A FLEXIBLE EXAMPLE-BASED PARSER BASED ON THE SSTC *

Mosleh Hmoud Al-Adhaileh & Tang Enya Kong
Computer Aided Translation Unit
School of computer sciences
University Sains Malaysia
11800 PENANG, MALAYSIA
mosleh@cs.usm.my, enyakong@cs.usm.my

Abstract

In this paper we sketch an approach for Natural Language parsing. Our approach is an example-based approach, which relies mainly on examples that already parsed to their representation structure, and on the knowledge that we can get from these examples the required information to parse a new input sentence. In our approach, examples are annotated with the Structured String Tree Correspondence (SSTC) annotation schema where each SSTC describes a sentence, a representation tree as well as the correspondence between substrings in the sentence and subtrees in the representation tree. In the process of parsing, we first try to build subtrees for phrases in the input sentence which have been successfully found in the example-base - a bottom up approach. These subtrees will then be combined together to form a single rooted representation tree based on an example with similar representation structure - a top down approach.

Keywords: Example-based parsing, SSTC .

1. INTRODUCTION

In natural language processing (NLP), one key problem is how to design an effective parsing system. Natural language parsing is the process of analyzing or parsing that takes sentences in a natural language and converts them to some representation form suitable for further interpretation towards some applications might be required, for example, translation, text abstraction, question-answering, etc. The generated representation tree structure can be a phrase structure tree, a dependency tree or a logical structure tree, as required by the application involved.

Here we design an approach for parsing natural language to its representation structure, which depends on related examples already parsed in the example-base. This approach is called example-based parsing, as oppose to the traditional approaches of natural language parsing which normally are based on rewriting rules. Here linguistic knowledge extracted directly from the example-base will be used to parse a natural language sentence (i.e. using past language experiences instead of rules). For a new sentence, to build its analysis (i.e. representation structure tree), ideally if the sentence is already in the example-base, its analysis is found there too, but in general, the input sentence will not be found in the example-base. In such case, a method is used to retrieve close related examples and use the knowledge from these examples to build the analysis for the input sentence. In general, this approach relies on the assumption that if two strings (phrase or sentence) are "close", their analysis should be "close" too. If the analysis of the first one is known, the analysis of the other can be obtained by making some modifications in the analysis of the first one.

The example-based approach has become a common technique for NLP applications, especially in MT as reported in [1], [2] or [3]. However, a main problem normally arises in the current approaches which indirectly limits their applications in the development of a large scale and practical example-based system. Namely the lack of flexibility in creating the representation tree due to the restriction that correspondences between nodes (terminal or non terminal) of the representation tree and words of the sentence must be one-to-one and some even restrict it to only in projective manner according to certain traversal order. This restriction normally results to the inefficient usage of the example-base. In this paper, we shall first discuss on certain cases where projective representation trees are inadequate for characterizing representation structures of some natural linguistic phenomena, i.e. featurisation, lexicalisation and crossed dependencies. Next, we

* The work reported in this paper is supported by the IRPA research programs, under project number 04-02-05-6001 funded by the Ministry of Science, Technology and Environment, Malaysia.

propose to overcome the problem by introducing a flexible annotation schema called Structured String-Tree Correspondence(SSTC) which describes a sentence, a representation tree, and the correspondence between substrings in the sentence and subtrees in the representation tree. Finally, we present a algorithm to parse natural language sentences based on the SSTC annotation schema.

2. NON-PROJECTIVE CORRESPONDE -NCES IN NATURAL LANGUAGE SENTENCES

In this section, we shall present some cases where projective representation tree is found to be inadequate for characterizing representation tree of some natural language sentences. The cases illustrated here are featurisation, lexicalisation and crossed dependencies. An example containing mixture of these non-projective correspondences also will be presented.

2.1 Featurisation

Featurisation occurs when a linguist decides that a particular substring in the sentence, should not be represented as a subtree in the representation tree but perhaps as a collection of features. For example, as illustrated in figure 1, this would be the case for prepositions in arguments which can be interpreted as part of the predicate and not the argument, and should be featurised into the predicate (e.g. "up" in "picks-up"), the particle "up" is featurised as a part of the feature properties of the verb "pick".

