自然语言处理中的逻辑文法

陈震杰　闵珍晖　夏振华

（南京航空学院计算机系）

摘要

Logic grammars is a new subject developed abroad recently. The base is modern linguistics, the tool is logic programming language (i.e Prolog) and the goal is to solve the problems of linguistics in natural language processing. It has the characteristics of being simple, expressive and easy to implement, whick make it find extensive application in the fields of natural language processing, and so on. This paper introduces logic grammars such as DCG, XG, MSG, RLG and the like.

逻辑文法是指用谓词逻辑来表达的文法[1]。它属于计算语言学的范畴，是逻辑程序设计和现代语言学相结合的产物。在人工智能的自然语言处理等领域里，谓词逻辑通常用来描述知识和逻辑推理。70年代，逻辑用于程序设计的思想以Prolog语言的形式投入应用以来，谓词逻辑不再仅仅用于描述这些问题，还作为逻辑程序设计的工具去描述解决问题的过程。PROLOG语言使得逻辑和程序设计这样两个完全不同的概念协调统一为一个概念——逻辑程序设计。如何用PROLOG语言来有效地解决自然语言处理中的一些问题的研究，促进了逻辑文法科学的发展。

近几年来，国外许多学者在逻辑文法的研究上做了大量的工作，取得了一定的成绩。先后取得的成果有定子句文法DCG（Definite Clause Grammars）、外位文法 XG（eXtraposition Grammars）、修饰语结构文法MSG(Modifier Structure Grammars) 和限制逻辑文法 RLG（Restricting Logic Grammars）等。

一　定子句文法DCG

1956年，Chomsky在他的短语结构文法PSG(Phrase Structure Grammar)中，将2型文法，即上下文无关文法CFG(Context Free Grammar)，定义为：它的每条规则都具有形式

$$A \Longrightarrow B$$

其中，A为非终结符，B为终结符和非终结符组成的串[2]。

CFG文法很适于描述自然语言，目前虽没人能证明自然语言是CFG，却有人声称"目前还没有理由认为自然语言不是CFG[3]"。对大多数英语句子，可用CFG描述为：

$$\text{sentence} \Longrightarrow \text{noun\_phrase}, \text{verbp-hrase} \qquad (2.1)$$

1980年Warren和Pereira 通过扩展CFG提出了DCG。所谓DCG就是仅使用CFG规则的逻辑文法[4]。它的主要思想是：文法的符号可以是广义的逻辑项，而不仅仅是原子符号。DCG完成了形式文法（一种语言的描述文法）到逻辑文法（一种语言的推理文法）的转变过程。同样的2.1规则，CFG表示一个句子由名词短语和动词短语两部份组成。而DCG则表示如果存在一个名词短语和一个动词短语，则存在一个句子的推理过程。二者在形式上有许多相同之处，本质上是有很大区别的。

一个具有N个参数的DCG逻辑项可以通过增加两个串操作指针而直接转换成一个只有N+2个参数的Horn子句或Prolog谓词。Horn子句是至多含有一个正文字的短句[5]。若从逻辑程序设计的观点来解释，就是"它的规则的左部最多只能有一个谓词[4]"。上面的规则可用Horn子句描述为：

```
sentence(S0, S):- noun_phrase (S0, S1),
verb_phrase(S1,S)                    (2.2)
```

这里S0、S1、S为字符串指针，这个Horn子句可解释为：如果S0到S1之间是一个名词短语，而且S1到S之间是一个动词短语，则S0到S之间是一个句子。

由于DCG文法中的符号是逻辑项，这就使DCG规则中的非终结符可以携带上下文，转换，结构等信息。而且DCG规则的右部不仅可以是终结符，非终结符，还可以是谓词或测试。所以虽然DCG仅使用了CFG文法，它的描述能力事实上已相当于Chomsky所定义的0型文法[4,6]。Warren和Pereira的工作被认为是对计算语言学的杰出贡献之一[7]。

