**4.4 RelEx2Logic：把语言依赖关系转换成逻辑关系**

现在解释我们的自然语言理解管道中最后的纯语言学阶段，这称为RelEx2Logic。RelEx2Logic是我们本论题所做工作的一个关键组成部分，用于将RelEx生成的句法-语义语言分析结果转换成PLN和其它OpenCog认知组件可以使用的逻辑表示，从而使得我们的自然语言理解管道可以与逻辑推理系统（例如PLN）、控制系统（例如OpenPsi）和OpenCog的其它组件进行有效交互。

**4.4.1 RelEx2Logic**

自然语言允许内在含义相同的内容存在许多不同的表示。要想用OpenCog AtomSpace表示格式来表达自然语言语句的含义，必须面对这个事实：可以有许多语句不同但是语义等同的方法来表示一个简单的句子。RelEx2Logic系统的目标是以尽可能简单的方式将RelEx输出映射到PLN可以理解和推理的AtomSpace表示，生成可以获取相关表达所包含的大量含义的精确表示。RelEx2Logic不会试图解析RelEx输出中所有的语义歧义，而是努力解析为了生成精确的原子（Atom）结构（这些原子结构较易进行处理而获得有用的推断）所必须解析的那些歧义。

从操作上讲，RelEx2Logic的核心理念是通过应用一系列简单的重写规则以及较复杂的后处理规则（处理复杂句子和问题需要这些后处理规则，例如进行概念实例区分）把RelEx关系映射到语义解释（例如OpenCog AtomSpace表示）。

每个核心重写规则的输入是句法分析图中满足特定约束的子图，输出是一个原子（Atom）超图。在实践中，需要的规则通常采用成对标记(G,A)的形式，其中：

* G是一个图，其节点是单词或变量，其边是RelEx关系类型。
* A是一个超图，其节点是单词、变量或特殊的语言学节点（根据一个小型词汇表绘制），其超图边是OpenCog原子（Atom）类型（例如InheritanceLink和EvaluationLink）。
* G和A中的变量列表必须相同。

我们把符合这种说明的规则称为“简单映射规则”。

对于处理人类语言真正需要的简单映射规则来讲，满足约束条件“G中的每个边映射到A中的单个超图边”。从数学上讲，这个约束条件的后半部分意味着每个重写规则各自都是一个图同态[?]，这进一步意味着一起应用的一系列重写规则也是一个图同态。

这种规则的一个简单例子是这样一个(G,A)，其中

G = {S⇤(v1, v2), O⇤(v1, v3)}

A = (EvaluationLink v1 v2 v3)

这把主语为v2，宾语为v3的动词v1映射到一个v1为谓语，(v2, v3)为参数列表的OpenCog EvaluationLink。例如，S⇤指任何S的链路分析器子类型。当然，大部分规则都比这个例子复杂。

在具体实现中，由于自然语言的复杂性，仍然有一些这种形式的RelEx2Logic规则无法处理的语言学信息。所以，为了获取句子正确的逻辑表示而不影响其含义，我们向RelEx2Logic框架添加一个后处理步骤，这个步骤主要包含下列部分：

* 处理OpenCog Atomspace中的实例。根据输入的自然语言的含义，将通用概念或谓语与特定概念或谓语进行区分。
* 估计模糊谓语或概念的概率。例如，如果句子是“Maybe the cat chased a snake.”，那么我们需要给最后的逻辑表示指定相较于句子“the cat chased a snake.”较低的概率。
* 正确处理量词。众所周知，自然语言中的量词十分复杂。不可能创建一个通用规则而以通用方式获取所有量词的正确抽象表示。而且，我们想要使得我们的规则更加通用，所以我们不想为每个量词使用单独的规则，这需要以简化的方式编写量词规则，而不是极力利用后处理方法允许的更大灵活性。

下面将解释后处理的更多细节。

将我们当前的简单映射规则应用于上述的例句“The cat chased a snake.”，会生成以下基本输出：

EvaluationLink

PredicateNode chase@3453432

ListLink

ConceptNode cat@1243546464

ConceptNode snake@564636322

这些“基本输出”仅仅是这个句子解释后Atomspace中所创建的完整原子（Atom）集合的一小部分。一个更完整的列表是

XXX paste in full output for the sample sentence "The cat chased a snake ."