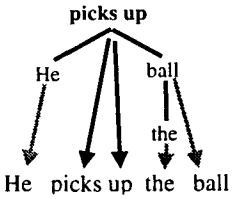


Figure 1: Featurisation

2.2 Lexicalisation

Lexicalisation is the case when a particular subtree in the representation tree presents the meaning of some part of the string, which is not orally realized in phonological form. Lexicalisation may result from the correspondence of a subtree in the tree to an empty substring in the sentence, or substring in the sentence to more than one subtree in the tree. Figure 2 illustrates the sentence "John eats the apple and Mary the pear" where "eats" in the sentence corresponds to more than one node in the tree.

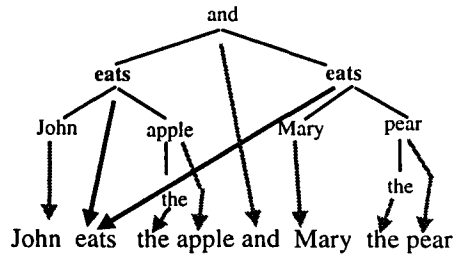


Figure 2: Lexicalisation

2.3 Crossed dependencies

The most complicated case of string-tree correspondence is when dependencies are intertwined with each other. It is a very common phenomenon in natural language. In crossed dependencies, subtree in the tree corresponds to single substring in the sentence, but the words in a substring are distributed over the whole sentence in a discontinuous manner, in relation to the subtree they correspond to. An example of crossed dependencies is occurred in the sentences of the form $(a^n v b^n c^n \mid n>0)$, figure 3 illustrates the representation tree for the string "**aa v bb cc** " (also written **a.1a.2 v b.1b.2 c.1c.2** to show the positions), this akin to the 'respectively' problem in English sentence like "John and Mary give Paul and Ann trousers and dresses respectively" [4].

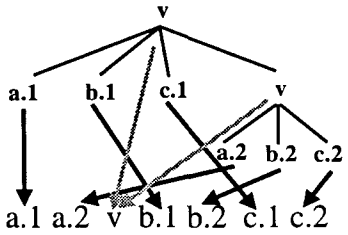


Figure 3: Crossed dependencies

Sometimes the sentence contains mixture of these non-projective correspondences, figure 4 illustrates the sentence "He picks the ball up", which contains both featurisation and crossed dependencies. Here, the particle "up" is separated from its verb "picks" by a noun phrase "the ball" in the string. And "up" is featurised into the verb "picks" (e.g. "up" in "picks-up").

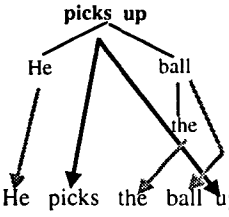


Figure 4: Mixture of featurisation and crossed dependencies

3. STRUCTURED STRING-TREE CORRESPONDENCE (SSTC)

The correspondence between the string on one hand, and its representation of meaning on the other hand, is defined in terms of finer subcorrespondences between substrings of the sentence and subtrees of the tree. Such correspondence is made of two interrelated correspondences, one between nodes and substrings, and the other between subtrees and substrings, (the substrings being possibly discontinuous in both cases).

The notation used in SSTC to denote a correspondence consists of a pair of intervals X/Y attached to each node in the tree, where X(SNODE) denotes the interval containing the substring that corresponds to the node, and Y(STREE) denotes the interval containing the substring that corresponds to the subtree having the node as root [4].

Figure 5 illustrates the sentence "all cats eat mice" with its corresponding SSTC. It is a simple projective correspondence. An interval is assigned to each word in the sentence, i.e. (0-1) for "all", (1-2) for "cats", (2-3) for "eat" and (3-4) for "mice". A substring in the sentence that corresponds to a node in the representation tree is denoted by assigning the interval of the substring to SNODE of the node, e.g. the node "cats" with SNODE interval (1-2) corresponds to the word "cats" in the string with the similar interval. The correspondence between subtrees and substrings are denoted by the interval assigned to the STREE of each node e.g. the subtree rooted at node "eat" with STREE interval (0-4) corresponds to the whole sentence "all cats eat mice".

