International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org, editorijettcs@gmail.com
Volume 3, Issue 1, January – February 2014 ISSN 2278-6856

Design of A Web-Based System to Support CSA Instruction

Yukon Chang[1] and Chi-Chang Chen[2*]

[1,2]Information Engineering Department, I-Shou University,
Kaohsiung, Taiwan
*corresponding author

Abstract: ***English reading comprehension is recognized as a common problem for many non-English-speaking college students in science and technology. Cumulative Sentence Analysis, or CSA, was developed to help students master the skill of reading and understanding technical writings in English. Since each analysis step is rigorously defined, CSA is especially suitable for e-learning and a web-based system has been created to facilitate various activities in teaching and learning CSA. This paper presents the design of such a system.***

Keywords: *E-learning, Cumulative Sentence Analysis, E-education, Web-Based Systems*

1. INTRODUCTION

English is adopted as the standard language of communication in many disciplines in science and technology. Naturally, many textbooks and technical papers in these areas are written in English. More often than not, these writings employ long sentences with complicated structures because authors need to present complicated ideas with as much precision as possible. Many non-English-speaking college students in these fields of study, therefore, face the daunting task of overcoming the language hurdle so as to be able to read and understand textbooks and other technical writings correctly and efficiently.

The situation is not much different for college students in Taiwan. Even though they have six years of English in high school, where basic English grammar is not only taught but also made a big part of many tests, most of them still have difficulties applying grammar rules effectively in analyzing difficult long sentences. Failure to understand sentences in a textbook makes reading comprehension at higher levels, such as paragraphs and sections, even more difficult.

CSA, short for Cumulative Sentence Analysis [1], is our effort to help students deal with this problem. It is basically a method for analyzing English sentences in precisely defined steps. CSA has been included as part of the English for Science and Technology (EST) course at Information Engineering Department in I-Shou University, where most courses are still taught using English textbook. Since the primary goal of the EST course is to teach students how to read and understand textbooks in English, students are first taught CSA and then asked to apply CSA on selected sentences from the textbook of another course taught at the same time as EST. In the past few years, we have been using *Absolute C++* [2], which is the textbook used in Computer Programming. A separate study has shown that CSA is indeed effective in improving students' reading comprehension [3].

Since its inception, CSA has been accompanied by software that facilitates the creation of formatted analysis. This is out of necessity because CSA steps are so rigorous and the format of the resulting analysis so rigid that creating CSA results manually would be too time-consuming and error-prone to be viable. This software has gone through several versions over the past few years.

The purpose of this paper is to describe from a technical point of view the design of the web-based system that is currently used to support CSA instruction. However, in order to fully justify some of the design decisions, it is necessary to have a thorough understanding of the steps in CSA. Section II presents a rather detailed introduction for this purpose. In Section III, the design of the web-based system is described with screenshots illustrating functionalities provided by the system. Section IV discusses future works and concludes the paper.

2. 2. BRIEF DESCRIPTION OF CSA

It should be noted that the term CSA is used in this paper to refer to two different things. In the broader sense, the term refers to the method of incrementally accumulating words to restore a sentence to its original form. In the narrower sense, it refers to the end result one gets after applying the method to a particular sentence. The context in which the term appears usually provides enough information to distinguish between the two usages.

2.1 Main Idea behind CSA

The main idea behind CSA is simple -- within a group of words that form a phrase, clause, or sentence, some words are more important than others. Typically, the purpose of the less important words is to add details to the more important one. In grammatical terms, this is called modification. For example, the English grammar dictates that nouns can be modified by adjectives. Thus, in the phrase "a white dress", we say the adjective "white" modifies the noun "dress" because it provides more detail to the noun, namely, its color. The noun "dress" is

nevertheless the primary focus of the phrase. Similarly, a prepositional phrase such as "in a white dress" can be used to add detail to a noun "girl" and form the phrase "a girl in a white dress". In sentences with only one clause, called simple sentences by grammarians, the single pair of subject and main verb are universally taken to be the most important words. Other words are modifiers that can be added to the subject-verb pair in some suitable sequence to make the sentence complete. For sentences with more than one clause, the clauses are separated first and each clause is dealt with individually in the same manner.