本节最后会给出更多映射例子以及所涉及原子（Atom）类型的解释。

如果链路分析器给出多个分析选项，RelEx2Logic规则将为每个分析选项提供一个逻辑解释。目前已经实现基于语料库进行统计启发来排列多个分析的可信性，当然这并不完美。一些情况中，会向OpenCog提供多个逻辑输出选项，并且必须基于更高层次的上下文推断在这些选项之间进行选择，这是当前研究的一个难题和主题。

**4.4.2 RelEx2Logic规则**

为了将RelEx关系转换成抽象的OpenCog AtomSpace逻辑表示，我们引入一系列RelEx2Logic规则。RelEx2Logic规则（以前称为Link2Atom和Syn2Sem规则[?]）是一系列人工编码的规则，用于把RelEx的输出映射到OpenCog原子（Atom）表示的形式，这种表示更适合于由PLN和其它工具进行自动化推断。本节总结性地阐述规则集合并给出一些规则例子。可以在opencog/nlp/scm/relex-to-logic.scm在线找到所有规则。

RelEx2Logic规则有多个目标：

* 作为运行的OpenCog自然语言处理系统的一部分，其潜在用途包括文本分析和自然语言对话。
* 作为概念研究工具，帮助我们理解句法-语义输出（例如RelEx生成的句法-语义输出）映射到更抽象的逻辑关系（例如PLN采用的逻辑关系）的问题。
* 作为工具来帮助生成成对（链路分析和抽象的PLN形式的原子）的语料库，然后可以自动分析此语料库来产生语言理解和生成规则（更多信息请参见Syn2Sem）。

示例规则（Tense规则）

为了方便阐述剩下的内容，我们从一个非常简单的RelEx2Logic规则例子开始。当然，大部分规则比这个例子复杂。此处，为了方便人理解，采用RelEx SFF格式（RelEx许多输出格式中的一种）来说明规则输入。

为了用便于阅读的方式说明RelEx2Logic规则，我们在此使用类似以下的格式：

规则输入：

tense(W, Tense)

pos(W, verb)

规则输出：

(tense-rule W (get\_instance\_name W word\_index sentence\_index) Tense)

在OpenCog中运行此规则，规则的输出将启动一个帮助器函数（在Scheme中编码）。下面会列出这个帮助器函数。然后帮助器函数将最终生成最后的OpenCog Atomspace表示。

如同上一子节中所述，在我们真实的软件实现中，我们使用诸如(ImplicationLink PG PA)这样的格式来表示RelEx2Logic规则，从而使得RelEx2Logic与OpenCog系统的其余组件保持一致。例如，这种格式意味着规则可以由OpenCog模式匹配器来执行（OpenCog模式匹配器用来在OpenCog Atomspace中搜索原子的特定模式、排列或“模板”）。所以上述简单的RelEx2Logic规则在当前系统中将被如下表示：

ImplicationLink

EvaluationLink

PredicateNode "tense"

ListLink

WordNode W

ConceptNode Tense

ExecutionOutputLink

GroundedSchemNode "r2l/tense-rule.scm"

ListLink

WordNode W

ConceptNode Tense

NumberNode word\_index

NumberNode sentence\_index

对应于上述RelEx2Logic规则的提供后处理的相应帮助器函数为：

帮助器函数：

(define (tense-rule verb instance tense)

(define new\_predicate (PredicateNode instance))

(define verb\_node (PredicateNode verb))

(define tense\_node (ConceptNode tense))

(list

(InheritanceLink new\_predicate verb\_node)

(InheritanceLink new\_predicate tense\_node)

)

)

例子：

tense(chase, Past)

pos(chase, verb)

==>

(tense-rule "chase" "chase@3453432" "Past")

==>

(InheritanceLink "chase@3453432" chase)

(InheritanceLink "chase@3453432" Past)

注意：这个例子使用了额外的帮助器函数*get instance name。*

下面解释这个帮助器函数。本质上，这个函数只是为概念的实例选择一个在给出的AtomSpace中唯一的名字（例如，它可能为*chair*的实例选择名字chair77）。如果单词W在句子中是第K次出现，那么单词序号为K。句子序号是该句子区别于其它句子的一个标识符。

