University of Toronto, Department of Computer Science
CSC 485H/2501H: Computational linguistics, Fall 2023

Assignment 3

Due date: 23:59 on Thursday, December 7, 2023.
Late assignments will not be accepted without a valid medical certificate or other documentation of an emergency.
For CSC485 students, this assignment is worth 33% of your final grade.
For CSC2501 students, this assignment is worth 25% of your final grade.

- **Read the whole assignment carefully.**
- What you turn in must be your own work. You may not work with anyone else on any of the problems in this assignment. If you need assistance, contact the instructor or TA for the assignment.
- Any clarifications to the problems will be posted on the Piazza forum for the class. You will be responsible for taking into account in your solutions any information that is posted there, or discussed in class, so you should check the page regularly between now and the due date.
- The starter code directory for this assignment is accessible on Teaching Labs machines at the path `/u/csc485h/fall/pub/a3/`. In this handout, code files we refer to are located in that directory.
- When implementing code, make sure to **read the docstrings** as some of them provide important instructions, implementation details, or hints.
- Fill in your name, student number, and UTORid on the relevant lines at the top of each file that you submit. (Do not add new lines; just replace the `NAME`, `NUMBER`, and `UTORid` placeholders.)

Overview: Symbolic Machine Translation

In this assignment, you will learn how to write phrase structure grammars for some different linguistic phenomena in two different languages: English and Chinese. You can use the two grammars to create an *interlingual machine translation system* by parsing in one, and generating in the other. Don't panic if you don't speak Chinese, and also don't cheer up yet if you can speak the language — it won't give you much of an advantage over other students. A facility with languages in general will help you, as will the ability to learn and understand the nuances between the grammars of two different languages. In particular, you will start by working on agreement. Then, you will need to analyse the quantifier scoping difference between the two languages.

TRALE Instructions The TRALE system can be run with:

```
/h/u2/csc485h/fall/pub/trale/trale -fsug
```

(which you are welcome to alias). For this assignment, TRALE needs to start a graphical interface: Gralej. Therefore, if you don't have access to the labs and want to run TRALE remotely, you can either use:

- Remote Labs: X2Go (RECOMMENDED)
  https://www.teach.cs.toronto.edu/using-labs/remote-labs-x2go/;
- RDP over SSH
  https://www.teach.cs.toronto.edu/using-labs/remote-labs-rdp-over-ssh/;
- or connect to `teach.cs` using ssh with either the `-X` or `-Y` flag:
  `ssh -X myutorid@teach.cs.toronto.edu`

1. Agreement: Determiners, Numbers and Classifiers [10 marks]

English expresses subject–verb agreement in person and number. English has two kinds of number: singular and plural. The subject of a clause must agree with its predicate: they should be both singular or both plural. However, the number of a direct object does not need to agree with anything.

(1) A programmer annoys a dolphin.

(2) Two programmers annoy a dolphin.

(3) * Two programmers annoys two dolphins.

(4) * A programmer annoy two dolphins.

Chinese, on the other hand, does not exhibit subject–verb agreement. As shown in the examples below, most nouns do not inflect at all for plurality. Chinese does, however, have a classifier (CL) part of speech that English does not. Semantically, classifiers are similar to English collective nouns (a *bottle* of water, a *murder* of crows), but English collective nouns are only used when describing collectives. With very few exceptions, classifiers are mandatory in complex Chinese noun phrases. Different CLs agree with different classes of nouns that are sorted by mostly semantic criteria. For example, 程序员 (chengxuyuan)[1] *programmer* is a person and an occupation, so it should be classified by either 个 (ge) or 名 (ming) and cannot be classified by the animal CL 只 (zhi). However, the rules of determining a noun's class constitute a formal system that must be followed irrespective of semantic similarity judgements. For example, while geese and fish are both animals and can both be classified by the animal CL 只 (zhi), 鱼 (yu) *fish* can take another classifier, 条 (tiao), for livestock.

