CONSTRUCTOR: A Natural Language Interface Based on Attribute Grammar*, **

Z. ALEXIN[1], J. DOMBI[1], K. FABRICZ,[2] T. GYIMOTHY,[1] T. HORVATH[1]

[1] *Research Group on Theory of Automata*
Hungarian Academy of Sciences, Somogyi u. 7., H—6720 Szeged
[2] *Attila József University, Egyetem u. 2., H—6722 Szeged*

Abstract

The paper gives an overview of a natural language interface currently being developed at the Research Group on Theory of Automata at the Hungarian Academy of Sciences in collaboration with the Attila József University, Szeged. The interface supports natural language communication between the user issuing commands as steps in plane geometry constructions and the actual graphical presentation. The Natural Language Interface named CONSTRUCTOR is described and the experiences of the authors are outlined with a view to generalizing the results thus obtained.

1. Introduction

Natural Language Interfaces [NLIs] are one of the most common applications of natural language processing. The majority of such interfaces have been developed for manipulating databases [Cliff 88].

The literature on the methodology for NLI evaluation is by and large restricted to interfaces to databases [Schr 88]. Other kinds of NLIs do not only lack general principles for objective evaluation; their value is rather hard to assess due to the fact that they are usually oriented to some special task with a microcosm of words, rules, and knowledge.

We can roughly distinguish two types of Natural Language Interfaces. Less complicated Natural Language Interfaces are based on a sentence-by-sentence analysis. As a rule, they extract information from the main constituents of sentences.

*The computer facilities provided by the Alexander von Humboldt Foundation have greatly contributed to the completion of this project.

** Lecture presented at the 1st Finnish—Hungarian Workshop on Programming Languages and Software Tools, Szeged, Hungary, August 8—11, 1989.

This approach allows for skipping different parts of the input while restricting parsing to finding the words of direct semantic relevance. In case of a simple NLI, knowledge representation does not go hand in hand with natural language analysis. In fact, here natural language understanding is replaced by pattern matching and pre-wired procedures. These interfaces are relatively easy to construct and they can even be made portable. An example of such a system is JAKE [JAKE 88].

On the other hand, more detailed analysis along with deeper understanding is achieved by interfaces which do not omit parts of the input considering them irrelevant. Rather, they are designed to capture the overall content of the input, therefore they are suitable for analyzing intersentential relations. Their construction, however, presupposes global understanding of what the input is about. Therefore, they can provide insight into what representing knowledge or understanding natural language means. Although, in principle, their transportability is per se precluded, later in this paper some considerations suggest that this should not be the case.

2. CONSTRUCTOR — an NLI for plane geometry constructions

CONSTRUCTOR, the NLI we are currently working on belongs to this latter group of interface systems. It has been designed to accept English sentences used as instructions for plane geometry constructions. The basic idea is to let the user issue commands whose output is a step in producing a more or less complicated geometrical construction. (A prototype NLI for plane geometry constructions is described in [Arz 85], where an interface of the simpler kind is presented.)

With CONSTRUCTOR, the main steps to be taken are:

(i) analyze the input in its entirety,
(ii) translate the result into a semantic representation,
(iii) produce, on the basis of (ii), a visualized construction,
(iv) keep track of the sequence of inputs in order to:

   a) maintain control over the whole procedure of construction,
   b) supply the user with a means of feed-back (evaluation).

Thus, CONSTRUCTOR is part of a program package that consists of the following basic modules.

CONSTRUCTOR itself consists of the following parts:

a lexical analyzer
a syntactic parser
an attribute evaluator
a semantic interpreter
a construction creator

These moduls can be briefly described as follows.