It is an experience common to many English learners that long sentences with complicated structures are the ones that give them the most trouble. By removing the modifiers and focusing only on the modified, we make a sentence shorter while retaining the core meaning of the sentence. Since shorter sentences are easier to comprehend than longer ones, we make initial understanding of the core meaning much easier. The parts that are left out are subsequently added back in chunks incrementally to restore the original meaning. If the restoration process is done in a suitable sequence, the reader is given a chance to view the sentence in various levels of details, which may also contribute to improving his or her reading comprehension at the sentence level.

2.2 Steps in CSA

CSA consists of six steps, as illustrated in Fig. 1. Description of each step is given below.

Step 1. Identify finite verbs. A finite verb is a verb in one of three forms: the un-conjugated base form, the third person singular form, usually with -s suffix, and the past tense. The English grammar requires each clause to have one finite verb unless it is omitted. Other verb forms, namely the present participle, the past participle, and the infinitive, are called nonfinite. In this step, all finite verbs in a given sentence are identified. In most cases, the number of finite verbs is also the number of clauses in the sentence.

Although not directly related to cumulating chunks of words to restore a sentence, identifying finite verbs is nonetheless an important preparatory step in CSA. In addition to providing a reliable means to correctly determine the number of clauses in a given sentence to facilitate their separation, it also helps identify subject-verb pairs because finite verbs identified here will always be part of S+V pair.

Step 2. Find keywords. Keywords are those words that appear at the beginning of a dependent clause in a complex sentence or an additional independent clause in a compound sentence. This step is included in CSA to make the next step easier because (1) a sentence with n clauses normally has n-1 keywords, and (2) keywords often reveal the locations where one clause ends and another starts. Furthermore, the type of clauses can sometimes be determined by the keywords. For instance, a clause led by a subordinate conjunction is almost always an adverbial clause. Correctly identifying the type of a clause is important when trying to discover the meaning of a long, complicated sentence.

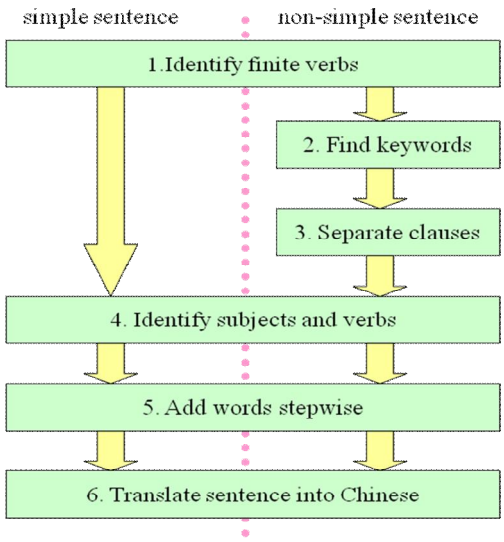

Figure 1 Six steps of CSA

Step 3. Separate clauses. Depending on the type of clauses, three different actions are used to separate them. They are bisection, extraction, and substitution. Bisection is used to separate either two independent clauses or a dependent and an independent clause by simply making a cut at the point the two clauses meet. Extraction is used to temporarily take a relative clause out of the sentence. Substitution works by first taking away a noun clause and then in its place put in a generic noun, such as something, as a placeholder. These three actions are shown in Fig. 2.

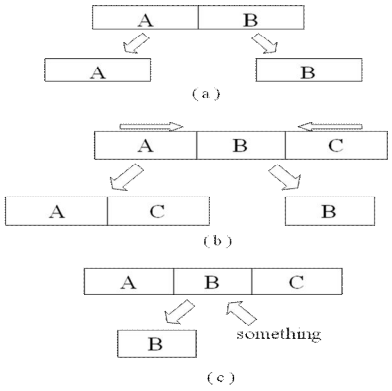

Figure 2 Actions to separate clauses: (a) bisection (b) extraction (c) substitution

As mentioned earlier, different keywords are related to different types of clauses, which in turn require different separation actions. Table 1 gives a short list of keywords

along with their corresponding actions. Since simple sentences have only one clause, steps 2 and 3 can be skipped completely, as can be seen in Fig. 1.