(5) 一 个 程序员
yi ge chengxuyuan
one *ge*-CL programmer

(6) 两 个 程序员
liang ge chengxuyuan
two *ge*-CL programmer

(7) 三 个 程序员
san ge chengxuyuan
three *ge*-CL programmer

(8) * 三 程序员
san chengxuyuan
three programmer

(9) * 三 只 程序员
san zhi chengxuyuan
three *zhi*-CL programmer

(10) 一 只 鹅
yi zhi e
one *zhi*-CL goose

(11) 两 只 鹅
liang zhi e
two *zhi*-CL goose

(12) 三 只 鹅
san zhi e
three *zhi*-CL goose

(13) * 三 条 鹅
san tiao e
three *tiao*-CL goose

(14) * 三 名 鹅
san ming e
three *ming*-CL goose

[1] Use either Chinese characters or the Romanized form, but with no spaces or hyphens, e.g., `chengxuyuan`, for multi-character lexical entries.

You should be familiar by now with the terminology in the English grammar starter code for this question. The Chinese grammar is fairly similar, but there is a new phrasal category called a classifier phrase (CLP), formed by a number and a classifier. The classifier phrase serves the same role as a determiner does in English.

The two grammars below don't appropriately constrain the NPs generated. You need to design your own rules and features to properly enforce agreement.

English Grammar:

Rules:

NP → Det N
NP → Num N
VP → V NP
S → NP VP

Lexicon:

a:	Det
one:	Num
two:	Num
three:	Num
programmer:	N
programmers:	N
goose:	N
geese:	N
fish:	N
fish:	N
observe:	V
observes:	V
observed:	V
catch:	V
catches:	V
caught:	V

Chinese Grammar:

Rules:

CLP → Num CL
NP → CLP N
VP → V NP
S → NP VP

Lexicon:

一	yi *one/a*	Num
两	liang *two*	Num
三	san *three*	Num
程序员	chengxuyuan *programmer*	N
鹅	e *goose*	N
鱼	yu *fish*	N
观察	guancha *observe*	V
抓到	zhuadao *catch*	V
个	ge	CL
名	ming	CL
只	zhi	CL
条	tiao	CL

Here is a list of all of the nouns in this question and their acceptable classifiers:

- 鹅 e *goose*: 只 zhi;
- 鱼 yu *fish*: 只 zhi, 条 tiao;
- 程序员 chengxuyuan *programmer*: 个 ge, 名 ming.

(a) (6 marks) Implement one grammar for each language pursuant to the specifications above. English: `q1_en.pl` and Chinese: `q1_zh.pl`.

Neither of your grammars need to handle embedded clauses, e.g., *a programmer caught two geese observe a fish.* Similarly for Chinese, your grammar doesn't need to parse sentences like example (15):

(15) 一名 程序员 抓到 两 只 鹅 观察 一条 鱼
yi ming chengxuyuan zhuadao liang zhi e guancha yi tiao yu
A programmer caught two geese observe a fish.

For the Chinese grammar, the lexical entries can be coded in either pinyin (the Romanized transcriptions of the Chinese characters) or in simplified Chinese characters.

(b) (4 marks) Use your grammars to parse and translate the following sentences. Save and submit all the translation results in the `.grale` format. The results of sentence (16) should be named `q1b_en.grale` and the results of sentence (17) should be named `q1b_zh.grale`.

(16) Two geese caught one fish

(17) 两 个 程序员 观察 三 条 鱼
liang ge chengxuyuan guancha san tiao yu

Operational Instructions

- If you decide to use simplified Chinese characters, enter them in Unicode and use the `-u` flag when you run TRALE.
- Independently test your grammars in TRALE first, before trying to translate.

