

编译原理课程设计--生物可解释解析器

陈俊皓

chenjh535@mail2.sysu.edu.cn

May 9, 2024



1. 编译原理课程设计规格说明总览

2. 自然语言解析过程与编译解析过程

2.1. 自然语言与形式语言的相似性

2.2. 词性标注过程与词法解析过程

2.3. 自然语言中的依赖解析 (dependency parsing) 过程与编译过程语法解析过程

3. 集合演算基础运算逻辑

4. 基于原论文给出的伪代码实现自然语言依赖解析

5. 附加选项

6. 参考文献

基本要求

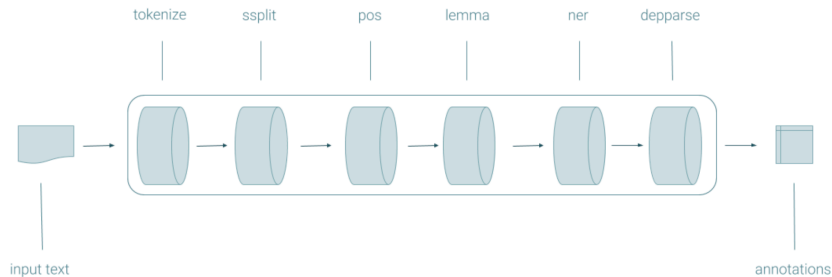
本课程设计要求同学们利用大脑计算中所提出的集合演算算法，利用 C/C++ 实现该算法 (可参考原论文所提供的源代码)，并利用该算法实现自然语言的依赖解析过程 (参考原论文读出依赖过程的伪代码)。

附加要求

推荐同学们实现该算法的 java 版本，利用 stanfordCoreNLP 中所提供的 (tagger 和 parser) 接口扩展原算法的可用性。鉴于 stanfordCoreNLP 为 java 编写且项目代码量较大，该要求不强制，仅作为加分项。

1. 编译原理课程设计规格说明总览
2. 自然语言解析过程与编译解析过程
 - 2.1. 自然语言与形式语言的相似性
 - 2.2. 词性标注过程与词法解析过程
 - 2.3. 自然语言中的依赖解析 (dependency parsing) 过程与编译过程语法解析过程
3. 集合演算基础运算逻辑
4. 基于原论文给出的伪代码实现自然语言依赖解析
5. 附加选项
6. 参考文献

在传统的自然语言解析中，会将一个较为复杂的任务划分多个依赖关系的很多子任务，在 stanfordNLP 划分如下图所示：



这些子任务通常按照顺序依次进行，形成一条数据流动链 (pipeline)，以逐步深入理解和处理自然语言文本。

自然语言解析过程与编译解析过程

自然语言与形式语言的相似性



自然语言与形式语言虽然在二义性等方面有着显著的区别，但是二者在语法规则和语义解析上有着相似之处。

- 二者都具有语法规则，用于描述有效的句子或表达式结构。它们都依赖于一定的规则和约定来构成合法的语言结构。
- 二者都有语义，即语言单位的意义或解释。它们都使用特定的规则和约定来赋予语言单位以含义。

Google 的技术文档编写教程

If you change the name of a variable midway through a method, your code won't compile. Similarly, if you rename a term in the middle of a document, your ideas won't compile (in your users' heads).

可以看到，其中将大脑解析自然语言的过程类比为编译器编译的过程，这也印证了编译过程和大脑解析自然语言过程的相似性。

自然语言解析过程与编译解析过程

词性标注过程与词法解析过程



中山大学
SUN YAT-SEN UNIVERSITY

自然语言中的词性 (part of speech) 与编译原理词法解析阶段中的 token

自然语言处理中，第一步往往是对于最小语义单元 (i.e 在英语中为每个单词) 标注词性信息，随后在利用解析结果进行后续的语法解析与语义解析操作；而在编译原理中，第一步即为利用正则语言解析的方法对源代码进行词法解析，生成所需要的 tokens 序列为后续的语法解析过程服务。