二 外位文法XG

虽然DCG已有较强的描述能力，但对于自然语言中的一些语法现象（如关系从句中，关系代词的左移位现象）有时仅用DCG去描述是很不方便的。基于这个原因，继1980年的DCG后，1981年Pereira在变形文法MG (Metamorphosis Grammars)的基础上提出了XG。

MG是1978年由A. Colmerauer提出的。在MG中，每条规则都具有形式：

$$Sa \Longrightarrow B$$

其中S是非终结符，a是终结符和非终结符的字符串，B是终结符，非终结符和过程调用的字符串。

MG可以用较灵活的规则来描述象定语从句中，关系代词在句子中的实际位置和应在位置相分离那样的语法现象。借助于MG的这一思想，Pereira在其定义的XG规则中，允许在DCG的规则中，出现形式为

$$S_1 \cdots S_2 \text{ etc } S_{K-1} \Longrightarrow r$$

的规则。这里"…"称之为"间隔(gap)"，r和$S_i(i=1,2,\cdots,k-1)$是终结符和非终结符组成的字符串。通俗点解释该规则就是对任意一个符号序列

$$S_1\ X_1\ S_2\ X_2 \text{ etc } S_{K-1}X_{K-1}S_K$$

可以重写(rewrite)成$rX_1X_2\cdots X_{K-1}$即$S_i(i=1,\cdots,k)$被重写成r，而$X_i(i=1,\cdots,k-1)$则依次移到r的右边。

设有如图1所示的一组XG规则：

```
sentence⇒noun_phrase,verb_phrase.        (1)
noun_phrase⇒determiner,noun,relative.    (2)

noun_phrase⇒trace.                       (3)
verb_phrase⇒verb,noun_phrase.            (4)
verb_phrase⇒verb.                        (5)
trace⇒[].                                (6)
relative⇒[].                             (7)
relative⇒rel_marker,sentence.            (8)
rel_marker…trace⇒rel-pronoun.            (9)
```

图1 关系从句的XG

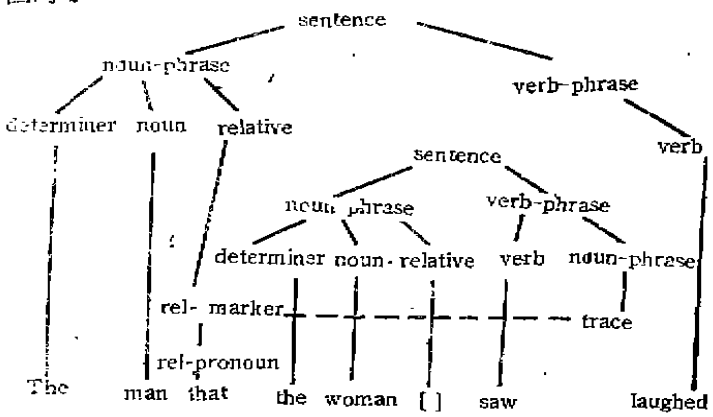

图2 句子"The man that the woman saw laughed."的推导图

则可以形成如图2所示的句子"The man that the woman saw laughed."的推导图。

图1中的规则(9)描述了定语从句中，关系代词的左位移现象。rel_marker指出了关系代词在句子中的位置。trace则指出了关系

代词应在从句中的位置。前者是位移的标记(marker)，后者是留下的痕迹(trace)。

由于在XG中，规则Sa⇒B中的a为终结符和非终结符的字串，就使得XG表达的位移是跟随非终结符(或终结符)的移位。若用MG去描述定语从句的话，须在图1中删去规则(9)，再增加下列两条规则：

$$\begin{cases}\text{verb-phrase}\Rightarrow\text{moved-dobj, verb.}\\ \text{rel-marker, noun-phrase, moved-dobj}\Rightarrow\\ \text{rel-pronoun, noun-phrase.}\end{cases}$$

比较增删的两组规则，可以知道XG完成左移时比MG灵巧的多。它实际上是跳过(skip)了一些非终结符(或终结符)，而不是一个一个地移过。

图1的XG规则可以接受下列英语句子：

$$\begin{cases}\text{The mouse squeaks.}\\ \text{The cat likes fish.}\\ \text{The cat chased the mouse.}\end{cases}$$

再进一步，它还能接受下面的错误的句子：

* The mouse that the cat that 〔trace〕 chased 〔trace〕 likes fish squeaks.