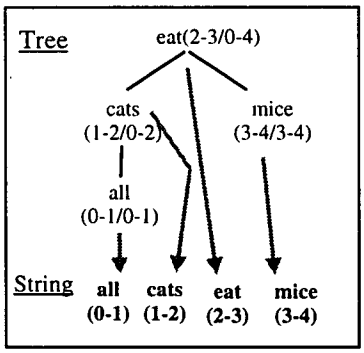


Figure 5: An SSTC recording the sentence "all cats eat mice" and its Dependency tree together with the correspondences between substrings of the sentence and subtrees of the tree.

4. USES OF SSTC ANNOTATION IN EXAMPLE-BASED PARSING

In order to enhance the quality of example-based systems, sentences in the example-base are normally annotated with theirs constituency or dependency structures which in turn allow example-based parsing to be established at the structural level. To facilitate such structural annotation, here we annotate the examples based on the Structured String-Tree Correspondence (SSTC). The SSTC is a general structure that can associate, to string in a language, arbitrary tree structure as desired by the annotator to be the interpretation structure of the string, and more importantly is the facility to specify the correspondence between the string and the associated tree which can be interpreted for both analysis and synthesis in NLP. These features are very much desired in the design of an annotation scheme, in particular for the treatment of linguistic phenomena which are not-standard e.g. crossed dependencies [5].

Since the example in the example-base are described in terms of SSTC, which consists of a sentence (the text), a dependency tree[1] (the linguistic representation) and the mapping between the two (correspondence); example-based parsing is performed by giving a new input sentence, followed by getting the related examples(i.e. examples that contains same words in the input sentence) from the example-base, and used them to compute the representation tree for the input sentence guided by the correspondence between the string and the tree as discussed in the following sections. Figure 6 illustrates the general schema for example-based NL parsing based on the SSTC schema.

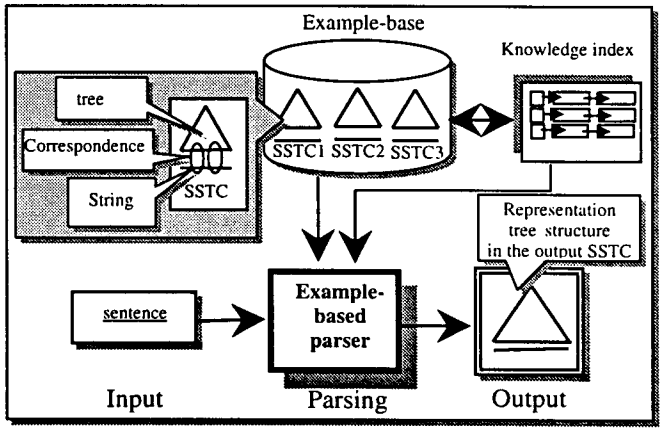


Figure 6: Example-based natural language parsing based on the SSTC schema.

4. 1 The parsing algorithm

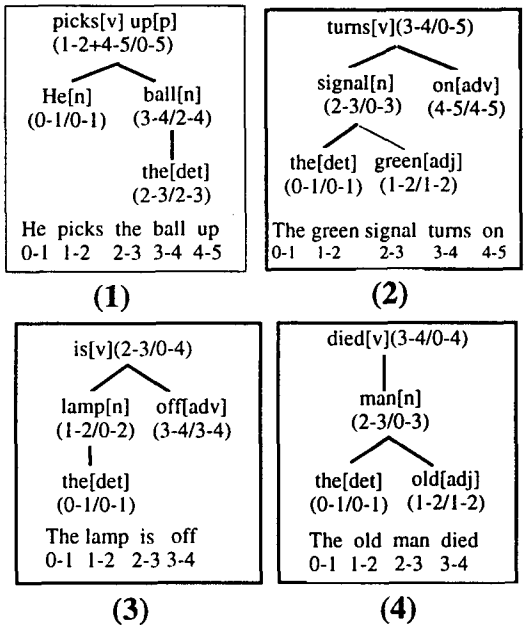
The example-based approach in MT [1], [2] or [3], relies on the assumption that if two sentences are "close", their analysis should be "close" too. If the analysis of the first one is known, the analysis of the other can be obtained by making some modifications in the analysis of the first one (i.e.

[1] Each node is tagged with syntactic category to enable substitution at category level.

close: distance not too large, modification: edit operations (insert, delete, replace) [6].