在英语中，词性 (part of speech) 有以下属性：

Part of Speech	Definition
Noun	A person, place, concept, or thing
Pronoun	A noun that replaces another noun (or larger structure)
Adjective	A word or phrase that modifies a noun
Adverb	A word or phrase that modifies a verb, an adjective, or another adverb
Preposition	A word or phrase specifying the positional relationship of two nouns
Conjunction	A word that connects two nouns or phrases
Transition	A word or phrase that connects two sentences

在 C 语言中，则有如下的属性：

Attribution	Definition
Keywords	The keywords are pre-defined or reserved words in a programming language.
Identifiers	Identifiers are used as the general terminology for the naming of variables, functions, and arrays.
Constants	The constants refer to the variables with fixed values.
Strings	Strings are nothing but an array of characters ended with a null character ('\0').
Special Symbols	Special symbols are used in C having some special meaning and thus, cannot be used for some other purpose.(e.g. <code>[]</code> , <code>{}</code>)
Operator	Operators are symbols that trigger an action when applied to C variables and other objects.(Binary or Unary)

自然语言解析过程与编译解析过程

词性标注过程与词法解析过程



中山大学
SUN YAT-SEN UNIVERSITY

可以看到，无论是自然语言处理中还是编译过程中，针对最小语法单元有着相似的处理。而其处理结果也有着相似之处。

利用 `clang++ -fsyntax-only -Xclang -dump-tokens *.cpp`，我们可以得到如下的输出：

```
...
l_paren '(' Loc=</usr/local/opt/llvm@17/bin/../include/c++/v1/__verbose_abort:25:28>

const 'const' Loc=
</usr/local/opt/llvm@17/bin/../include/c++/v1/__verbose_abort:25:29>

char 'char' [LeadingSpace] Loc=
</usr/local/opt/llvm@17/bin/../include/c++/v1/__verbose_abort:25:35>

star '*' [LeadingSpace] Loc=
</usr/local/opt/llvm@17/bin/../include/c++/v1/__verbose_abort:25:40>

identifier '__format' Loc=
</usr/local/opt/llvm@17/bin/../include/c++/v1/__verbose_abort:25:41>

comma ',' Loc=</usr/local/opt/llvm@17/bin/../include/c++/v1/__verbose_abort:25:49>

ellipsis '...' [LeadingSpace] Loc=
</usr/local/opt/llvm@17/bin/../include/c++/v1/__verbose_abort:25:51>

r_paren ')' Loc=</usr/local/opt/llvm@17/bin/../include/c++/v1/__verbose_abort:25:54>

semi ';' Loc=</usr/local/opt/llvm@17/bin/../include/c++/v1/__verbose_abort:25:55>

r_brace '}' [StartOfLine] Loc=
</usr/local/opt/llvm@17/bin/../include/c++/v1/__verbose_abort:58^C/usr/local/opt/llv
m@17/bin/../include/c++/v1/__config:846:37
...
```


自然语言解析过程与编译解析过程

词性标注过程与词法解析过程



中山大学
SUN YAT-SEN UNIVERSITY

类似，在自然语言处理中，一般为标注单词的词性。二者的输出实际上相当相似。在 stanfordCoreNLP 框架中，其在 tokenization 中的输出如下：

```
Input : <<
Scores of properties are under extreme fire threat as a huge blaze
continues to advance through Sydney's north-western suburbs. Fires
have also shut down the major road and rail links between Sydney and
Gosford.

The promotional stop in Sydney was everything to be expected for a
Hollywood blockbuster - phalanxes of photographers, a stretch limo to
a hotel across the Quay - but with one difference. A line-up of
masseurs was waiting to take the media in hand. Never has the term
"massaging the media" seemed so accurate.

Output : >>
Scores/NNS of/IN properties/NNS are/VBP under/IN extreme/JJ fire/NN threat/NN as/IN
a/DT huge/JJ blaze/NN continues/VBZ to/TO advance/VB through/IN Sydney/NNP 's/POS
north/NN -/: western/JJ suburbs/NNS ./
Fires/VBZ have/VBP also/RB shut/VBN down/IN the/DT major/JJ road/NN and/CC rail/NN
links/NNS between/IN Sydney/NNP and/CC Gosford/NNP ./
The/DT promotional/JJ stop/NN in/IN Sydney/NNP was/VBD everything/NN to/TO be/VB
expected/VBN for/IN a/DT Hollywood/NNP blockbuster/NN -/: phalanxes/NNS of/IN
photographers/NNS ,/, a/DT stretch/NN limo/NN to/TO a/DT hotel/NN across/IN the/DT
Quay/NNP -/: but/CC with/IN one/CD difference/NN ./
A/DT line/NN -/: up/IN of/IN masseurs/NNS was/VBD waiting/VBG to/TO take/VB the/DT
media/NNS in/IN hand/NN ./
Never/RB has/VBZ the/DT term/NN "' massaging/VBG the/DT media/NNS "'` seemed/VBD
so/RB accurate/JJ ./
```