**4.4.3 后处理**

RelEx2Logic最初的设计意图是如同上述例子那样通过直接的逻辑含义进行所有需要的处理。但是随着工作的进展，我们认识到这样做在许多情况中会非常麻烦，同时我们发现引入一些更精巧的后处理方法是最直接的解决办法。此处我们阐述这些后处理规则的基本类型。

现阶段，我们已经实现的后处理规则包括下列类别：

* 量词：为带有量词的单词引入变量并通过使用相应的链路限定变量的范围，同时给链路指定一些真值。
* 模糊情态动词：为模糊情态动词发生过修改的谓语指定一些置信下限。例如“Maybe dogs like fish.”，这表示“dogs like fish”的置信度不是很高，所以我们把“he is a genius.”的置信度从默认的0.99调整为0.5。
* 冗余实例清理：为可以从句法分析知道的通用概念清理实例号。例如“Dogs like meat.”，没有为dogs和meat指定任何明确的冠词、修饰语或代词，并且也没有意指任何其它概念，所以我们认为dogs和meat指通用概念dog和meat，而不是某些特定概念（特定概念需要指定实例号）。

下面更详细地阐述我们怎样为上面列出的每种类别实现后处理。

量词：在前面的阶段，我们使用量词记号来标记需要进行后处理的量词。例如“All dogs like meat.”，在应用量词RelEx2Logic规则期间创建“allmarker”并生成下列结果。

EvaluationLink

PredicateNode "allmarker"

ListLink

ConceptNode "dog@13456"

本例中，在后处理阶段，我们为量词“all”引入“ForAllLink”（它代表OpenCog Atomspace中的通用量化）和变量节点“$X”来限定范围包含所有含有dog@13456的链路。然后，每个dog@13456都将被替换为VariableNode “$X”。“allmarker”将在后处理后被删除。所以，在后处理结束后，将创建：

ForAllLink

VariableNode "$X"

ImplicationLink

InheritanceLink "$X" noun\_instance

AndLink

\*\* links involving "dog@13456" \*\*

模糊情态动词：模糊情态动词后处理类似于量词后处理，我们在应用相关的RelEx2Logic规则期间使用相应的记号来标记模糊情态动词。例如，对于句子“Maybe dogs like fish.”，在应用Maybe RelEx2Logic规则后生成下列表示：

EvaluationLink

PredicateNode "maybemarker"

ListLink

PredicateNode "like@9768"

然后，用于后处理的maybemarker帮助器函数将查找所有包含“like@9768”的根链路，并将置信度从默认的0.99调整为0.5。

冗余实例清理：因为不想在句子转换成抽象语义表示期间丢失任何来自句法层面的信息，我们在后处理阶段设立“实例清理”步骤。在当前方法中，我们为句子中每个非组合单词创建一个单独的实例节点。但是，在很多情况中，这会生成许多不必要的实例节点。例如，在句子“Dogs like yummy meat.”中，无需创建诸如yummy@1765和dog@2593的实例节点，因为这个句子没有指定特定的“dog”和特殊的“yummy”。所以只创建类似以下的链路就已足够：

InheritanceLink

ConceptNode "meat@3976"

ConceptNode "yummy"

EvaluationLink

PredicateNode "like"

ListLink

ConceptNode "dog"

ConceptNode "meat@3976"

但是，一个诸如“meat@3976”的实例节点更加有用。因为在这个句子中，它仅陈述“Dogs like ’YUMMY’ meat”，而并非一般的“meat”。

要想知道哪些单词真正需要实例节点，必须在一定程度上理解句子的结构。当前，我们使用一些常识性的规则开始这个过程。确切地讲，如果一个单词没有被修改（名词被形容词修改或者动词被副词修改），或者没有被定冠词限定，或者使用代词，那么此单词的实例节点为“冗余”。清理实例的后处理进行以下工作：

* 在Atomspace中（或者仅在RelEx2Logic规则的本地输出中）检查是否存在任何此名字的原子为“冗余”。
* 如果找到冗余原子，那么将其删除并将它的链路移至其父概念。

**4.5 例子和分析**

本节中给出更多例子来解释我们的语言理解的理解管道，包含前面阐述的所有部分：Link Parser、RelEx和RelEx2Logic。在后面的章节中，这些特定例子用来阐述PLN对语言理解管道输出的推断。

XXX添加一些解释和这些句子的RelEx2Logic输出（以及相关的原子类型解释？）XXX

例子1：Socrates is a man.