Table 1: Keywords and corresponding actions

Action	Keywords
bisection	coordinating conjunctions: and, but, or, for, yet, nor, so subordinate conjunctions: if, because, although, since, while, until, unless, so that, before, after, as if, rather than, as long as, etc.
extraction	relative pronouns: who, whose, whom, which, that
substitution	that, where, when, what, how, why

Step 4. Identify the subject and main verb pairs. This step is to find out the subject and main verb for each clause. The subject should be as concise as possible. The verb part should include auxiliary verbs, all words that are used to form the complete tense and voice, and any other words that go with the main verb to form an idiom. This rather stringent rule is intend to make the core meaning concise on one hand while adherent to the original meaning on the other. The subject-verb pair become the first words selected in the first pass in step 5.

Step 5. Add words stepwise. This is where the major work of CSA resides. This step includes a number of passes where one or more words are added in each pass to incrementally restore the original sentence. For non-simple sentences, the same process is applied to each clause in the sentence.

CSA is extremely flexible in this step. Unlike previous steps where a correct answer can be uniquely specified, there is no single standard answer for step 5. Even though a particular sequence has to be chosen when presenting the analysis of a sentence, any sequence in which (1) the chunking of words is acceptable and (2) modifiers are added later than or at the same time as the words they modify will have to be considered correct. Granularity of the analysis can be controlled by adding more or fewer words in each pass. An additional goal of CSA is that following the rules above will make most partially filled sentences in the sequence not only grammatically correct but also consistent with the original sentence semantically.

Step 6. Translate the sentence into Chinese. In this step, Chinese translation is provided for the resulting partially filled sentence in each pass of step 5. Once translation for each clause is given, the translation of the entire sentence can be finalized.

An example of all six steps in CSA is shown in Fig. 3. Fonts in the figure are intentionally made large enough so that the content is easily readable. The figure is actually the screenshot of the Viewer, to be discussed in Section III. It displays the analysis of the sentence "Call-by-value parameters are local variables just like the variables you declare within the body of a function". Colors are used in each step to highlight selected words, individual clauses, and words added in each pass. The screenshot shows that the finite verbs are "are" and "declare"; the keyword is an omitted and restored "that"; one clause is shown in red and the other in blue; and "parameters are" and "you declare" are the subject-verb pairs. The sequence in step 5 and translation in step 6 should be self-explanatory.

3. DESIGN OF THE SYSTEM

3.1 System Overview

The current system supporting CSA instruction has a classical client-server architecture. The server is a PC running FreeBSD [4] with Apache [5] installed as the web server. Server-side programs are coded in PHP [6] and persistent data is stored in MySQL [7] database at the server. Any web browser conforming to W3C standards can be used as the client. Client-side programming is done mainly in JavaScript [8] with occasional use of other frameworks to gain special functionality. For instance, the CSA Animator uses YUI library [9] to provide animation.

To make the system as easy to use as possible, user interface at the client side is mostly done with simple mouse clicking except when text input is required. In steps 1, 2, 4, and 5, clicking a word identifies the word as a finite verb, a keyword, a word belonging to a subject-verb pair, and a word to be added in the next pass, respectively. In step 3, different colors are used to represent different clauses and there is always a current color. Clicking a word colors the word with the current color. Clicking a word while pressing down the shift key colors the clicked word as well as the entire block of uncolored words directly before it.

It should be pointed out that the purpose of this system is to facilitate the creation, management, and application of CSA for a pool of sentences. This system has never been intended to be a full-fledged course management system. For instance, lecture notes and slides are not the responsibility of the system. Community interaction is also not supported by the system.