Copy

此处的标签代表的是在自然语言词性标注的 45-tag Penn Treebank tagset.

45 tag penn treebank tagset

An important tagset for English is the 45-tag Penn Treebank tagset (Marcus et al., 1993), shown below, which has been used to label many corpora. In such labelings, parts of speech are generally represented by placing the tag after each word, delimited by a slash:

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	<i>and, but, or</i>	PDT	predeterminer	<i>all, both</i>	VBP	verb non-3sg present	<i>eat</i>
CD	cardinal number	<i>one, two</i>	POS	possessive ending	<i>'s</i>	VBZ	verb 3sg pres	<i>eats</i>
DT	determiner	<i>a, the</i>	PRP	personal pronoun	<i>I, you, he</i>	WDT	wh-determ.	<i>which, that</i>
EX	existential 'there'	<i>there</i>	PRP\$	possess. pronoun	<i>your, one's</i>	WP	wh-pronoun	<i>what, who</i>
FW	foreign word	<i>mea culpa</i>	RB	adverb	<i>quickly</i>	WP\$	wh-possess.	<i>whose</i>
IN	preposition/ subordin-conj	<i>of, in, by</i>	RBR	comparative adverb	<i>faster</i>	WRB	wh-adverb	<i>how, where</i>
JJ	adjective	<i>yellow</i>	RBS	superlatv. adverb	<i>fastest</i>	\$	dollar sign	<i>\$</i>
JJR	comparative adj	<i>bigger</i>	RP	particle	<i>up, off</i>	#	pound sign	<i>#</i>
JJS	superlative adj	<i>wildest</i>	SYM	symbol	<i>+, %, &</i>	"	left quote	<i>' or "</i>
LS	list item marker	<i>1, 2, One</i>	TO	"to"	<i>to</i>	"	right quote	<i>' or "</i>
MD	modal	<i>can, should</i>	UH	interjection	<i>ah, oops</i>	(left paren	<i>[, (, {, <</i>
NN	sing or mass noun	<i>llama</i>	VB	verb base form	<i>eat</i>)	right paren	<i>],), }, ></i>
NNS	noun, plural	<i>llamas</i>	VBD	verb past tense	<i>ate</i>	,	comma	<i>,</i>
NNP	proper noun, sing.	<i>IBM</i>	VBG	verb gerund	<i>eating</i>	.	sent-end punc	<i>. ! ?</i>
NNPS	proper noun, plu.	<i>Carolinas</i>	VBN	verb past part.	<i>eaten</i>	:	sent-mid punc	<i>: ; ... - -</i>

依赖解析 (dependency parsing)

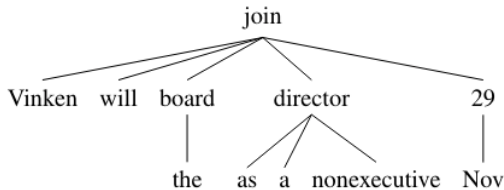
自然语言中的依赖解析 (dependency parsing) 是一种仅仅使用通过句子中的单词 (或词性) 以及单词之间相关的有向二元语法关系的语法解析过程, 其是句法分析的其中一种表现形式。