- Use the function `translate` to generate a semantic representation of your source sentence. If your sentence can be parsed, the function `translate` should open another `gralej` interface with all of the translation results.

```
| ?- translate([two,geese,observe,one,fish]).
```

- To save the translation results, on the top left of the `Gralej` window (the window with the `INITIAL CATEGORY` entry and all of the translated sentences listed), click `File >> Save all >> TRALE format`.
- Don't forget to **close all of the windows** or kill both of the `Gralej` processes after you finish. Each `Gralej` process will take up one port in the server, and no one can use the server if we run out of ports.

2. Quantifier Scope [30 marks]

For this assignment, we will consider two quantifiers: the universal quantifier (*every*, 每 mei) and the existential quantifier (*a*, 一 yi). In English, both quantifiers behave as singular determiners.

(18) A professor stole every cookie.

(19) * A professor stole every cookies.

(20) * A professors stole every cookie.

In Chinese, both of these quantifiers behave more like numerical determiners. In addition, when a universal quantifier modifies an NP that occurs before the verb (such as with a universally quantified subject), the preverbal operator 都 (dou) is required. When a universally quantified NP occurs after the verb, the dou-operator must not appear with it.

(21) Every professor stole a cookie.

(22) A professor stole every cookie.

(23) 每 个 教授 都 偷了 一块 饼干
mei ge jiaoshou *dou* tou-le yi kuai binggan
∀ *ge*-CL professor DOU stole ∃ *kuai*-CL cookie

(24) * 每 个 教授 偷了 一块 饼干
mei ge jiaoshou tou-le yi kuai binggan
∀ *ge*-CL professor stole ∃ *kuai*-CL cookie

(25) 一个 教授 偷了 每 块 饼干
yi ge jiaoshou tou-le mei kuai binggan
∃ *ge*-CL professor stole ∀ *kuai*-CL cookie

(26) * 一个 教授 都 偷了 每 块 饼干
yi ge jiaoshou *dou* tou-le mei kuai binggan
∃ *ge*-CL professor *dou* stole ∀ *kuai*-CL cookie

Quantifier Scope Ambiguity In lecture, we talked about different kinds of ambiguity. In many English sentences, no matter what the order of the quantifiers, there is a quantifier scope ambiguity. For example, the sentence *every programmer speaks a language* has two readings:

- (∃ > ∀) Every programmer speaks a language. [The language is Inuktitut.]
- (∀ > ∃) Every programmer speaks a language. [Some programmers speak Inuktitut and some programmers speak Aymara.]

The symbol (∃ > ∀) means the existential quantifier outscopes the universal quantifier in a logical form representation of the sentence.

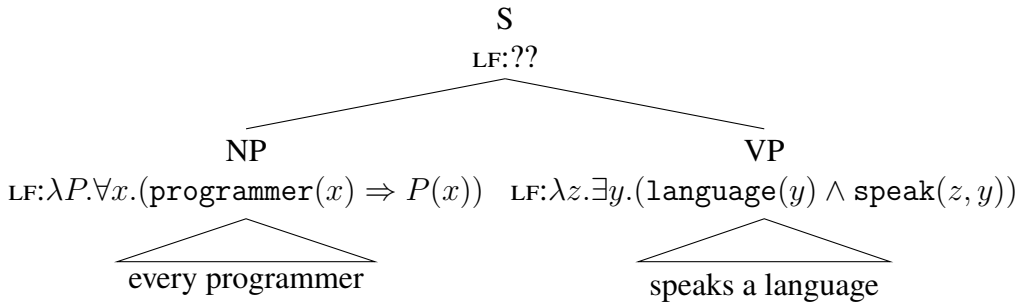

Figure 1: Beta Reduction. What should be the LF of S?