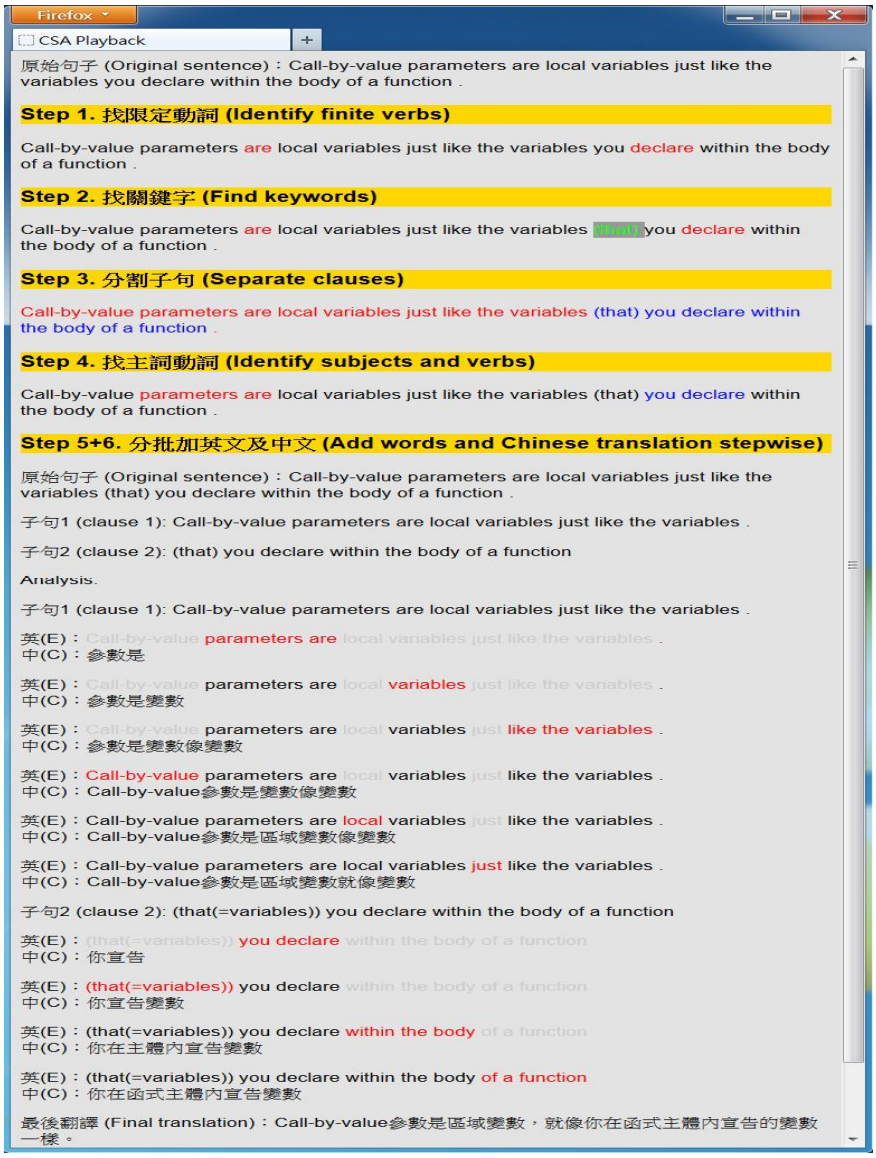

Figure 3 CSA of a sentence, displayed through the Viewer

3.2 Users of the System

Users of the system can be categorized into four groups based on the roles they play. Arranged in decreasing privilege levels, they are authors, instructors, students, and visitors. The role of each group is described below.

Authors are assigned the responsibility to select sentences, create CSA for them, and commit the results to the sentence pool in the database. The tool for this task is the CSA Editor.

An instructor of the system is allowed to create classes, add new or existing users to a class, create exams and retrieve exam results. It is also the responsibility of the instructor to manually grade step 6 because step 6 cannot be automatically graded.

A student in a class can take exams administered by the instructor and practice CSA from problem banks

All aforementioned users must be authenticated before using the system. A person without an account can still use the system as a visitor. A visitor can view the analysis of a sentence using the Viewer and try the Freeform Editor as an anonymous user.

