **LCCG** and its semantic conversion in the end of the paper which can be programmed expediently. Because it only deals with the essential tense syntax such as present, past, and future tense, so the calculation is called "Basic Temporal Syntax Calculation" in the paper.

Key Words：Tense Syntax, Lambek Calculus, Concurrence Type, Category Grammar