We can write the semantics of the two sentences in their logical forms (LF) to distinguish the two readings:

- $\exists y.(\texttt{language}(y) \wedge \forall x.(\texttt{programmer}(x) \Rightarrow \texttt{speak}(x,y)))$
- $\forall x.(\texttt{programmer}(x) \Rightarrow \exists y.(\texttt{language}(y) \wedge \texttt{speak}(x,y)))$

English sentences (27, 28) are scopally ambiguous no matter what the linear order of the quantifiers is. But in Chinese, a sentence is scopally ambiguous only when the universally quantified NP precedes the existential NP: (29) is ambiguous, but (30) is unambiguous.[2]

(27) Every programmer speaks a language
Ambiguous: $\exists > \forall, \forall > \exists$

(28) A programmer speaks every language
Ambiguous: $\exists > \forall, \forall > \exists$

(29) 每 个 程序员 都 会说 一 种 语言
mei ge chengxuyuan dou huishuo yi zhong yuyan
$\forall$ *ge*-CL programmer DOU speak $\exists$ *zhong*-CL language
Ambiguous: $\exists > \forall, \forall > \exists$

(30) 一 个 程序员 会说 每 种 语言
yi ge chengxuyuan huishuo mei zhong yuyan
$\exists$ *ge*-CL programmer speak $\forall$ *zhong*-CL language
Unambiguous: $\exists > \forall$

How can we derive the LF of the two readings? We use a process called *beta reduction*. Recall the lambda-calculus notation: $\lambda x.x^2$ denotes a function that takes a variable x, and returns the square of its value (x^2). After substituting the value for the bound variable x, we can reduce the function application in the body of the lambda term to a new expression. For example, applying 2 to $\lambda x.x^2$ will get us:

$$\lambda x.x^2(2) = 2^2$$

[2]The actual principles that determine the scopal readings of a Chinese sentence are still an active area of research, but this is obviously beyond the scope of this assignment. You only need to construct an analysis that explains every example mentioned in this assignment.

