Relative Clause Practice Program Report

Nora Alorainy, Yuan Chai[1]

12/11/2015

1. The authors are listed in an alphabetic order.

Abstract

English language learners have been reported to have difficulties acquiring relative clause constructions. And the choice of relative pronoun or adverb in the relative clause was among the difficulties they have encountered. In order to facilitate their understanding of relative clause, the researchers built a program enabling learners to fill out the missing relative pronouns in sentences and provided them with feedback and explanations. Through using the sentences in Brown Corpus, this program provided the users with authentic examples of relative clauses and the usage of relative pronoun in real life contexts. Through giving them feedback and explanations, this program offered the users a chance for self-review and self-learning. This paper would be a description of the program. The drawbacks of this program such as the limited size and a lack of alternative answers called for program updates and troubleshooting in the future.

Key words: Relative clause practice; relative pronoun; online feedback

Introduction

The grammatical construction of English relative clause has caused many confusions for English language learners. The definition of *relative clause* is "a clause that generally modifies a noun or noun phrase and is introduced by a relative pronoun or adverb" ("What Are Relative Clauses", n.d.). The difficulties in relative clause acquisition have been investigated in many studies on different subjects with several native tongues. In order to work out those difficulties, students should be exposed to relative clauses in real context. This need could be satisfied by the incorporation of corpus in example elicitation. This paper describes a program built by the researchers which serves as a relative clause test or practice. The program is built using python language, with the import of Natural Language Toolkit (NLTK), which is a library that contains many downloadable corpora and provides several classes and function for human language processing. In addition, this program employs SimpleGuitk, which is "a module is for building interactive programs and drawing" (Greiner, n.d.). The target users of this program are people who learn English as a second or foreign language. The objective is to offer them exercises regarding deciding the relativizers in relative clauses using authentic examples retrieved from Brown corpus (Francis & Kucera, 1979).

Literature Review

Extensive studies have suggested that it is difficult to acquire English relative clauses syntactically, semantically, or pragmatically for no matter native or non-native English speakers. According to Diessel and Tomasello (2005), relative clauses are characterized by "the head, which is the main clause element modified by the relative clause, and the gap, which is the gapped element in the relative clause" (p. 882). According to the variety of the gap elements, they summarized six variations of relative clauses:

i) S-relatives (relative clauses with an intransitive verb and a subject gap)

e.g. The girl that came with us.

ii) A-relatives (relative clauses with a transitive verb and a subject gap)
e.g. The man who saw the farmer.
iii) P-relatives (relative clauses with a direct object gap)
The cat that the dog chased.
iv) IO-relatives (relative clauses with an indirect object gap)
e.g. The girl who the boy give his ball to.
v) OBL-relatives (relative clauses with an oblique gap)
e.g. The boy who the girl played with.
vi) GEN-relatives (relative clauses with a genitive relative pronoun)
e.g. The man whose cat caught a mouse.

Most previous studies focus on people's comprehension ability on A and P-gaps. Many studies have declared that it is harder to process A-gap than to process P-gap (Diessel & Tomasello, 2005; Hsu, 2014; Sheldon, 1974). One possible rationale under those results has been hypothesized as the more interruptions between filler and gap, the more working memory it requires, and the more difficult the comprehension would be. However, the investigation of this hypothesis in syntactic domain is rather few. Chang (2004) is one of them. He once conducted an exam testing the usage of relative clause by Mandarin speakers. Several questions in that exam addressed the choice of relative pronouns in relative clauses. The test results show that the subjects have most difficulties in deciding the relative pronouns of IO, OBL, and GEN gapped relatives. Although the author may not have tested on the relative pronouns of all above six type relatives or made a comparison between them, he does have illustrate that language learners are likely to meet difficulties on choosing the relative pronouns of relative clauses. The degree of difficulty may increase as the distance between filler and gap increases. Based on such a background, this study aims to build a relative pronoun training program incorporating all the six aforementioned types of relative clauses. This program will on one hand provide English learner with more practice to help them work out the relative pronoun confusion. On the other hand, this program may function as a data elicitation tool for researchers who work on the filler-gap hypothesis from a syntactic perspective.

This program is going to be corpus-based. In the aforementioned studies, the materials are mostly created by the researchers. This might be good for illustration simplicity and variable manipulation in the research, however, the authenticity of the material is not guaranteed. According to Hanks (2009), using artificially created examples in dictionary may result in outdated syntactic structure, unusual lexical meaning and incomprehensive pragmatic usage. The same could be true in selecting materials for grammar exercises. This corpus-based exercise program will provide the language learners with relative clauses of various types and in authentic contexts, helping the learners understand the usage of relative pronouns in real-life context to the most extend.