实际上, 无论是在自然语言处理还是编译过程语法解析中, 我们一般都选用树状结构作为最终输出的表现形式。而在依赖解析过程中, 其输入一般为完成词性标注过程的 token, 输出为以短语或单词为节点的多叉树, 这与编译原理中的语法解析以句法解析后的 token 作为输入, 并以 AST 作为输出的形式是颇为相似的。

为了更加直观的显示二者的相似关系，此处直接比较二者的树状输出结果。在终端中输入 `clang++ -Xclang -ast-dump -fsyntax-only *.cpp`，我们可以看到：

```
-NamespaceDecl 0x7fcd0a0e5900 prev 0x7fcd0a0e5800 </usr/local/opt/llvm@17/bin/../include/c++/v1/___config:846:39, line:846:49 std
|-original Namespace 0x7fcd0a0e58d8 'std'
|-NamespaceDecl 0x7fcd0a0e59f0 prev 0x7fcd0a0e5910 <col:55, line:846:37> /usr/local/opt/llvm@17/bin/../include/c++/v1/___config_site:13:31 __1 inline
|-original Namespace 0x7fcd0a0e59d8 '__1'
|-FunctionTemplateDecl 0x7fcd0a0e6118 </usr/local/opt/llvm@17/bin/../include/c++/v1/___utility/exchange.h:28:1, line:36:1> line:38:5 exchange
|-TemplateTypeParmDecl 0x7fcd0a0e5d00 <line:28:18, col:15> col:15 referenced class depth 0 index 0 __T1
|-TemplateTypeParmDecl 0x7fcd0a0e5e08 <col:21, col:33> col:27 referenced class depth 0 index 1 __T2
|-TemplateArgument type '__T1'
|-TemplateTypeParmType 0x7fcd0a0e5e08 '__T1' dependent depth 0 index 0
|-TemplateTypeParm 0x7fcd0a0e5e08 '__T1'
|-FunctionDecl 0x7fcd0a0e6000 <line:29:1, line:36:1> line:38:5 exchange '__T1 (&__T1, __T2 &&) noexcept(is_nothrow_move_constructible<__T1>::value && is_nothrow_assignable<__T1 &, __T2>::value)' inline
|-ParmVarDecl 0x7fcd0a0e5c08 <col:14, col:19> col:19 referenced __obj '__T1 &'
|-ParmVarDecl 0x7fcd0a0e5ca8 <col:26, col:32> col:32 referenced __new_value '__T2 &&'
|-CompoundStmt 0x7fcd0a0e6000 <line:32:1, line:36:1>
|-DeclStmt 0x7fcd0a0e6000 <line:33:5, col:41>
|-VarDecl 0x7fcd0a0e62d8 <col:15, col:40> col:9 referenced __old_value '__T1' nrv0 cinit
|-CallExpr 0x7fcd0a0e63d8 </usr/local/opt/llvm@17/bin/../include/c++/v1/___config:847:17, /usr/local/opt/llvm@17/bin/../include/c++/v1/___utility/exchange.h:33:40> '<dependent type>'
|-UnresolvedLookupExpr 0x7fcd0a0e6378 </usr/local/opt/llvm@17/bin/../include/c++/v1/___config:847:17, /usr/local/opt/llvm@17/bin/../include/c++/v1/___utility/exchange.h:33:38> '<overloaded function types> lvalue (no ADL) = 'move' 0x7fcd0a0e67cc0
|-DeclRefExpr 0x7fcd0a0e6308 <col:13> '__T1' lvalue ParmVar 0x7fcd0a0e5c08 '__obj' '__T1 &'
|-BinaryOperator 0x7fcd0a0e6548 <line:34:1, col:44> '<dependent type>' '*'
|-DeclRefExpr 0x7fcd0a0e6418 <col:8> '__T1' lvalue ParmVar 0x7fcd0a0e5c08 '__obj' '__T1 &'
|-CallExpr 0x7fcd0a0e6528 </usr/local/opt/llvm@17/bin/../include/c++/v1/___config:847:17, /usr/local/opt/llvm@17/bin/../include/c++/v1/___utility/exchange.h:34:44> '<dependent type>'
|-UnresolvedLookupExpr 0x7fcd0a0e6488 </usr/local/opt/llvm@17/bin/../include/c++/v1/___config:847:17, /usr/local/opt/llvm@17/bin/../include/c++/v1/___utility/exchange.h:34:31> '<dependent type>' lvalue (no ADL) = 'forward' 0x7fcd0a0e690158 0x7fcd0a0e69080
|-DeclRefExpr 0x7fcd0a0e6508 <col:33> '__T2' lvalue ParmVar 0x7fcd0a0e5ca8 '__new_value' '__T2 &&'
|-ReturnStmt 0x7fcd0a0e6588 <line:35:5, col:12>
|-DeclRefExpr 0x7fcd0a0e6568 <col:22> '__T1' lvalue Var 0x7fcd0a0e62d8 '__old_value' '__T1'
|-VisibilityAttr 0x7fcd0a0e6178 </usr/local/opt/llvm@17/bin/../include/c++/v1/___config:703:64, col:72> Hidden
|-ExcludeFromExplicitInstantiationAttr 0x7fcd0a0e6108 <line:745:72>
-AbiTagAttr 0x7fcd0a0e6578 <line:820:26, col:77> uel70006
```