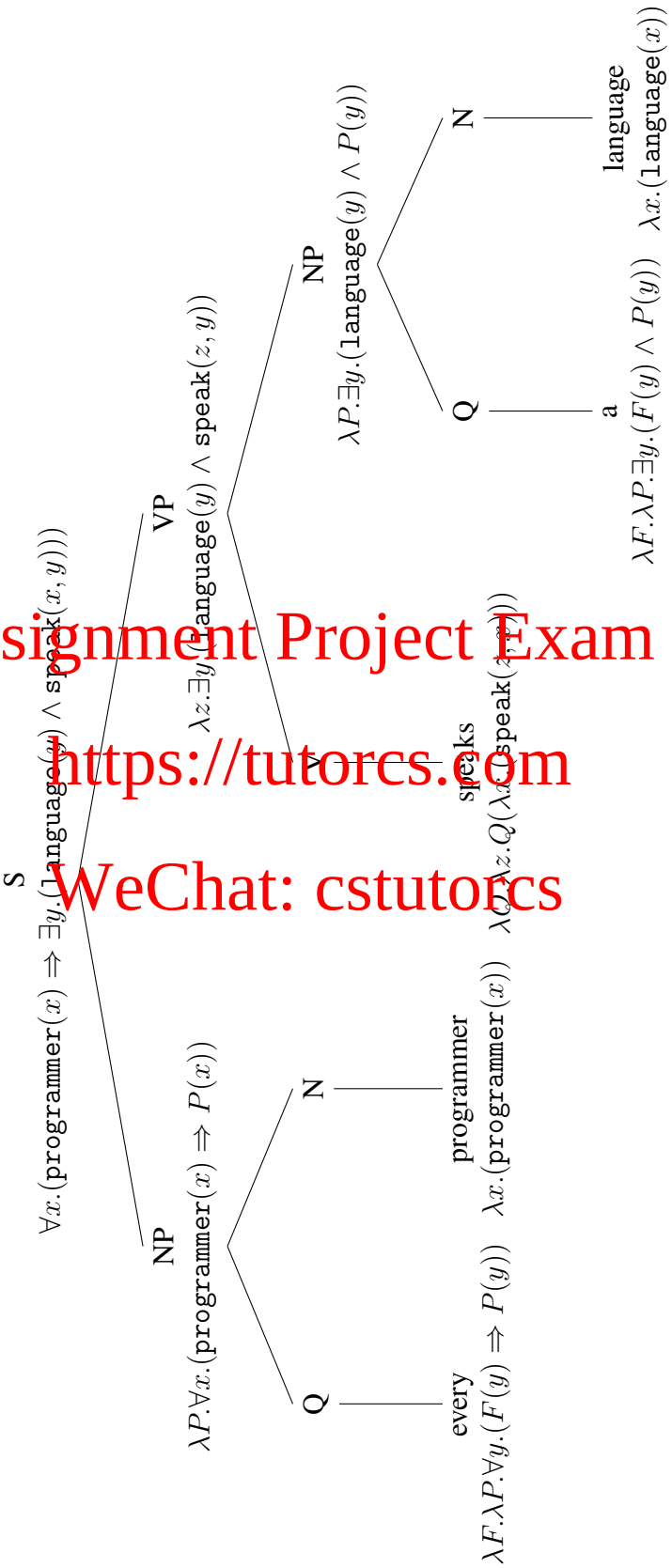

Figure 2: Beta reduction analysis of the sentence *every programmer speaks a language.*

This process is also known as beta reduction (denoted as $\Leftrightarrow_\beta$). Note that beta reduction itself does not tell us that this equals 4. That is obtained by a subsequent process of arithmetic evaluation. But we can use beta reduction even if we don't evaluate.

We can also perform beta reduction on variables for functions. For example, applying in $\lambda F.F(2)$ to $\lambda x.x^2$ will yield:

$$\lambda F.F(2)\ (\lambda x.x^2) = (\lambda x.x^2)(2) = 2^2$$

Now, let's look at an example that uses beta reduction to compute the LF of a sentence. For example, as shown in figure 1, we know that the LF of the NP *every programmer* is $\lambda P.\forall x.(\texttt{programmer}(x) \Rightarrow P(x))$ and the LF of the VP *speaks a language* is $\lambda z.\exists y.(\texttt{language}(y) \wedge \texttt{speak}(z, y))$. What is the LF of *every programmer speaks a language*?

$$\begin{aligned}
&\lambda P.\forall x.(\texttt{programmer}(x) \Rightarrow P(x))(\lambda z.\exists y.(\texttt{language}(y) \wedge \texttt{speak}(z, y))) \\
\Leftrightarrow_\beta\ &\forall x.(\texttt{programmer}(x) \Rightarrow \lambda z.\exists y.(\texttt{language}(y) \wedge \texttt{speak}(z, y))(x)) \\
\Leftrightarrow_\beta\ &\forall x.(\texttt{programmer}(x) \Rightarrow \exists y.(\texttt{language}(y) \wedge \texttt{speak}(x, y)))
\end{aligned}$$

Each step of repeatedly applying beta reduction to every subterm until we reach an irreducible statement is called *beta normalisation*.

Figure 2 shows the complete analysis of the sentence *every programmer speaks a language*. Familiarize yourself with every part of the analysis. But this only generates one of the two readings – the surface reading ($\forall > \exists$). We will use a technique called *quantifier storage* to capture the scopal ambiguity and make both readings available.

Quantifier Storage If quantifier scoping is a semantic effect, how do we represent it in syntax? When there is no ambiguity, keeping track of the quantifier scope is pretty straightforward. To keep track of and resolve scope ambiguities, we will use a list called a *quantifier store*. The idea behind QSTORE is that, instead of consuming all of the LF components right away, we can choose to keep them in QSTORE and apply them later.