问题是在使用规则(1)，(3)和(9)时，没能保证先出现的rel-marker必须和先出现的trace匹配。为此，Pereira把图1中的规则(8)修改为

relative⇒open, rel-marker, sentence, close. (8′)

同时，增加规则

open…close⇒[]. (10)

这里open和close好象一对括号，限制了rel-marker和trace的匹配。Pereira称之为"括号限制(bracketing constraint)"。

Pareira还给出了一个用Prolog语言实现的XG的算法。算法充分利用了Prolog搜索，匹配和推理的长处，设计得十分精巧优雅，耐人寻味。

在XG中，为处理gap，使用了一个称为外位表(extraposition list)的数据结构：

x(Context, Type, Symbol, Xlist)

这里，Context为gap或nogap

Type为终结符terminal或nonterminal

Symbol随使用的规则不同取不同的值

Xlist为下层的外位表

这样对于不含"…"的XG规则，只须增加两个用操作指针S0，S和两个传递外位表的参数X0，X。如对规则(1)有

sentence(S0, S, X0, X):-noun-phrase(S0, S1, X0, X1), verb-phrase(S1, S, X1, X).

而含有"…"的规则，即

rel-marker…trace⇒rel-pronoun.

写成

rel-marker(S0, S, X0, x(gap, nonterminal, trace, X)):-rel-pronoun(S0, S, X0, X).

和

trace(S0, S0, X0, X):-virtual(trace, X0, X).

对于规则

open…close⇒[].

则写成

open(S0, S, X, x(gap, nonterminal, close, X)).

和

close(S0, S0, X0, X):-virtual(close, X0, X).

辅助谓词virtual定义为：

virtual(NT, x(-, nonterminal, NT, X), X).

在1981年，V. Dahl指出XG中的gap存在一些局限性：出现在规则左部的gap必须是顺序的，而且多个gap间须服从嵌套限制，即或者互相独立或者一个必须完全包含另一个。为此Dahl设计了一个"间隔文法GG(Gapping Grammars)"。

GG除了在规则中可以出现gap和将XG中的"…"代之以"gap(G)"外，在形式上和MG相同。这里的G代表了一个形成gap分量的实际的字串。在同一规则中允许出现多个gap，而且它们可以在规则的右部保留，复制和删除。如规则

A, gap(X), B, gap(Y), C⇒gap(Y), C,

B,gap(X)

将此规则分别作用于字串 AEFBDC 和 AB-DEFC

若令X=EF, Y=D 则有 AEFBDC⇒DCBEF

若令X=[], Y=DEF 则有 ABDEFC⇒DEFCB

或许是由于XG文法简洁，易于实现的原因吧，在逻辑文法的研究中，XG比GG更多地为后来的研究者们所引用。

三 修饰语结构文法MSG

1982年，McCord用Prolog语言设计了一个把对数据库的自然语言查询转换成逻辑表达式的语言解释器[8]。该解释器可以将句子：

who taught CS607 Spring 1980?

表达成

wh(indef,_1,true&_2=CS670 & (cls1(_3,_2,_1,_4)&_5=session(Spring, 1980)&session(_4,_5))&past(_4))

这里cls1定义为关系：

cls1(Class,Course,Instructor,C)

其中C为文本(context)参数。McCord的解释器首先将句子转换成一棵句法树，树的每个结点都带有语义修饰成份(modifier)，然后，解释器再根据修饰成份将句法树转换成句子的逻辑表达式。

83年，Dahl和McCord受扩充转换网ATN (Augmented Transition Network)处理连接词思想的启发，结合XG，在McCord的解释器的基础上，实现了一个能处理连接词，关系从句的英语解释器，并提出了MSG。该解释器可以将句子

A man and a woman saw a train.