A user in a group with higher privilege can do everything that is permitted to a user in a group with lower privilege. For instance, a student can also use the Freeform Editor to practice CSA on sentences not in a problem bank.

The user management subsystem shares the same code and user information with that in Personal WorkBank, another web-based system created by the author [10]. Because both systems are used by registered students in

EST courses, consolidating users means one fewer password to remember for the users and more coherent user management.

3.3 Data Structure for CSA

Central to the system design is the choice of a suitable data structure to store the result from each step of CSA. Since encoding steps 1-4 and 6 is more or less trivial, the main focus is on selecting a data structure to represent the sequence in which words are cumulated in step 5. Adding to the complexity of the selection is that recording the sequence alone is not enough, because that particular sequence is only one of many acceptable ways to build up the sentence. As described in Section II, any sequence that satisfies the condition that the modified words are added before or at the same time as the modifier words must also be considered as a correct sequence.

This requirement can be satisfied by using a tree in which each node represents a chunk of words added in the same pass and an edge from a child to its parent represents words in the child must be added no earlier than the words in the parent. We call such a tree a pass tree. Using the example shown in Fig. 3, the pass tree for the first clause is shown in Fig. 4.

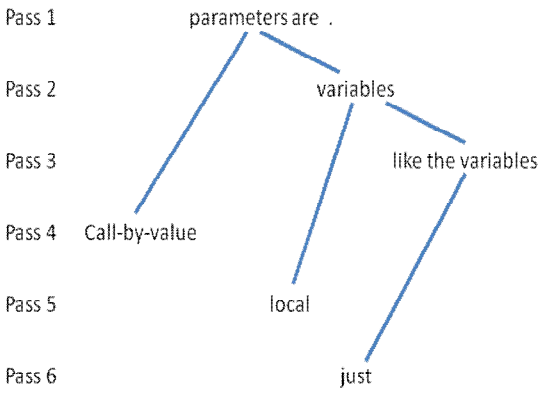

Figure 4 Pass tree specifying precedence for adding words

Two most frequently used formats in the web era are XML [11] and JSON [12]. Realizing that CSA requires only array and key-value pairs to represent its steps, we pick JSON because it is "fat-free" [13], flexible, and comes with excellent support in both JavaScript and PHP.

The pass tree in Fig. 4 is encoded as a JSON array [-1,-1,1,2,1,2,3]. Since JSON arrays start with index 0 and pass 1 must precede all other passes, the first two entries are filled with a nonexistent pass number, -1. Other entries point up to their respective parents. Table 2 lists major key-value pairs in JSON representation of CSA.

Once this data structure is determined, programs can be written to create, modify, and access this data structure to provide services to various users. Almost all components in the system make use of this data structure in certain ways. Some of the more important components are presented in the following subsections.

Table 2: JSON representation of CSA

Step	Key	Value and Interpretation
1	fv	fv = 1 if the word is a finite verb
2	kw	kw = 1 if the word is a keyword kw = 2 for an omitted keyword, usually "that"
3	cl	cl = *i* if the word is in clause *i*
4	sv	sv = *i* if the word is part of S+V pair in clause *i*
5	pa passtree	pa = *i* if the word is added in pass *i*an array specifies partial ordering among passes
6	trClauses trAll	Chinese translation for each pass in a clause Chinese translation for the entire sentence

3.4 CSA Editor

The primary goal of the Editor is to create CSA for a given sentence. The Editor first breaks up the sentence into tokens and then guides the author through the steps prescribed by CSA. The result of tokenization can be manually corrected if necessary. All processing performed by the Editor is carried out at the client side without server interaction. This is accomplished by manipulating style sheets and DOM objects through JavaScript. Only when the final JSON is produced does the Editor send it to the server and terminates. An example of the Editor half way through step 3 is presented in Fig. 5.