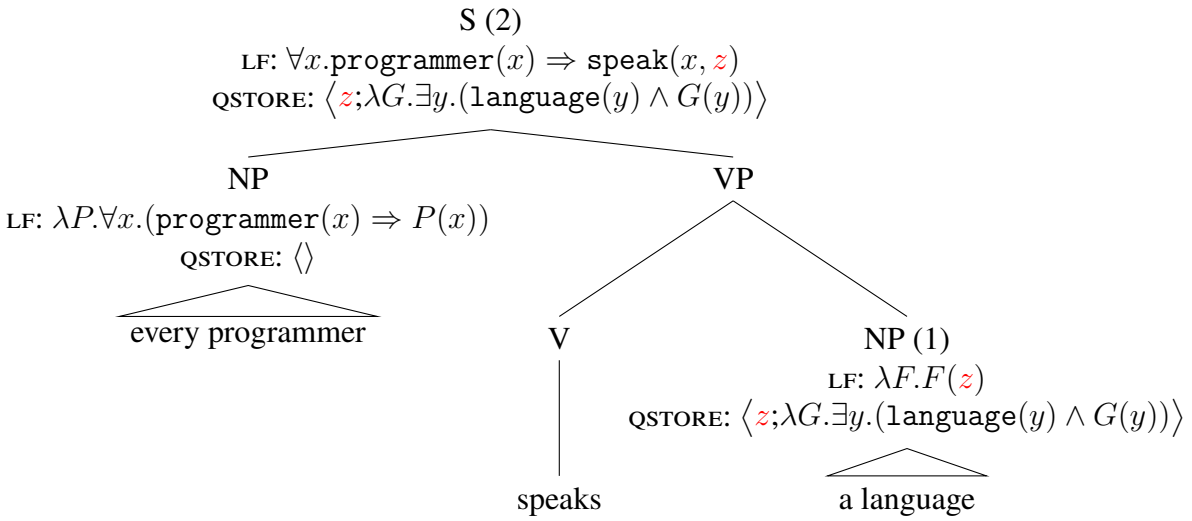

Figure 3: Quantifier Storage. Storing the quantifier at (1), and retrieve it later at (2).

Let's go back to the example, *every programmer speaks a language* (figure 3). We first store the LF of the NP *a language* at (1) and replace the LF of the NP with a placeholder $\lambda F.F(z)$. The variable z in this expression is a free occurrence, and it is the same variable as the z in the store and in the LF of the sentence (the free occurrences of z are highlighted in red). We retrieve the logical form from the store at (2). The retrieval process consists of three steps:

1. First, we construct a function $\lambda z.L_S$, where L_S is the current LF, and z is the variable paired in the QSTORE entry. In our particular case, this will yield $\lambda z.(\forall x, \texttt{programmer}(x) \Rightarrow \texttt{speak}(x, z))$.

2. Then, we apply this function to the LF from the QSTORE entry.

3. Finally, we beta normalise. Using beta normalisation, we obtain the second reading of the sentence.

$$\begin{aligned} & \lambda G.\exists y.(\texttt{language}(y) \wedge G(y))(\lambda z.(\forall x, \texttt{programmer}(x) \Rightarrow \texttt{speak}(x, z))) \\ \Leftrightarrow_\beta\ & \exists y.(\texttt{language}(y) \wedge (\lambda z.(\forall x.\texttt{programmer}(x) \Rightarrow \texttt{speak}(x, z))(y)) \\ \Leftrightarrow_\beta\ & \exists y.(\texttt{language}(y) \wedge (\forall x.\texttt{programmer}(x) \Rightarrow \texttt{speak}(x, y)) \end{aligned}$$

Topicalization and Movement Topicalization is a linguistic phenomenon in which an NP appears at the beginning of a sentence in order to establish it as the topic of discussion in a sentence or to emphasize it in some other way. It plays an important role in the syntax of fixed-word-order languages because grammatical function is mainly determined by word order. Both Chinese and English exhibit topicalization. The entire object NP, for example, can be moved to the beginning of the sentence in either language. But in Chinese, object topicalization is more restricted when the subject is quantified: it can happen when the subject is universally quantified, but not when it is existentially quantified (33-36).