Method

The program was built using Python 3.4. The programming was divided into three parts: 1) Grep relative sentences from corpus; 2) Import the sentences in files into three aligned lists: question, key, and explanation; 3) Construct an interactive interface for program-user interactions.

Grep Target Sentences

Step 1. The first step was to grep the relative clauses for tests. The corpus where the sentences were retrieved was Brown Corpus (Francis & Kucera, 1979). The retrieval was conducted within NLTK module. The sentences were retrieved by searching for a match of target relative pronouns and adverbs: “that”, “which”, “who”, “whom”, “whose”, “when”, “where”, “why”. In order to guarantee the target words function as the relative pronoun or adverb in the sentence, the programmers specified their part of speech (POS) as (“that”|”who”, “WPS”|”WPO”), (“whom”, “WPO”), (“which”, “WDT”), (“whose”, “WP$), (“when”|”where”|”why” “WRB”). For “when”, “where”, and “why”, further restrictions were needed because they function as Wh-pronouns in other types of clauses. The researchers limited the preceding words of “when” as “day(s), time”; “where” as “NN-TL” POS (place names); “why” as “reason(s)”. There were possibilities that those target words occur in other conditions, but for the ease of grepping in python, the researchers decided to limit the choices within such simple conditions.

Step 2. A grep function was built to extract the target sentences into files. Using “brown.tagged_sents()” method, the programmers searched for the tuples created in Step 1 within the tagged brown corpus. By using “if” conditions, the programmers set the program to find 15 matches for each tuple and store the matched sentences in separate lists. By using “for” loop, the items of each list (sentence strings) were written in separate files.

Step 3. In order to make the program more effective for language learning, the programmers wrote explanations for the choices of relative pronoun/adverb Because the explanations had to be sentence-specific, it were constructed by the programmers manually. That was a reason why the individual file size was restricted within 15. The rubric of the explanation was illustrated in Appendix 1 with a reference to an online source (“Relative Clauses”, n.d.). During the write-up of the explanation, the researchers left out several sentences that were not relative clauses or had inappropriate contents. There were 10 files and 128 sentences for both the original sentences and explanations in total after the screening. The sentence sequence between each sentence and explanation was aligned.

Import File into Lists

The objective of this part was to transfer the sentences into lists. Three lists were to be built: questions “list_joined_sentence”, keys “list_relative”, and explanations “list_exp”. The order of the items in those three lists had to be aligned. Those three empty lists were created at the beginning of this part.

Step 1. The questions were presented to the users as string items whose relative pronoun/adverb were replaced by a blank. The keys were the pronoun/adverb being omitted. In order to achieve this goal, the programmers built a “retrieve(x)” function. In order to refer to the files, the file names of sentences, questions and explanations were stored in three lists: list1, 2, 3. The variable of the retrieve function was the index of either list.

Step 1.1. Open the file by using the index variable to refer to the file name in list. Read the file by lines

Step 1.2. Find the relatives in each line (sentence) using "findall" method and assign the first item in the list as the key. Store the keys in "list_relative_each".

Step 1.3. Replace the relative in each sentence by a blank by splitting each sentence into a word list, locating the relative word using index, and replacing that by a "_____".

Step 1.4. Join the modified words into sentence: use "for" loop to join each sentence into a string and store it in "list_joined_each".

Step 1.5. Use "for" loop to append the sentence list and key list of each file into two big lists: "list_joined_sentence" and "list_relative".

Step 2. Append the lines of each explanation file into a big list: "list_exp" using "for" loop.

Build Interface

This part adopted an interface builder module – SimpleGuitk to realize program-user interaction. The interface consisted of a frame, buttons, input area, and a canvas. The programmers designed three major events: "button event", "draw event", and "input event". Within "button event", there would be "instruction", "new test", "next question", and "explanation" buttons on the interface. The "draw event" would draw the texts of instructions, questions, keys, feedback, explanations and scores according to the triggering of the input and different buttons. The performance of the events were defined by its corresponding functions. Such functions of events were called "handlers". The following sections would describe those handlers.