The Editor can also be used to modify an existing CSA. For example, when an author creates CSA for all sentences in a paragraph, it is sometimes more efficient to first finish steps 1-5 for each sentence. Adding translation, which requires a large amount of typing and cut-and-paste, can be done at a later time.

The scenario mentioned in the previous paragraph is actually not uncommon. Experience has shown that having an entire paragraph or page analyzed in CSA works better on improving comprehension. Students are presented with a complete paragraph where every sentence can be clicked to reveal its CSA. This gives a sense of context, which is nonexistent in case of isolated sample sentences. To facilitate this common task, the Editor is accompanied by a front end called Sentencizer, whose job is to divide a paragraph into sentences and to handle proper indexing of each sentence. Since technical writing sometimes contains

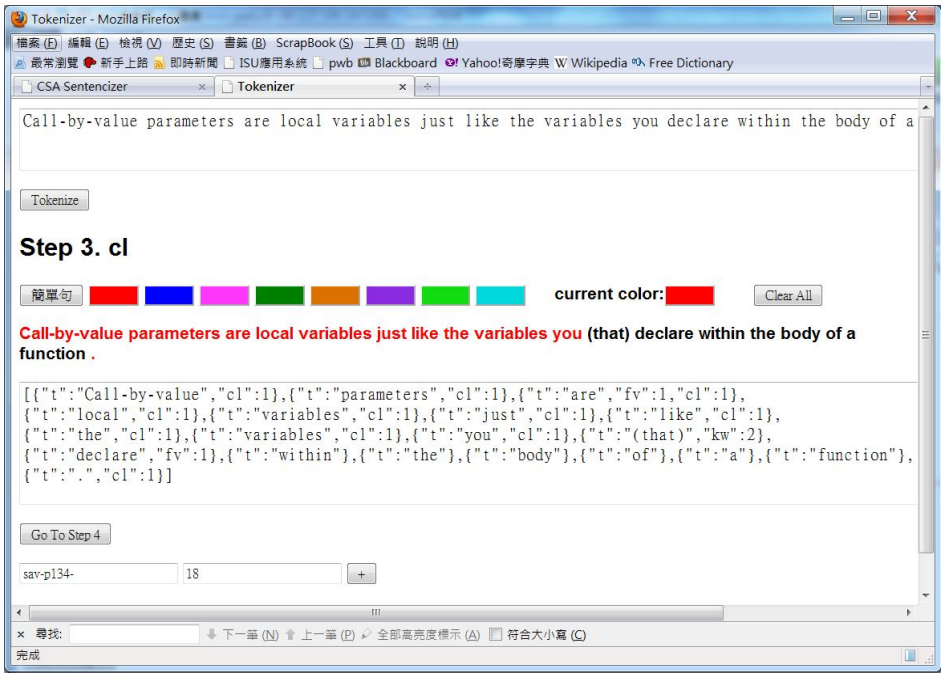

Figure 5 CSA Editor in step 3

notations, equations, or code segments that may be difficult to break up automatically, a manual override mechanism is provided to the Sentencizer.

After the CSA for a sentence is created and entered into database, other activities based on this sentence may start.

3.5 Viewer and Animator

The task of the Viewer is simply to render the CSA of a sentence as a static web page. The processing is rather straightforward. The JSON representation of the sentence is retrieved from the database and converted into a JavaScript record with the built-in function json_decode( ). Relevant information is extracted in each step to produce the output. An example of the Viewer output has already be given in Fig. 3.

The beauty of having a set of well-defined steps and a large sentence pool is that once a program is written to provide some sort of sophisticated presentation or processing, that capability is readily available for every sentence in the sentence pool automatically. As an example of CSA's versatility, we designed a client side program, called Animator, which can present the sequence of building up a sentence in an animated web page as long as CSA of that sentence is available.