依赖关系：

\_obj(be, man)

\_subj(be, Socrates)

属性：

tense(be, present)

subscript-TAG(be, .v)

pos(be, verb)

pos(., punctuation)

subscript-TAG(man, .n)

pos(man, noun)

noun\_number(man, singular)

definite-FLAG(Socrates, T)

pos(Socrates, noun)

noun\_number(Socrates, singular)

pos(a, det)

例子2：Men breathe air.



依赖关系：

\_obj(breathe, air)

\_subj(breathe, man)

属性：

tense(breathe, present)

subscript-TAG(breathe, .v)

pos(breathe, verb)

pos(., punctuation)

subscript-TAG(air, .n-u)

pos(air, noun)

noun\_number(air, uncountable)

subscript-TAG(man, .p)

pos(man, noun)

noun\_number(man, plural)

例子3：Socrates breathes air.



依赖关系：

\_obj(breathe, air)

\_subj(breathe, Socrates)

属性：

tense(breathe, present)

subscript-TAG(breathe, .v)

pos(breathe, verb)

pos(., punctuation)

subscript-TAG(air, .n-u)

pos(air, noun)

noun\_number(air, uncountable)

definite-FLAG(Socrates, T)

pos(Socrates, noun)

noun\_number(Socrates, singular)

例子4：I think Socrates is a man.



依赖关系：

\_obj(be, man)

\_subj(be, Socrates)

\_rep(think, be)

\_subj(think, I)

属性：

definite-FLAG(Socrates, T)

pos(Socrates, noun)

noun\_number(Socrates, singular)

pos(., punctuation)

subscript-TAG(man, .n)

pos(man, noun)

noun\_number(man, singular)

pos(a, det)

tense(be, present)

HYP(be, T)

subscript-TAG(be, .v)

pos(be, verb)

tense(think, present)

subscript-TAG(think, .v)

pos(think, verb)

pronoun-FLAG(I, T)

gender(I, person)

definite-FLAG(I, T)

subscript-TAG(I, .p)

pos(I, noun)

noun\_number(I, singular)

例子5：Bob thinks Socrates is a woman.



依赖关系：

\_obj(be, woman)

\_subj(be, Socrates)

\_rep(think, be)

\_subj(think, Bob)

属性：

definite-FLAG(Socrates, T)

pos(Socrates, noun)

noun\_number(Socrates, singular)

pos(., punctuation)

subscript-TAG(woman, .n)

pos(woman, noun)

noun\_number(woman, singular)

pos(a, det)

tense(be, present)

HYP(be, T)

subscript-TAG(be, .v)

pos(be, verb)

tense(think, present)

subscript-TAG(think, .v)

pos(think, verb)

gender(Bob, masculine)

definite-FLAG(Bob, T)

person-FLAG(Bob, T)

subscript-TAG(Bob, .m)

pos(Bob, noun)

noun\_number(Bob, singular)

**4.6 比较级：范例分析**

比较级提供了从表层形式映射到我们设计和实现的逻辑表示的有趣例子。

关于正确处理英语和其它语言中的比较级，理论上的语言学从来没有达成一致。一些理论家假定一种省略理论，建议从句子的表层结构得出比较级语法，忽略深层结构中存在的某些单词[?] [?]。另外一些理论家假定一种移动理论[?] [?]，这种理论更多地由传统的生成语法启示，假设比较级语法包含一个重新排列深层结构的表层结构。

链路语法框架从根本上回避了这种问题：不管省略理论还是移动理论，都由链路语法词典中的某些对称性表示，但是这些对称性不需要被链路分析器本身显式地识别或使用（虽然这些对称性可以指导人类或AI系统创建链路语法词典）。从目前的经验上讲，链路分析器对比较级的处理相当好，但是相关的词典条目有点混乱并且实际上并不完全对称。这说明两种可能：

1. 英语比较级的语法“复杂”而混乱，不适用任何可用的理论。并且/或者
2. 链路语法词典可以在比较级方面进行极大的改进

我们猜测事实是两者兼而有之。但是请注意，将链路语法作为用于理解复杂句子（包含比较级）的实际管道的一部分而部署时，我们不需要决定这个问题。

我们通过一个例子说明我们的框架如何处理比较级。RelEx2Logic的一个用于比较级的规则，其简短形式如下：

than(w1 , w2)

\_comparative(ad , w)

==>

TruthValueGreaterThanLink

InheritanceLink w1 ad

InheritanceLink w2 ad

图16给出了完整形式。

使用这个规则的一个简单例子是：

Pumpkin is cuter than the white dog.