(31) A language, every programmer speaks.
∃ language ∀ programmer speak
Ambiguous: ∀ > ∃ and ∃ > ∀

(32) Every language, a programmer speaks.
∀ language ∃ programmer speak
Ambiguous: ∀ > ∃ and ∃ > ∀

(33) 一种 语言 每 个 程序员 都 会说
yi zhong yuyan mei ge chengxuyuan dou huishuo
∃ *zhong*-CL language ∀ *ge*-CL programmer *dou* speak
Unambiguous: ∃ > ∀

(34) 每 种 语言 每 个 程序员 都 会说
mei zhong yuyan mei ge chengxuyuan dou huishuo
∀ *zhong*-CL language ∀ *ge*-CL programmer *dou* speak

(35) * 一种 语言 一个 程序员 会说
yi zhong yuyan yi ge chengxuyuan huishuo
∃ *zhong*-CL language ∃ *ge*-CL programmer speak

(36) * 每 种 语言 一 个 程序员 (都) 会说
mei ben shu yi ge chengxuyuan (dou) huishuo
∀ *ben*-CL language ∃ *ge*-CL programmer *dou* speak

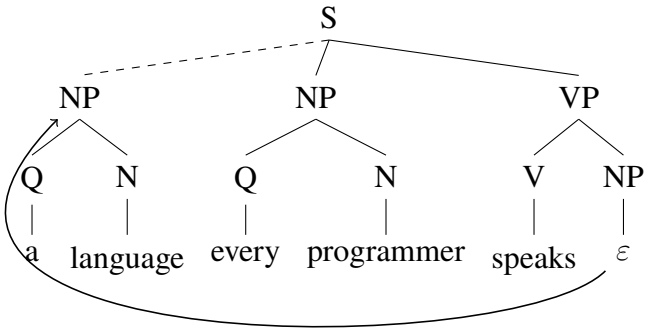

Figure 4: English topicalization parse tree: example (31).

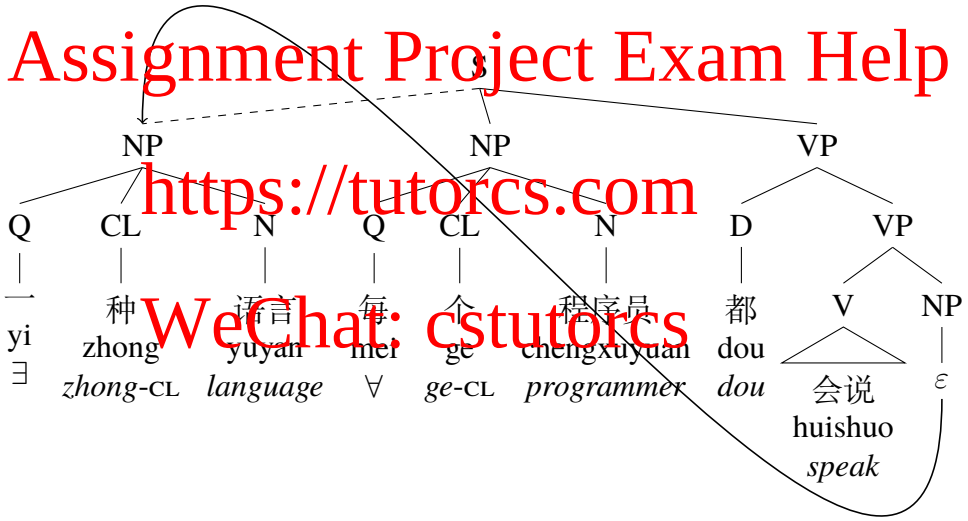

Figure 5: Chinese topicalization parse tree: example (33).