而在依赖解析中的树状输出则如下所示 (其一般以谓语作为根节点):



1. 编译原理课程设计规格说明总览
2. 自然语言解析过程与编译解析过程
 - 2.1. 自然语言与形式语言的相似性
 - 2.2. 词性标注过程与词法解析过程
 - 2.3. 自然语言中的依赖解析 (dependency parsing) 过程与编译过程语法解析过程
3. 集合演算基础运算逻辑
4. 基于原论文给出的伪代码实现自然语言依赖解析
5. 附加选项
6. 参考文献

集合演算是应用于依赖解析中的方法，从原论文的伪代码中我们可以得知，该模型需要输入文本的词性标注信息（词性以及句子成分）。而后，通过其定义的大脑计算模型，解析出原句子中的依赖关系。

其中涉及的实体以及实体间的运算关系如下：

brain: 整个大脑被定义为一个概率连接有向图 $G_{n,p}$ ，其中划分为多个脑区。脑区为其子图，脑区内的节点以 p 的概率随机连接；脑区间的连接则成为 **fiber**，如果两个脑区之间有 **fiber**，则两个脑区间的神经元可以以概率 p 进行连接。在整个动力系统（在突触权重更新的情况下，可以建立该系统的微分方程或差分方程表示）运行时，假设其以 **timestep** 的离散时间步进行，在每个时间步会进行突触输入计算，选举 cap 以及权重等操作。

area: 脑区拥有两种状态，抑制或者解除抑制。在抑制状态下保留其 cap ，在解除抑制状态下则会通过计算突触输入得到新的 cap 。

assemblies: 一个脑区中神经元的子集，且具有固定的大小 k （一般为未抑制脑区中活跃度最高的 k 个神经元）。

projections: $project(x, B, y)$, 一个位于 A 脑区中的神经元集合 x 通过 **fiber** 将自身的信息 **project**(投影) 到脑区 B 中的神经元集合 y , 可以称 $x = parent(y)$ 。具体操作是利用已稳定的 x 神经元集, 通过 A 和 B 间的连接改变 B 中神经元的活跃度, 在一定的时间步后, 得到更新后 B 中最活跃的 k 个神经元组成 y 神经元集合。

神经元兴奋度 (突触输入 SI): 对于每一个神经元而言, 其通过突触连接 (有向边) 计算兴奋度的公式如下:

$$SI(i, t + 1) = \sum_{(j,i) \in E} fires(j, t) w_{ji}(t) \quad (1)$$

cap: 一个脑区中活跃度最高的 k 个神经元称为该脑区在对应时间内的 **cap**, 其中神经元的状态则为 **fired** 兴奋。设时间步 t 时活跃的神经元集为 A_t , 则定义 **core neurons** 为 $A_t \cap A_{t+1}$, 即在相邻时间步中都为 **cap** 的神经元。