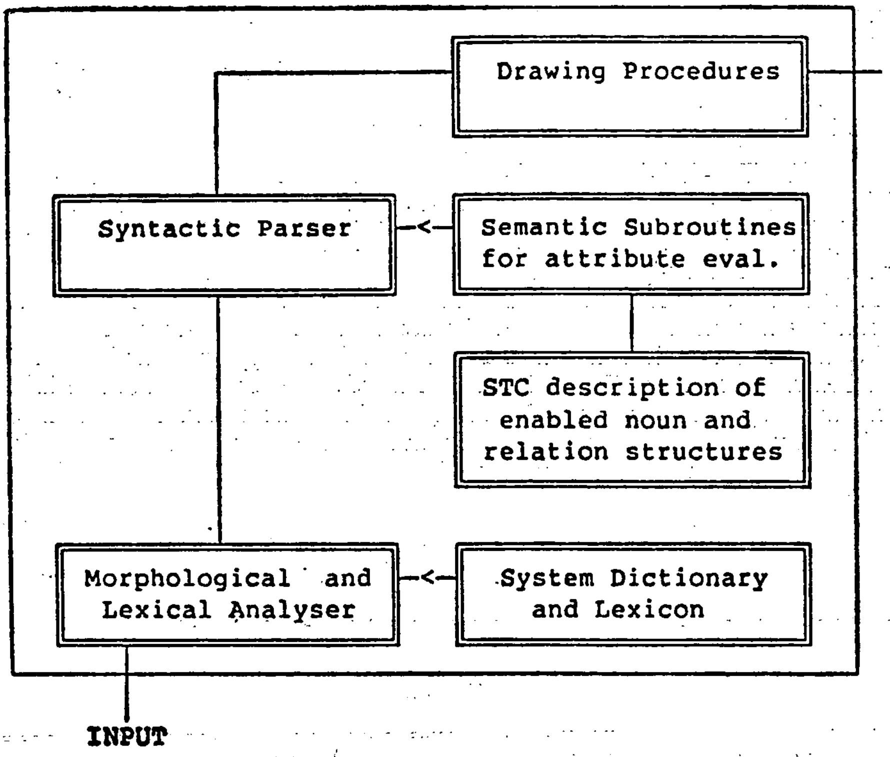
Drawing Procedures
Syntactic Parser
Semantic Subroutines for attribute eval.
STC description of enabled noun and relation structures
Morphological and Lexical Analyser
System Dictionary and Lexicon
INPUT

Figure 1.

2.1. The lexical analyzer

The lexical analyzer consists of a machine dictionary, a scanner and a morphological analyzer.

The dictionary of CONSTRUCTOR contains a lexicon of more than 300 items necessary for issuing commands (a typical set of instructions is provided with the description of the syntactic analyzer). The word stock incorporated in the lexicon makes it possible to maintain ease of reference. Thus the information contained in one instruction is related to other pieces of information from a sequence of commands.

A lexical entry consists of a word-stem (canonical form) and a set of codes that are necessary for subsequent analysis. In fact, it is in the lexical entry that basic morphological, syntactic, and semantic information is stored. Morphological features involve data for the derivation of inflected forms. From the point of view of syntax, the lexicon helps the parser assign a token to a particular word. The semantic information contained in the entry is the basic synthetised attribute underlying the final process of analysis, i.e. attribute evaluation.

The lexicon holds synonymous words to account for the fact that there is a difference in word usage among students depending on age and/or level of training (see, e.g. selection from synonymous verbs like "name", "mark", "label", "denote", or "designate" according to the above factors).

The lexical module is virtually extended by a morphological analyzer. Its function is to trace all the word-forms not found in the lexicon to their canonical lexeme. Its work is based on a tagging algorithm for the derivation of inflected word-forms. The relevant information for finding the base form (stem) of a lexeme is encoded in the lexical entry. The algorithm facilitates the derivation of all the four major parts of speech: verbs (past tense, past participle, third person in the present tense, and gerund), nouns (plural forms), adjectives (comparative and superlative degree), and adverbs (degrees and "-ly" traced back to adjectival canonical forms with the derivational path recorded). The inclusion of a morphological analyzer reduces the size of the lexicon to a minimum, while the overall amount of actual word-forms is potentially well over a thousand. In some cases actual word-forms appear in the lexicon along with their canonical form. This is due to matters of conversion, that is, some inflected forms may represent products of a change in linguistic status. For instance, the word-form "circumscribed" appears to be an adjective rather than a past participle. In this case the algorithm would yield a false result in the sense that it would analyze the form as the past participle of "circumscribe" instead of assigning it to the class of adjectives in base form. Here we create a lexical entry for "circumscribed" with the codes for an adjective. As dictionary look-up takes place prior to derivation, a match for "circumscribed" is found with no conflict with the analyzer. Table 1 shows the main morphological derivations handled by the analyzer.

Table 1.

MORPHOLOGICAL CATEGORIES	STEMS	VARIATIONS	INFLECTIONS
VERBAL MORPHEMES:	apply		-ing
		applie-	-s
(Canonical form: 'apply')		applie-	-d
NOMINAL MORPHEMES:	copy		
		copie-	-s
(Canonical form: 'copy')			
ADJECTIVAL/ADVERBIAL MORPHEMES:	big		
(Canonical form: 'big')		bigg-	-er
(Canonical form: 'great')	great		-er
(Canonical form: 'equal')	equal		-ly

2.2. Syntactic parsing

The input to the syntactic parser is a string of tokens and terminals to be processed into a sentence (or a list of sentences) with some structure assigned to the input on the basis of about two hundred re-writing rules. The parser is a hypothesis-driven (top-down) depth-first left-to-right syntactic analyzer. The syntactic rules represent a context-free grammar description. As for the type of look-ahead, the syntax is basically of the LL(1) type. The only exceptions are conjunctions together with a conjunction and more or less optional commas (","), and a few minor constructs.