Step 1. Building Handler functions

"new_test" Handler. The objectives of this handler were to randomly pick 20 questions and also present the first question every time a new test starts. The keys and explanations had to match the questions as well. This was realized by randomly picking indexes from the indexes of "list_joined_sentence". "While" loop was used to pick the index for 20 times and store them in "list_chosen_index". Because "list_joined_sentence", "list_relative", and "list_exp" were aligned, referring to items by the same index would guarantee an alignment between question, key, and explanation. After the first item in those three lists were picked out, that index would be removed from "list_chosen_index".

"next_question" Handler. The objectives of this function were 1) to pick questions, keys, and explanation using the same method in "new_text" handler; 2) to tell when the test is finished by building an "if" condition. If the list_chosen_index becomes empty, the user would be notified with his score.

"explanation" Handler. The objective of this function was to indicate the presentation of explanation on canvas. The method would be described in the "difficulties and solution" section. The explanation button would be disabled if there is no input, which means the user has to input his answer before he sees the explanation.

"input" Handler. The objective was to allow the users to input their answers and get the feedbacks on canvas. Once the user inputs his answer at input area and hits

"enter", he would be notified the correctness of his answer. "If" conditions were used to check whether the input was equal to the key.

"draw Handler". The result of drawing event presented on canvas was illustrated in Table 1.

Table 1. The Draw Event Results of Button Performances

Button	Canvas
instruction	instruction text
new test	question 1
next question	next question
	at test end: end text + score
input + "enter"	feedback
explanation	explanation text

Step 2. Register the Handlers

The frame was created by the method "SimpleGui.create_frame". The buttons were registered by method "add_button("BUTTON NAME", HANDLER NAME, BUTTON WIDTH)". The input arear was registered by "add_input("INPUT AREA LABEL", INPUT HANDLER, INPUT AREA WIDTH)". The draw handler was registered by "set_draw_handler(draw)".

Launch the Program

The instruction function was operated once the program was launched. The frame was started by "frame.start".

Difficulties and Solutions

Grep Target Relative Clauses

Exclude non-relative clauses. In the stage of grepping sentences and writing the conditions, the first difficulty the programmers faced was how to grep the relative pronoun/adverb. When matching the target relativizers in the corpus, the researchers would retrieve all kinds of sentences containing the target words, such as adverbial clauses (I was listening to music when he came home). The solution was specifying the POS of each pronoun/adverb by looking into the tag menu using "nltk.help.upenn_tagset()" method.

Exclude non-target words. Initially, the researchers used regular expressions of "who|that|which" to search for sentences with a matched pattern. However, it would grep words part of which contains the target word (such as "whole" for "who"). In order to solve this problem, a space was added after the target word ("who ").

Import files into lists

Align Three Lists. The next difficulty was the alignment between question, its key, and its explanation. The programmers were hesitating between using dictionary and lists at first. As dictionary is not sequential, three lists were created to store questions, keys, and explanations separately.

Locate Target Words. The following question was how to get the indexes of the target relative words and replace them by a blank. In order to overcome this problem, the programmers separated each sentences into words using "<list>.split" method, and stored words of the each sentence into individual lists. The position of the relative word was then located in the corresponding list by its index and replaced by a blank.

By using ".join(<list>)" method, the list of words was joined into a string, with the relative pronoun/adverb replaced by a blank.

Shuffle the Lists. Problems also arose in shuffling the question list at the beginning of each new test. A shuffle method was firstly applied to the question list. However, the alignment among the three lists was consequently broken. In order to solve this problem, the programmers created a fourth list which contains all the indexes of the question list using the method of "len(list_joined_sentence)". Then, by shuffling the index list using "random.randint" method, 20 random index numbers would be picked at the beginning of each new test. The corresponding questions, keys, and explanations were simultaneously retrieved by index.

Construct the Interface

Import SimpleGuitk Module into Python. As SimpleGuitk module was not inherently installed in python program, it required to be downloaded and installed manually. The method of importing SimpleGuitk into python was detailedly described in READ ME file. The users could use that file as a reference.

Specify What Text to be Drawn. The program requires different texts to be drawn on canvas when different buttons are pressed. In order to achieve that, separate flag was built for each event handler. For example, the original value of "answer_on" flag was set "False". Once the input handler was triggered, the "answer_on" flag changed into "True" and the answer was drawn on canvas.

Return String when Exceeding Canvas Width. Several problems were found during the process of running the program. One was the length of the text may exceed the canvas width. As a result, the text would be cut. Regarding this problem, the function "return_string()" was built. The function contains a condition that will return a line when the sentence exceeds the canvas width. In order not to return the string in the middle of a word, the condition also requires the return position to meet word boundaries.