==>

\_predadj(cute , Pumpkin)

than(Pumpkin , dog)

\_comparative(cute , Pumpkin)

\_amod(dog , white)

==>

AndLink

InheritanceLink dog\_11 white

InheritanceLink dog\_11 dog

TruthValueGreaterThanLink

InheritanceLink Pumpkin cute

InheritanceLink dog\_11 cute

另一方面，为了处理诸如“Amen is more intelligent than insane”这样的句子，我们使用一个不同的规则，这个规则的简短形式如下：

\_predadj(adj1 , W)

than(adj1 , adj2)

\_comparative(adj1 , W)

==>

TruthValueGreaterThanLink

InheritanceLink W adj1

InheritanceLink W adj2

结果输出如下：

\_predadj(intelligent , Amen)

than(intelligent , insane)

\_comparative(intelligent , Amen)

==>

TruthValueGreaterThanLink

InheritanceLink Amen intelligent

InheritanceLink Amen insane

**4.7 成果总结和未尽事宜**

本章中，我们总结了一个自然语言理解系统的操作。这个自然语言理解系统把英语句子映射到逻辑关系集合中，采用通用目标认知体系（OpenCog）中实现的概率性推断引擎（PLN）所使用的逻辑格式。这个理解系统正被应用在多个领域的原型应用中，包括视频游戏中的非玩家控制角色和室内环境中的类人机器人操作。

得益于许多不同人群的协作，这个理解系统得以不断发展。在这个过程中，我们贡献了理论和代码库的几乎各个方面。但是，我们最大的贡献如下：

* 改进链路语法词典，使其在语言学意义上更接近真实的依赖性语法（通过建立直接连接到短语首部动词的链路类型）。
* 改进RelEx，使其正确处理比较级和量词（还有许多其它RelEx改进）。
* 设计RelEx2Logic子系统来把RelEx输出映射到OpenCog原子中，并且实现RelEx2Logic规则解释器的核心。
* 设计和实现初始的RelEx2Logic转换规则。

总的来讲，我们创建的理解系统能够并且足够用作创建定制的实际应用的基础。但是，要想成为各种对话系统中能够普遍使用的健壮的理解系统，还有大量工作函待完成。处理连词和量词是我们当前工作的一个主要课题，以及使用PLN处理比本文中总结的简单推断场合更巧妙的常识性推断。此外，正如第？？章中描述的那样，我们正在研究采用学习到的规则（基于无人参与的语料库分析）来替代构成我们理解系统的人工编码的规则。

**5 对语言理解的输出进行推理**

本节阐述我们为了研究引言中所述的假设2（关于基于语言的推理）所完成的工作。假设2认为，结合上述自然语言理解框架输出的逻辑表示，可以进行各种体现人类常识性推理能力的简单逻辑推断。基于自然语言输入进行推断对于许多应用都非常重要，包括将在第？？章中讨论的自然语言对话。

**5.1 关于比较级的推理**

首先，按照上述关于比较级语法的讨论，我们简要地演示一些PLN对于比较级句子处理后的输出进行推断的例子。这用来演示在去除句法复杂性和实现正常逻辑形式后比较级推理的简单性。

确切地讲，考虑我们集成的系统如何接受以下输入

• Bob likes Hendrix more than the Beatles

• Bob is American

• Menudo is liked less by Americans than the Beatles

并得到结论：Bob likes Hendrix more than Menudo。

对于第一个句子，我们得到

\_subj(like , Bob)

\_obj(like , Hendrix)

than(Hendrix , Beatles)

\_comparative(like , Hendrix)

==>

TruthValueGreaterThanLink

EvaluationLink like Bob Hendrix

EvaluationLink like Bob Beatles

对于第二个句子，我们得到

\_subj(like , Americans)

\_obj(like , Menudo)

than(Beatles , Menudo)