3.6 Tester and Grader

No instructional system, computerized or otherwise, is complete without the capability to assess the progress made by the students. The Tester in our system provides just that capability. The Tester comes in two versions, one that provides immediate feedback to the student and one that does not. The former is well suited for practice purposes. A student can go through the sentences in a test bank using the program and find out if his understanding of CSA is valid. The latter is intended for formal test prepared by the instructor. Such a test usually consists of multiple sentences and a student must finish them all without getting any feedback while the test is still going on. Because the six steps of CSA are usually taught in order, each test has an argument specifying the number of steps a student should do. This feature is handy because by eliminating step 6 from the test, the entire test can be graded automatically

In either version, the Tester works in a way similar to the way Editor does. The student taking the test mostly clicks through the steps unless text input is required. After a sentence is finished and the answer submitted, the Grader will be activated to determine the correctness for steps 1-5. If feedback is to be provided, the Grader lists the answer submitted by the student and the standard answer side by side along with the verdict of whether a step is done correctly. Fig. 6 shows the output of the Grader for the sentence in Fig. 3. The test stops at step 5.

Note that although the step 5 answer provided by the student is different from the standard answer, it is nevertheless marked correct because no modified words are added after the modifiers.

The system provides yet a third form of Tester, which we call Freeform Editor. This is where a user can select any sentence and perform a CSA on the sentence. Naturally, no grading can be done in this case, but the CSA created by the user will be stored in the database so that it can be retrieved later for discussion or assessment.