The sentences that can be processed by the parser may be fairly complex. The only significant restriction imposed is that one sentence may refer to but one step of construction. It means that issuing commands like "Draw and move a right triangle..." is prohibited. On the other hand, nested sentences for object specification can be used. It means that sentences like "Draw a segment that connects points ˜*A* and ˜*B*." or "Label the line that crosses circle ˜*C* at points ˜*A* and ˜*B* by ˜1." can be freely used. To get a grasp of the range of sentences accepted by CONSTRUCTOR, consider the table below:

Table 2.

FRAME SENTENCE STRUCTURES

1. Draw two parallel lines.
   (Verb Phrase)(Noun Phrase)
2. Construct a triangle inside the circle.
   (Verb Phrase)(Noun Phrase)(Prep Phrase)
3. From point ˜A, drop a vertical line.
   (Pre-Specifier) (VP) (NP)
4. Label by ˜e a straight line that is above the circle.
   (VP) (NP) (Specifier)
5. Label by ˜J a point that divides segment ˜O˜B into parts with a proportion of 1:3.
   (VP) (NP) (Specifier within Specifier)
6. By measuring off the length of segment ˜A˜B, draw two circles with radius ˜A˜B at a distance equal to the difference between the base of the triangle and side ˜U˜V of the heptadecagon.
   (PreVP) (VP) (NP) (Specifier within Specifier)

2.3. Attribute evaluation for the basic grammatical structures

The system uses an L-attribute evaluation strategy. Its task is to compute the basic features of grammatical structures. For example, the attributes of the verbal object can be computed from the attributes synthetized for the noun phrase, the adjectival phrase, and the apposition. This computation mostly involves synthetized attribute evaluation, whereas further specification of the object (localization, relatedness etc.) requires the use of inherited attributes.

Facing the complexity of the sentences above does not appear to be a simple enterprise. Nevertheless, there do seem to be clues to semantic interpretation.

For one thing, there are some observations that can be made use of for a more thorough understanding of the semantic relations involved.

Prepositions, for example, correspond to markers of localization. Localization is taken to refer to either a place or a direction in the plane cf.:

Mark a point on the circle.
Move the triangle up.

Adjectives and nouns enter into relations of selectional restrictions, cf.:

* Draw a parallel circle.

Verbs appear to invoke one or more of the following actions:

drawing,
marking,
measuring,
manipulating.

These action types often result in overlapping actions due to the vagueness present in natural languages, cf.:

Label by $\bar{e}$ an arbitrary line.
1. Draw an arbitrary line.
2. Label it by $\bar{e}$.

Drop a vertical line $\bar{e}$.
1. Drop a vertical line.
2. Label it by $\bar{e}$.

2.4. Semantic interpretation

The result of syntactic parsing, attribute evaluation, and the observations all serve as input for semantic interpretation. The main bulk of analysis at this stage, however, is done through a metalevel description for building complex noun phrases. A part of the metalevel description is shown in Figure 2 below:

```
Objects are
                          {POSITION data structures}

IntegerPosition          is  X (alias X~ coordinate): IntegerNumber;
IntegerPosition          is  Y (alias Y~ coordinate): IntegerNumber;
RealPosition             is  X (alias X~ coordinate): RealNumber;
RealPosition             is  Y (alias Y~ coordinate): RealNumber;

                          {DESIGNATION Data structures}

Designation              is  ObjectName (alias Name);
Designation              has IntegerPosition (alias Locus);
Designation              has RealPosition (alias Locus);

Dot                      is  IntegerPosition;
Dot                      is  RealPosition;
Dot                      has Designation (alias Name);

                          {TRIANGLE data structures}

TriangleBySideType   = (EquiAngular (alias EquiLateral),
                        Isosceles, Scalene (alias General));
TriangleByAngleType  = (Acute, Right, Obtuse);

Triangle                 is  Dot;
Triangle                 is  Dot;
Triangle                 is  Dot;
Triangle                 is  Designation (alias Name);

Triangle                 is  of SizeType;
Triangle                 is  of TriangleBySideType;
Triangle                 is  of TriangleByAngleType;

Triangle                 has Edge [3] (alias Side): Line;
Triangle                 has Angle [3];
Triangle                 has Center ~Line [3]: Line;
Triangle                 has MidPoint [3]: Dot;

Triangle                 has Circumscribed ~Circle: Ellips;
Triangle                 has Inscribed ~Circle: Ellips;
Triangle                 has Circumference: Length;
Triangle                 has Area: RealNumber;
```

Figure 2.