In most of the cases, similar sentence might not occurred in the example-base, so the system utilized some close related examples to the given input sentence (i.e. similar structure to the input sentence or contain some words in the input sentence). For that it is necessary to construct several subSSTCs (called substitutions hereafter) for phrases in the input sentence according to their occurrence in the examples from the example-base. These substitutions are then combined together to form a complete SSTC as the output.

Suppose the system intends to parse the sentence ***" the old man picks the green lamp up"***, depending on the following set of examples representing the example-base.



The example-base is first processed to retrieve some knowledge related to each word in the example-base to form a knowledge index. Figure 7 shows the knowledge index constructed based on the example-base given above. The knowledge retrieved for each word consists of:

1. **Example number**: The example number of one of the examples which containing this word with this knowledge. Note that each example in the example-base is assigned with a number as its identifier.
2. **Frequency**: The frequency of occurrence in the example-base for this word with the similar knowledge.
3. **Category**: Syntactic category of this word.
4. **Type**: Type of this word in the dependency tree (0: terminal, 1: non-terminal).
   - *Terminal word*: The word which is at the bottom level of the tree structure, namely the word without any son/s under it (i.e. STREE=SNODE in SSTC annotation).
   - *Non terminal word*: The word which is linked to other word/s at the lower level, namely the word that has son/s (i.e. STREE≠SNODE in SSTC annotation).
5. **Status**: Status of this word in the dependency tree (0: root word, 1: non-root word, 2: friend word)
   - *Friend word*: In case of featurisation, if a word is featurised into other word, this word is called friend for that word, e.g. the word "up" is a friend for the word "picks" in figure 1.
6. **Parent category**: Syntactic category of the parent node of this word in the dependency tree.
7. **Position**: The position of the parent node in the sentence (0: after this word, 1: before this word).
8. **Next knowledge**: A pointer pointing to the next possible knowledge of this word. Note that a word might have more than one knowledge, e.g. "man" could be a verb or a noun.

Based on the constructed knowledge index in figure 7, the system built the following table of knowledge for the input sentence:

The input sentence: ***the old man picks the green lamp up***

				0-1	***1-2***	***2-3***	***3-4***	***4-5***	***5-6***	***6-7***	***7-8***
the	0	1	→	1	4	det	0	1	n	0	nil
old	1	2	→	4	1	adj	0	1	n	0	nil
man	2	3	→	4	1	n	1	1	v	0	nil
picks	3	4	→	1	1	v	1	0	-	-	nil
the	4	5	→	1	4	det	0	1	n	0	nil
green	5	6	→	2	1	adj	0	1	v	0	nil
lamp	6	7	→	3	1	n	1	1	v	0	nil
up	7	8	→	1	1	p	1	2	v	1	nil

Note that to each word in the input sentence, the system built a record which contain the word, SNODE interval, and a linked list of possible knowledge related to the word as recorded in the knowledge index. The following figure describes an example record for the word **<the>**:

This mean:
the word <the>, snode(0-1), one of the examples that contain the word with this knowledge is example 1, this knowledge repeated 4 time in the example-base, the category of the word is <det>, it is a terminal node, non-root node, the parent category is <n>, and the parent appear after it in the sentence.

the	0	1	→	1	4	det	0	1	n	0	nil

word	Example No.	frequency	category	type	status	Parent category	Position	Next Kn.
the	1	4	det	0	1	n	0	nil.
old	4	1	adj	0	1	n	0	nil.
he	1	1	n	0	1	v	0	nil.
turns	2	1	v	1	0	-	-	nil.
ball	1	1	n	1	1	v	1	nil.
green	2	1	adj	0	1	n	0	nil.
signal	2	1	n	1	1	v	0	nil.
on	2	1	adv	0	1	v	1	nil.
picks	1	1	v	1	0	-	-	nil.
off	3	1	adv	0	1	v	1	nil.
man	4	1	n	1	1	v	0	nil.
died	4	1	v	1	0	-	-	nil.
lamp	3	1	n	1	1	v	0	nil.
up	1	1	p	1	2	v	1	nil.

Figure 7: The knowledge index for the words in the example-base.