Count Score. Originally, the program would count the score every time the "Enter" key was hit. As a result, it would miscount the score by counting single input multiple times. To overcome this problem, flags were once again employed. Once the user hits "Enter", the "answer_on" flag would be assigned as "True". A condition was added to disable counting method when the "answer_on" flag was "True".

Ignore the Case of the Input. In order to be more user-friendly, the program decided to make the input of the answer case-insensitive. "answer.lower()" method was employed to achieve this goal. The character case of the input would thus be ignored.

Improvements and further modifications

Although many difficulties have been solved, this program is far from perfect. First, it is uneasy for users to upload their own examples to be tested in this program because the sentences are selected in a rather restricted condition. For example, the target relative pronoun/adverb has to be the first such word (e.g. For the sentence of "That girl that is pretty": the system will consider the first "that" as the key and the question would become: "____ girl that is pretty"). Further programs may work on this issue. Second, the size of the sentence pool is relatively small (128 sentences).

The reason is that explanations have to be written manually. In addition, even though POS of target words have been specified, the program would retrieve non-relative clauses sometimes (e. g. We were having a good time when Mary came home.), which means the programmer have to go over every sentence to screen such deviations from the question pool. Future studies may work on the refinement of grep program to improve the accuracy of sentence grepping. Third, this program does not offer alternative correct answers. For example, when a sentence grepped from Brown corpus has a relativizer "that", the program will consider this answer as the only correct one. But other grammatically correct answers, like "which", would be judged as incorrect. Future programmers may find a way to provide other possible correct answers in the feedback.

References

Francis, W. N., & Kucera, H. (1979). Brown corpus manual. *Letters to the Editor*, 5(2), 7.

Chang, Y. F. (2004). Second Language Relative Clause Acquisition: An Examination of Cross-Linguistic Influences. *Online Submission*.

Diessel, H., & Tomasello, M. (2005). A new look at the acquisition of relative clauses. *Language*, *81*(4), 882-906.

Greiner, J. (n. d.). SimpleGui Module – Frame. Retrieved on December 3, 2015, from http://www.codeskulptor.org/docs.html#Frames

Hanks, Patrick. "The impact of corpora on dictionaries." *Contemporary corpus linguistics* (2009): 214-236.

Hsu, C. C. (2014). The Role of Age in Mandarin-Speaking Children"s Performance of Relative Clauses. *Concentric: Studies in Linguistics*, *40*(2), 29-54.

Relative Clauses. (n. d.)https://www.ego4u.com/en/cram-up/grammar/relative-clauses

Sheldon, A. (1974). The role of parallel function in the acquisition of relative clauses in English. *Journal of verbal learning and verbal behavior*, *13*(3), 272-281.

What Are Relative Clauses in English? (n.d.). Retrieved on December 5, 2015, from http://grammar.about.com/od/rs/g/relativeclterm.htm

Appendix 1. Explanation Rubric

relative pronoun	use	example
who	The subject in the relative clause is missing. And that subject refers to “the woman”. We use “who” to refers to subjects for people.	I told you about the woman *who* lives next door.
which	The subject in the relative clause is missing. And that subject refers to “the cat”. We use “which” to refer to subjects for animals and things.	Do you see the cat *which* is lying on the roof?
which	The subject in the relative clause is missing. And that subject refers to the preceding sentence “He couldn”t read”. We use “which” to refer to subjects for things (events).	He couldn”t read *which* surprised me.
whose	The possessor of subject in the relative clause is missing. And that possessor refers to “the boy”. We use “whose” to refer to possession.	Do you know the boy *whose* mother is a nurse?
whom (modify the program)	The object in the relative clause is missing. And that object refers to “the professor”. We use “whom/who” to refer to object as people.	I was invited by the professor *whom* I met at the conference.
that	The subject in the relative clause is missing. And that subject refers to “the table”. We use “that” to refer to subjects for things.	I don”t like the table *that* stands in the kitchen.

relative adverb	meaning	use	example
when	in/on which	The time adverbial phrase “in the day” in relative clause is missing. We use “when” to refer to time expressions.	the day *when* we met him.
where	in/at which	The place adverbial phrase “at the place” in relative clause is missing. We use “where” to refer to place expressions.	the place *where* we met him
why	for which	The reason expression in relative clause is missing. We use “why” to refer to reason expressions.	the reason *why* we met him