In English, neither subject–verb agreement nor quantifier scope ambiguity is generally affected by movement. In particular, the number and person of the subject should always agree with the predicate no matter where it occurs. Here, you can assume that Chinese also follows subject–verb agreement (regarding the requirement of *dou*) in the same way that English does. But whereas in English, both readings are still available after the sentences are topicalised (31, 32), this is not the case in Chinese. Compared to its untopicalised counterpart (29), the topicalised sentence (33) is no longer ambiguous.

Figures 4 and 5 show the parse trees of sentences (31) and (33). Topicalization is generally analysed with gaps. An empty trace is left in the untopicalized position of the object NP, where the gap is introduced. The gapped NP then percolates up the tree, and is finally unified with the topicalized NP at the left periphery of the sentence.[3]

[3] Although Chinese is an SVO (Subject-Verb-Object) language, there is a means of performing "double movement."

(a) (2 marks) Manually convert all readings of the sentences (28) and (30) to logical expressions. Put your logical forms in section 2(a) of `analysis.txt`. Use `exists` and `forall` for the quantifiers, and use `=>` and the caret symbol `^` for implication and conjunction.

(b) (10 marks) Implement grammars for the syntax of quantifier scope ambiguity. You don't need to account for meanings, or for ambiguity in meanings (there should be no syntactic ambiguities). At this point, a correct grammar will produce exactly **one** parse for every grammatical sentence. Test your implementation before you move on to the next step.

(c) (10 marks) Augment your grammars to represent meaning and quantifier scope ambiguity. Marks for question 2(b) will be deducted if your work on this part causes errors in the syntactic predictions. Your grammar should generate more than one parse for each ambiguous sentence.

(d) (4 marks) Translate sentences (28) and (30), as you did in the first question.

Operational Instructions

- Use the function `translate` to generate semantic representations of your source sentences. If your sentences can be parsed, `translate` should open another `gralej` window and with all of the translation results.

  ```
  | ?- translate([a,programmer,speaks,every,language]).
  ```

- You will be prompted as follows to see the next parse.

  ```
  ANOTHER? y
  ...
  ANOTHER? y
  no
  ```

  Answer `y` to see the next parse until you reach the end. Each time TRALE will open a new `Gralej` window. You need to store all of your translation results by repeating the previous step. A `no` will be returned when you reach the end of your parses.

- Save your translations of sentence (28) as `q2d_28_1.grale`, `q2d_28_2.grale` ...and your translations of sentence (30) as `q2d_30_1.grale`, `q2d_30_2.grale` ...

- Submit a zip file `q2d.zip` containing all the translation results. You can use this command: `zip -r q2d.zip q2d_*.grale` to create the zip file.

- Again, don't forget to **close all the windows** and kill your `Gralej` processes after you finish.

(1)

一个	程序员	每	种	语言	都	会说
yi ge	chengxuyuan	mei	zhong	yuyan	dou	huishuo
∃ *ge*-CL	programmer	∀	*zhong*-CL	language	*dou*	speak

A programmer every language speak.

We will ignore these.

(e) (4 marks) Large Language Models (LLMs) have gained quite a lot of popularity recently. In this question, you will explore the question of whether LLMs such as ChatGPT[4] "understand" quantifier scope.

There are several approaches to this exploration: You can ask ChatGPT to translate sentences and compare its translations to your grammar's translations. Or, you can directly interrogate ChatGPT about quantifier scope readings and analyse its responses. You can also design some clever linguistic tasks that involve quantifier scope and observe how ChatGPT handles them. Be both creative and precise in your experimentation.

In your write-up, report at least one case where GPT's behaviour differs from that of your grammar. Document the prompts you used and describe your experimental design. Reflect on the differences observed and share your thoughts on why these differences may have occurred.

Your analysis should be submitted as section 2(e) in `analysis.txt`.

[4] https://chat.openai.com/

CSC 485H/2501H, Fall 2023: Assignment 3

Family name: _____

Given name: _____

Student #: _____

Date: _____

I declare that this assignment, both my paper and electronic submissions, is my own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters and the Code of Student Conduct.

Signature: _____