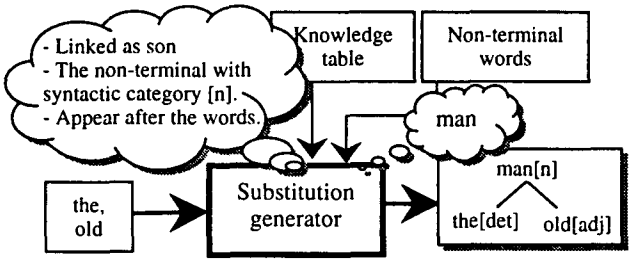
This knowledge will be used to build the substitutionsfor the input sentence, as we will discuss in the next section.

4.1.1 Substitutions generation

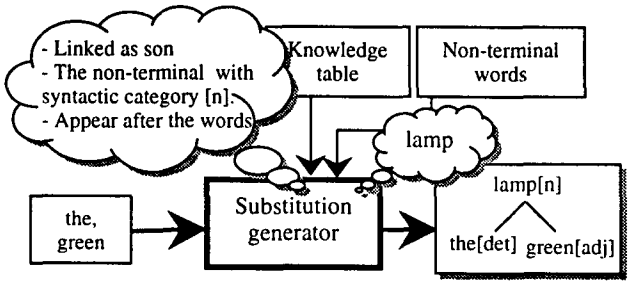
In order to build substitutions, the system first classifies the words in the input sentence into terminal words and non-terminal words. For each terminal word, the system tries to identify the non-terminal word it may be connected to based on the syntactic category and the position of the non-terminal word in the input sentence (i.e. before or after the terminal word) guided by SNODE interval.

In the input sentence given above, the terminal words are "the", "old" and "green" and based on the knowledge table for the words in the input sentence, they may be connected as son node to the first non-terminal with category [n] which appear after them in the input sentence.

For ("the" 0-1, and "old" 1-2) they are connected as sons to the word ("man" 2-3).

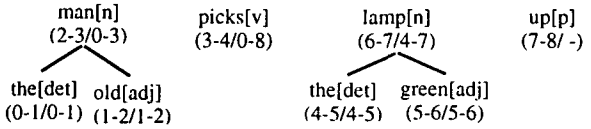


For ("the" 4-5, and "green" 5-6) they are connected as sons to the word ("lamp" 6-7).



The remainder non-terminal words, which are not connected to any terminal word, will be treated as separate substitutions.

From the input sentence the system builds the following substitutions respectively :



(1) **(2)** **(3)** **(4)**

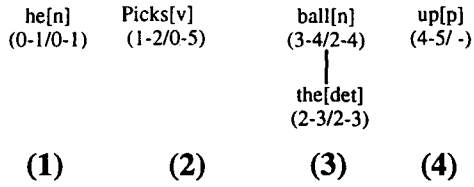
Note that this approach is quite similar to the generation of constituents in bottom-up chart parsing except that the problem of handling multiple overlapping constituents is not addressed here.

4.1.2 Substitutions combination

In order to combine the substitutions to form a complete SSTC, the system first finds non-terminal words of input sentence, which appear as root word of some dependency trees in the example SSTCs. If more than one example are found (in most cases), the system will calculate the distance between the input sentence and the examples, and the closest example

(namely one with minimum distance) will be chosen to proceed further.

In our example, the word "picks" is the only word in the sentence which can be the root word, so example (1) which containing "pick" as root will be used as the base to construct the output SSTC. The system first generates the substitutions for example (1) based on the same assumptions mentioned earlier in substitutions generation, which are :



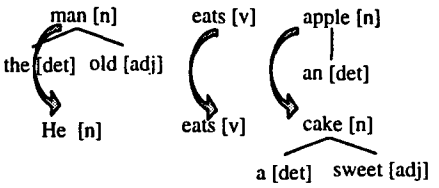
Distance calculation:

Here the system utilizes distance calculation to determine the plausible example, which SSTC structure will be used as a base to combine the substitutions at the input sentence. We define a heuristic to calculate the distance, in terms of editing operations. Editing operations are insert ($\varepsilon \rightarrow p$), deletion ($p \rightarrow \varepsilon$) and replacing ($a \rightarrow s$). Edition distances, which have been proposed in many works [7], [8] and [9], reflect a sensible notion, and it can be represented as metrics under some hypotheses. They defined the edition distances as number of editing operations to transfer one word to another form, i.e. how many characters needed to be edited based on insertion, deletion or replacement. Since words are strings of characters, sentences are strings of words, editing distances hence are not confined to words, they may be used on sentences [6].