解释为

exist(Y,train(Y),exist(X,man(X),saw(X,Y))&exist(X,woman(X),saw(X,Y))).

在MSG中，其规则只有两种形式：

1. A:Sem⇒B.

2. A:l_true⇒B.

这里A⇒B是XG规则。Sem是一个语义项，l_true为左连接操作。Sem与l_true和句法树的生成无关。它们的作用是作为句法树到句子的逻辑表达式转换的依据。

在MSG中，每条规则A:Sem⇒B都转换成谓词

rule(NT,Ext,Sem,Bl).

放入事实库中。这里，NT是A里的起始非终结符。Ext是XG中定义的外位表。Bl是B的表的形式。例如，MSG的规则

A…B:l_p⇒[c], e

被转换成

rule(A,x(gap,b,nil),l_p,[[c],e]).

MSG处理连接词的思想和ATN基本相同。如对下列形式的句子：

A X and Y B.

如，

The man saw and cleaned it. (4.1)
A X Y B

当句法分析程序处理到and时，首先保存当前文法的状态历史，然后开始一个回溯、处理、将当前文法状态和前面保存的文法状态比较的循环过程。当回溯到X，且处理完Y时，由于此时的状态和前面保存的文法状态相同，将X和Y合一，继续往前处理B。这样就将

A X and Y B.

转换成了

A X B and A Y B.

Dahl和McCord声称MSG处理连接词的方法在下述两个方面优于ATN。

1)由于MSG的回溯是以文法非终结符为单位进行的，故MSG的效率比ATN高。

2)由于MSG使用Prolog设计系统，谓词间递归调用，故可以处理嵌套的并列结构。

MSG处理连接词时，使用了两个栈：父结点栈(parent stack)和匹配栈(merge stack)。parent栈用来保存句法分析的历史。

merge栈用来保存回溯点的现场。若不考虑MSG规则中的 Sem 部份，参考图1的XG 规则，MSG处理句子(4.1)时，parent栈和merge栈的动作过程如图3。

状态	单词	parent栈	merge栈	说明
[S]	[The man]	nil	nil	,
[NP, Vp]	[The man]	p([S], nil)	nil	, S⇒Np, Vp
[Vp]	[saw]	p([S], nil)	nil	, Np⇒[The man]
[verb,Np]	[saw]	p([Vp], p([S], nil))	nil	, Vp⇒verb, Np
[Np]	[and]	p(Vp], p[S], nil))	nil	, verb⇒[saw]
[Vp]	[cleaned]	nil	m([Np], p([Vp], p([S], nil)), nil)	保存现场
[verb, Np]	[cleaned]	p(Vp], nil)	m[Np], p([Vp], p([S], nil)), nil)	Vp⇒verb, Np
[Np]	[it]	p([Vp], nil)	m([Np], p([Vp], p([S], nil)), nil)	状态合一
[Np]	[it]	p([Vp], p([S], nil))	nil	, 恢复parent栈
[]	[]	p([S], nil))	nil	, Np⇒it
[]	[]	nil	nil	, 正常结束

图3 MSG句法处理过程示意图

四 约束逻辑文法RLG

由于人们在设计一个句法分析器时，通常追求其通用性和有效性，因而如果发现一个很好的句法分析器有时不能识别一个语法上错误的句子，就是不足为奇的了。通常随着系统为覆盖更多的语法现象而进行扩充时，那些可能产生错误的规则是会被修订的。若能在开始时，便对文法规则作较多的限制，以后的修订就会少一些。

基于上述想法，Stabler运用Chomsky的支配-约束理论(Government_Binding theory)，在DCG和XG的基础上，推出了RLG[8]。Stabler认为RLG和XG的区别有下面三点：

1. 提出了一种新的规则—切换规则(switch rule)

2. 对XG的左移位做了一些扩展和限制，以便适应更多的语言现象。

3. 提出了处理有限制的右移位的方法。

切换规则提出的原因是很难用CFG去精巧地解释当在一个句子中有多个助动词时，什么样的排列次序是正确的，或是不正确的。如用CFG就很难解释下面的四个句子是正确，还是不正确。

I have been successful.