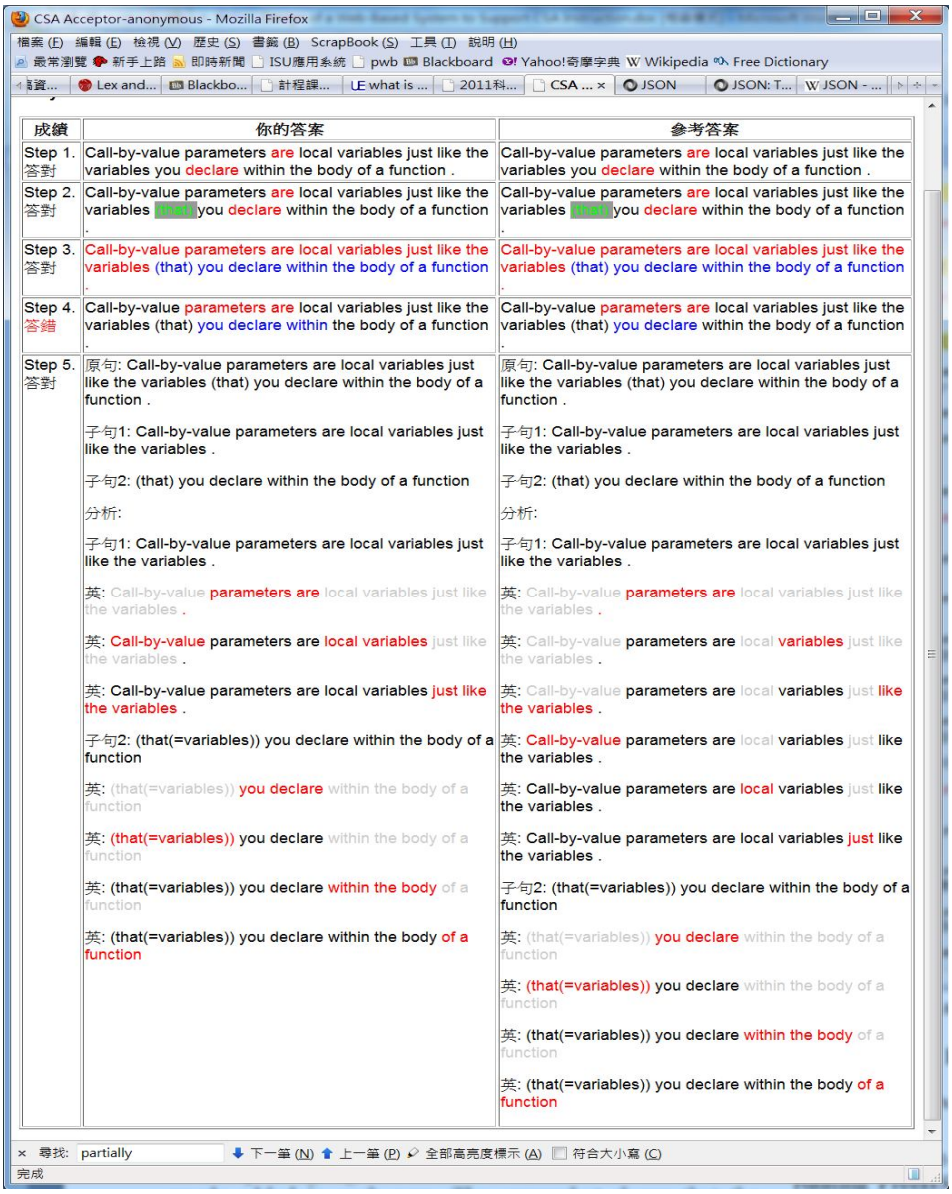

Figure 6 Grader output with correct/incorrect verdict

4. CONCLUSIONS

CSA was originally designed to be self-explanatory -- once a user gets acquainted with the output format, he or she can read the CSA of any sentence without further explanation. It has been used in real-world English for Science and Technology courses and its effectiveness has been established.

One of the most unique features of CSA is the well-defined steps, through which the mental process of comprehending a sentence is in some way made visible. This feature creates many possibilities for further studies to understand the English learning process. In a separate study, for example, error patterns among CSA steps are identified and analyzed using data mining [14]. Studies such as this may lead to further improvement of CSA.

References

[1] Y. Chang, English for Science and Technology, 2nd Ed., New Wun Ching Publishing, 2008. (in Chinese)

[2] W. Savitch, Absolute C++, 5th Ed., Boston: Pearson/Addison-Wesley, 2013.

[3] Y.-R. Tsai, Y. Chang, & Y.-H. Shiu, Supporting EFL learners' reading comprehension through an on-line Cumulative Sentence Analysis (CSA) strategy instruction, British Journal of Educational Technology, submitted for publication.

[4] The FreeBSD Project, http://www.freebsd.org/

[5] The Apache Software Foundation, http://www.apache.org/
[6] PHP, http://www.php.net/
[7] MySQL, http://www.mysql.com/
[8] M. Haverbeke, Eloquent JavaScript, A Modern Introduction to Programming, No Starch Press, CA USA, 2011.
[9] YUI Library, http://developer.yahoo.com/yui/
[10]M. Lin, Y. Chang, W. Ku, W. Chen, The study of Backword in personal wordbank, Proceedings of Information Technologies, Applications and Management Conference, Kaohsiung, Taiwan, 2011.
[11]XML, http://www.w3.org/XML/
[12]JSON, http://json.org/
[13]D. Crockford, JSON: The Fat-Free Alternative to XML, International World Wide Web Conference, Edinburgh, Scotland, 2006.
[14]Y.-R. Tsai, Y. Chang, C.-S. Ouyang, Mining Error Patterns of Engineering Studetns' English Reading Comprehension, 2011 International Conference on Machine Learning and Cybernetics, pp.68-72, Guilin, Guangxi, China, 2011.

AUTHORS

Yukon Chang received the Ph.D. degree in computer science from Pennsylvania State University, University Park, in 1986. He was an assistant professor of computer science with the State University of New York at Albany from 1986 to 1992. In 1992, he joined the Department of Information Engineering, I-Shou University, Kaohsiung, Taiwan, R.O.C., where he is now an associate professor. He also served as Director of Computer Center from 1993 to 1996, Director of the Library at I-Shou University from 1998 to 2001, and Department Head from 2002 to 2005. His primary research interest is in multimedia networks, information network, software engineering, and digital Learning

Chi-Chang Chen (corresponding author) received the BS degree in computer science from Shochow University, Taiwan, in 1984, and the MS degree in information engineering from Tatung University, Taiwan, in 1986. He received the PhD degree in computer science from the Texas A&M University in 1995. He is currently an associated professor in information engineering department, I-Shou University, Kaohsiung, Taiwan. His research interests include wireless sensor networks, cluster computing, cloud computing, and digital Learning.