With the similar idea, we define the edition distance as: ***(i)*** The distance is calculated at level of substitutions (i.e. only the root nodes of the substitutions will be considered, not all the words in the sentences). ***(ii)*** The edit operations are done based on the syntactic category of the root nodes, (i.e. the comparison between the input sentence and an example is based on the syntactic category of the root nodes of their substitutions, not based on the words). The distance is calculated based on the number of editing operations (deletions and insertion) needed to transfer the input sentence substitutions to the example substitutions, by assigning weight to each of these operations: 1 to insertion and 1 to deletion.

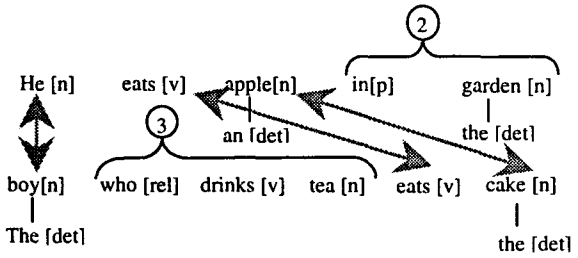
e.g. :

a) S1: The old man eats an apple.
S2: He eats a sweet cake.



In (a), the distance between S1 and S2 is 0.

b) S1: He eats an apple in the garden.
S2: The boy who drinks tea eats the cake.



In (b), the distance between S1 and S2 is (3+2)=5.

Note that when a substitution is decided to be deleted from the example, all the words of the related substitutions (i.e. the root of the substitutions and all other words that may link to it as brothers, or son/s), are deleted too. This series is determined by referring to an example containing this substitution in the example-base. For example in (b) above, the substitution rooted with "who" must be deleted, hence substitutions "drinks" and "tea" must be deleted too, similarly "in" must be deleted hence "garden" must be deleted too.

Before making the replacement, the system must first check that the root nodes categories for substitutions in both the example and the input sentence are the same, and that these substitutions are occurred in the same order (i.e. the distance is 0). If there exist additional substitutions in the input sentence (i.e. the distance $\neq 0$), the system will either combine more than one substitution into a single substitution based on the knowledge index before replacement is carried out or treat it as optional substitution which will be added as additional subtree under the root. On the other hand, additional substitutions appear in the example will be treated as optional substitutions and hence can be removed. Additional substitutions are determined during distance calculation.

Replacement:

Next the substitutions in example (1) will be replaced by the corresponding substitutions generated from the input sentence to form a final SSTC. The replacement

process is done by traversing the SSTC tree structure for the example in preorder traversal, and each substitution in the tree structure replaced with its corresponding substitution in the input sentence. This approach is analogous to top down parsing technique. Figure 8, illustrates the parsing schema for the input sentence " *The old man picks the green lamp up*".