Have I been successful?

*I been have successful.

*Been I have successful?

Pereira为解决这个问题，曾使用下面的XG规则：

s⇒fronted_verb,s.

fronted_verb…aux_verb(Features) ⇒ aux_verb(Features).

但是该规则将认为下面这个句子也是正确的。

*Has he has been saying that he [t] been succeeding?

Stabler在RLG中使用的切换规则为：

s⇒switch(aux_verb,np),vp.

这里switch的功能是：若在句首有一个aux_verb，则暂不处理而先去归约np，在np的归约完成后，再将aux_verb放在剩余部份的首部去归约vp。

切换规则可以很容易地用Prolog子句描述为：

```
s([First|L0],L,X0,X):-aux_verb(First),np(L0,L1,X0,X1),vp([First|L1],L,X1,X).
```

RLG的左移规则的基础是Chomsky在1981年提出的约束(binding)移动(bounding)理论。Chomsky的理论可以形式化地叙述为下面的二点限制：

1. 一个移动分量必须c_command它的痕迹。

这里，一个结点A c_command B结点当且仅当若A不支配B则第一个支

配A的分支结点一定支配B结点。

2．没有任何规则能产生下述形式的结构，使得移动分量X和Y或Z有关系。

…Y…〔A…〔B…X…〕…〕…Z…

这里，A和B是“移动结点”。在英语中，假定是s和np。

第一点将保证句法分析器不会接收下面的句子：

*The computer 〔which, you wrote the program〕 uses 〔np_trace〕,

*I saw the man 〔who, you knew him〕 and I told 〔np_trace〕,

因为在这里支配which和who的第一个分支结点不支配np_trace。

第二点是用于移动限制的。它将排斥下面的句子：

*who, 〔s did 〔np the man with〔np_trace〕,〕 like〕?

*〔About what〕, 〔s did they burn 〔np my book 〔pp_trace〕,〕〕?

为使句法分析器能满足上面的要求，Stabler首先将XG中的外位表的X栈结构改为X表结构。同时亦将XG规则做了一些改动。如将

(XG规则)

relative⇒rel_marker,s.

rel_marker…np_trace⇒rel_pronoun.

改为(RLG规则)

relative<<<np_trace⇒rel_pronoun,s.

这里“…”到“<<<”的改变仅是为了表示RLG的左移操作。删去了XG中的rel_marker。保留痕迹的任务放在了s的归约里。为此，上述RLG规则用Prolog实现时须写为：

```
relative (L0,L,X0,X):-rel_pronoun
(L0,L1),
s(L1,L,〔trace(Index)〔X0〕,X),
tracegone(trace(Index),X).
```

这样在s的推导过程中，如遇到trace，则须在外位表中消去trace(Index)。这里Index是trace的序号。它保证了rel_pronoun和trace的c_command关系。tracegone的作用就是测试X中是否的确已消去了trace(Index)。

RLG中右移位规则的提出是为了句法分析程序能接受下列的句子：

〔The man 〔t〕,〕 arrived 〔who I told you about〕,

〔what book 〔t〕,〕 arrived 〔about the arms race〕,?