支持集: 一个脑区的支持集定义为在所有时间步中曾经活跃过的神经元集的并集，其数学表示为: $A^* = \cup_t A_t$ 。

Hebbian plasticity(赫布可塑性): 在该系统中，如果存在突触连接的神经元，源节点在时间步 t 兴奋 (*fired*)，目标节点在时间步 $t + 1$ 兴奋，则二者之间的权重增加为原来的 $1 + \beta$ 。 (β 为人为设定，后面会看到其与支持集密切相关)

1. 编译原理课程设计规格说明总览
2. 自然语言解析过程与编译解析过程
 - 2.1. 自然语言与形式语言的相似性
 - 2.2. 词性标注过程与词法解析过程
 - 2.3. 自然语言中的依赖解析 (dependency parsing) 过程与编译过程语法解析过程
3. 集合演算基础运算逻辑
4. 基于原论文给出的伪代码实现自然语言依赖解析
5. 附加选项
6. 参考文献

课程设计要求

同学们需要根据集合演算中提供的伪代码实现其依赖解析的 CPP 版本。在论文所给出的代码仓库中，实际上给出了一个较为简单的 CPP 版本代码，但其尚为能完成解析的过程。同学们需要重构其中的代码，并使其可以完成简单的依赖解析。

在提交作业的时候，需要同学们提交以下项目：

- 项目源代码文件夹，需要支持 clang++ 或 g++ 编译器，建议使用 C++20 版本。原论文的代码仓库使用的是 bazel 构建项目文件，此处建议同学们使用 cmake 这一较为传统的工具管理源文件和目标文件。
- 项目所支持的单词库文件，只需罗列出所有支持的可解析单词。
- 项目测试代码文件夹，需要使用 google test 验证其可以完成单词库中所有单词在满足英语语法规则下的句子均可解析成功，助教会验证相关结果。
- 项目性能测试文件夹，该文件夹需要测试同学们所编写的解析器的解析性能，最终的结果为解析时长/句子单词数。
- 项目实验报告文件夹，此处的报告推荐使用 markdown 或者 latex 编写，并转化为 pdf 版后提交。在实验报告中需要阐明：
 - 小组内组员的组员的分工；
 - 对于依赖解析以及集合演算模型的认识（类似读书报告的形式）；
 - 项目设计的思路，包括设计的数据对象，类之间的交互关系，如何保证线程安全，如何提升代码性能等；


截止时间:

- 5.23: 对于依赖解析以及集合演算模型的认识 (类似读书报告的形式), 以第 {} 组-{}组长学号}-{}组长姓名}-编译原理课程设计-依赖解析-集合演算.zip 的格式发送到 chenjh535@mail2.sysu.edu.cn 邮箱;
- 6.6: 整体项目, 以第 {} 组-{}组长学号}-{}组长姓名}-编译原理课程设计.zip 格式发送到 chenjh535@mail2.sysu.edu.cn 邮箱;

1. 编译原理课程设计规格说明总览
2. 自然语言解析过程与编译解析过程
 - 2.1. 自然语言与形式语言的相似性
 - 2.2. 词性标注过程与词法解析过程
 - 2.3. 自然语言中的依赖解析 (dependency parsing) 过程与编译过程语法解析过程
3. 集合演算基础运算逻辑
4. 基于原论文给出的伪代码实现自然语言依赖解析
5. 附加选项
6. 参考文献

完成了以上任务的同学可以考虑将集合演算耦合到 stanfordCoreNLP 的依赖解析模块中，利用 stanfordCoreNLP 提供的词性标注模型，提升集合演算的解析能力。

1. 编译原理课程设计规格说明总览
2. 自然语言解析过程与编译解析过程
 - 2.1. 自然语言与形式语言的相似性
 - 2.2. 词性标注过程与词法解析过程
 - 2.3. 自然语言中的依赖解析 (dependency parsing) 过程与编译过程语法解析过程
3. 集合演算基础运算逻辑
4. 基于原论文给出的伪代码实现自然语言依赖解析
5. 附加选项
6. 参考文献

-  MITROPOLSKY, D., COLLINS, M. J., AND PAPADIMITRIOU, C. H.
A biologically plausible parser.
CoRR abs/2108.02189 (2021).

Thank you for your attention!

Q&A