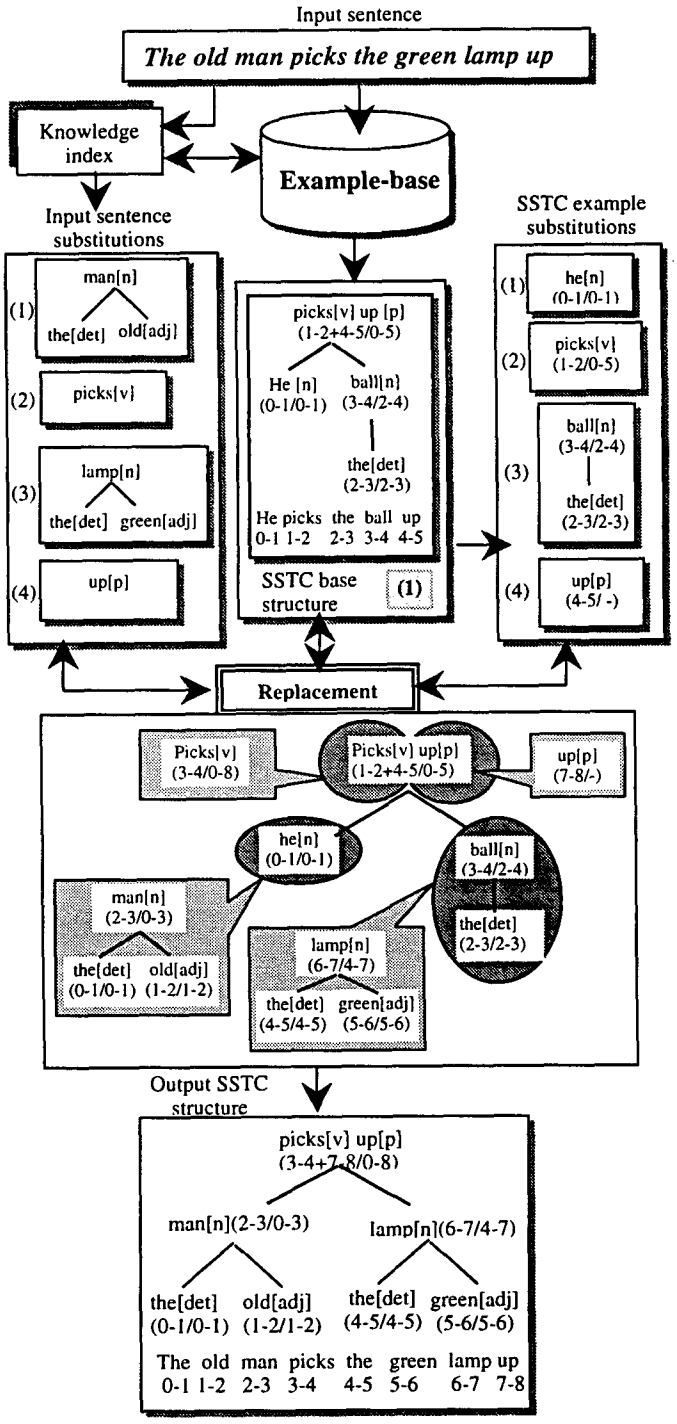


Figure 8: The parsing schema based on the SSTC for the sentence "the old man picks the green lamp up" using example (1).

5. CONCLUSION

In this paper, we sketch an approach for parsing NL string, which is an example-based approach relies on the examples that already parsed to their representation structures, and on the knowledge that we can get from these examples information needed to parse the input sentence.

A flexible annotation schema called Structured String-Tree Correspondence (SSTC) is introduced to express linguistic phenomena such as featurisation, lexicalisation and crossed dependencies. We also present an overview of the algorithm to parse natural language sentences based on the SSTC annotation schema. However, to obtain a full version of the parsing algorithm, there are several other problems which needed to be considered further, i.e. the handling of multiple substitutions, an efficient method to calculate the distance between the input sentence and the examples, and lastly a detailed formula to compute the resultant SSTC obtained from the combination process especially when deletion of optional substitutions are involved.

References:

[1] M.Nagao, "A Framework of a mechanical translation between Japanese and English by analogy principle", in; A. Elithorn, R. Benerji, (Eds.), Artificial and Human Intelligence, Elsevier: Amsterdam.

[2] V.Sadler & Vendelmans, "Pilot implementation of a bilingual knowledge bank", *Proc. of Coling-90*, Helsinki, 3, 1990, 449-451.

[3] S. Sato & M.Nagao, "Example-based Translation of technical Terms", *Proc. of TMI-93*, Koyoto, 1993, 58-68.

[4] Y. Zaharin & C. Boitet, "Representation trees and string-tree correspondences", *Proc. of Coling-88*, Budapest, 1988, 59-64.

[5] E. K. Tang & Y. Zaharin, "Handling Crossed Dependencies with the STCG", *Proc. of NLPRS'95*, Seoul, 1995.

[6] Y.Lepage & A.Shin-ichi, "Saussurian analogy: a theoritical account and its application", *Proc. of Coling-96*, Copenhagen, 2, 1996, 717-722.

[7] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals", *Dokl. Akad. Nauk SSSR*, 163, No. 4, 1965, 845-848. *English translation in Soviet Physics-doklady*, 10, No. 8, 1966, 707-710.

[8] Robert A. Wagner & Michael J. Fischer, " The String-to String Correction Problem", *Journal for the Association of Computing Machinery*, 21, No. 1, 1974, 168-173.

[9] Stanley M. Selkow, "The Tree-to-Tree Editing Problem", *Information Processing Letters*, 6, No. 6, 1977, 184-186.