右移位的处理思想是：在np的处理时，将一个携带np位置t_i的可选择的右移(optional rightward)变量Tree通过移位表X往后(右)传。在vp的推导完成时执行一个附属物(adjunct)检查操作。即检查是否有pp短语或关系从句。若有，则将pp短语或关系从句的语法树赋给Tree。若没有，则Tree赋[]值。

这里的难点是如何实现adjunct。在vp的推导完成后遇到pp或以wh开头的句子的剩余部份时，adjunct须作出判断，它们是句子的右移成份，还是句子的状语部分。

逻辑文法是语言学和PROLOG逻辑程序设计相结合的产物。逻辑文法的发展演变都是以语言学为基础，围绕如何充分利用PROLOG语言的特点有效地解决自然语言处理中的问题进行的。一种新的文法规则、一组相应的PROLOG算法，利用语言学的知识、解决了一个特定的语言问题就导致一种新的逻辑文法的诞生。PROLOG逻辑程序设计在这里起了很大的作用。在DCG，XG和GG中还强调文法的表示形式，在MSG和RLG中则更多的是PROLOG的逻辑程序设计了。逻辑文法的特点是简洁明了、实用性强。相信逻辑文法的发展将对自然语言处理的研究产生积极的影响。逻辑文法也将广泛地应用于人工智能的各个领域。

参考文献

〔1〕 Dahl V. and McCord M. C, Treating Coordination in Logic Grammars, American Journal of Computational Linguistics Vol. 9, No.2, Apr. -Jun. 1983

〔2〕 东火臣等，程序设计语言编译原理，国防工业出版社 1980

〔3〕 Gerald Gazdar, Phrase Structure Grammars and Natural Language, Proceedings of the 8th. International Jo.

原型法：设计与开发应用系统的方法论

A. Milton Jenkins

摘 要

本文首先对原型方法论进行详细定义，然后阐述如何建立一个业务原型的有关问题。规范模型说明这一过程应怎样起作用才好，而描述模型说明与规范模型不同的典型行为及其有关风险。接着讨论业务原型的可能应用。最后，就何时使用原型方法，何时不使用原型方法，提出一些经验性建议，并讨论迄今所获得的典型成本与效益的经验。本文是对五个组织及120个以上原型法应用项目进行观察研究后写成的。

一 导 论

本文的目的是对原型法(Prototyping Me-thodology, PM)进行详细的一般说明。这包括原型法的定义，如何开发业务原型的说明，业务原型的使用，应该何时使用原型法的建议以及对使用原型法所期望得到的结果等。

本文把原型法当作一种方法论，这种方法论就是描述设计与开发计算机化信息系统的系统化过程中的各种原则的应用和各种活动的有规则安排。原型法说明构造原型的过程，这个过程实际是把用户对信息系统的各种想法(输入)转变成一个功能性的产品——一个业务原型(输出)。原型法支持诸如Weth-erbe的"启发式开发"和Edelman的"渐进式开发"之类策略。此外，原型法是Edelman称为的"加速式开发方法"。

1.1 背景/方向 几千年来，在设计领域中原型法已是很普遍了，但它在计算机信息系统的设计与开发领域中却是一个新的概念和方法。理由很简单，因为对于原型应用系统所要求的软件工具直到几年以前才刚刚出现。合适的开发工具，象Focus, RAMI-S II, X/L, EXPRESS, NOMAD等，高级建模工具，象IFPS, MAPS, EIS等，以及专用的工具为CADIS SYSTEM4, BDU, LEF, ACT/1等直到最近问世，而且将会出现更好的工具。

通过与实际应用者讨论，发现他们运用与

-int Conf. on AI 1983

[4] Dahl V., Hiding Complexity fron th-e Casual writer of Parsers, Natural Language Understanding and Logic Programming, Elserier Science Publishers B. V. 1985

[5] 刘风岐等，逻辑程序设计原理和方法，国防科技大学出版社 1987

[6] Pereira F., Extraposition Grammars, American Journal of Computational Lininguistics Vol. 7, No. 4, Oct.-Dec. 1981

[7] Timothy W. and Martha S., Parsing with Logic Variable, Proceedings of Conference on Applied Natural Langu-age Processing 1983

[8] McCord M. C., Using Slots and Mo-difiers in Logic Grammars for Natu-ral Languages, Artificial Intelligen-ce 18(3)(1982)

[9] Stabler E. P., Restricting Logic Gra-mmars with Govsrnment-Binding Theory, Computational Linguistics Vo-l. 13, No. 1-2, Jan.-Juh. 1987