Another difficulty is computing the relations between the objects involved in some construction. Different kinds of specifiers get evaluated by way of logical expressions and mathematical functions and equations. For example, the location "on the triangle" is computed from the equations relating to the three sides of the triangle and defining a set of points to be found "on" the triangle.

From the nodes of a given triangle we can compute the equations for the edges of the triangle. If the coordinates of the point are within the sets of points defined by the equations, then the relation "on the triangle" holds.

2.5. *The execution of commands*

The action creator receives as an input a complete specification of the object to be created and it defines the procedure to be executed with all the parameters set. The definition does not involve arriving at a possible solution of the specifications but also questions of a suitable appearance are of relevance.

Although the set of sentences presented above may seem impressive as far as syntactic and semantic complexity are concerned, the most prominent feature of CONSTRUCTOR is most likely its ability to handle reference to some previously defined object or action. This feature of CONSTRUCTOR does not simply imply a syntactic sugar of using words like "it" or "its" but, from a broader perspective, it opens the way to picturing a series of instructions as related steps of some geometrical construction. Keeping a record of what has been done makes it possible to resolve or, at least, detect, cases of ambiguity.

3. Summary

The kind of natural language interface under consideration appears to be a perspective candidate for a large scale of applications from CAD through text editors to intelligent database query languages. Our aim has been to develope a software tool for generating NLIs of this kind.

Since a software generator is considered the right tool in case

a) it can generate a major part of the software, and

b) it can provide some high level user friendly means for the description of the variable parts (cf. [Mart 83]),

we have tried to find the more or less readily standardizable parts of CONSTRUCTOR and provide a metalanguage for the specification of the variable parts.

In the case of CONSTRUCTOR that has basically been generated by a generator based on attribute grammars, the following modules seem to have been apt to generation:

— its lexical analyzer is highly suitable for generation.

— the algorithm for morphological derivation appears as a standard procedure of the lexical analyzer. We have constructed a convenient tool for dictionary maintenance.

— the syntactic parser is easily generated by PROF—LP [Gyim88] as long as the number of LL(1) conflicts is kept to a minimum. In other cases, procedures defined by the user can be implemented (this has only partly been carried out in the present version). Slight modifications in the syntactic description of CONSTRUCTOR might be sufficient for applications in syntactically related domains.

— a considerable amount of attribute evaluation can be standardized. In cases where linguistic structure shows significant variation (e.g. the structure of objects), the metalevel description can be used for object definition. This description is the basis of the procedures that handle the object table. There are several parts of the specification which are suitable for generalization, but others are problem-specific. Here, again, the metalevel description can provide a possible way-out by defining clues for establishing relations between objects.

— although the implementation of relations depends on the very application, it seems probable that a natural language interface connected to some CAD or data-base would have much in common with CONSTRUCTOR,

— at present we cannot give a positive answer as to whether the actions invoked by CONSTRUCTOR could be straightforwardly transferred to some other Natural Language Interface, if at all, but a deeper insight in the semantic configuration of the class of verbs might lead to some result in the future.

4. Further research

Our further research in the area of Natural Language Interface generation will mainly be oriented to developing a generator that generates Natural Language Interfaces in a unique framework (now PROF—LP and metalevel object descriptions are separate entities). Another field of interest would be developing further methods for generalization.

References

[Arz85] ARZ, J.: TRICON Ein System für geometrische Konstruktionen mit natürlichsprachlicher Eingabe, Technische Bericht, Universität des Saarlandes, Saarbrücken, KI—LABOR.

[Cliff88] CLIFFORD, J.: Natural Language Querying of Historical Databases, Computational Linguistics 1988, 14 (4), 10—34 pp.

[Gyim88] GYIMÓTHY, T., HORVÁTH, T., KOCSIS, F., TOCZKI, J.: Incremental algorithms in PROF—LP, Lecture Notes in Computer Science Vol. 371, 93—102 pp.

[JAKE88] JAKE The application-independent natural language user interface, English Knowledge Systems, Inc. Scotts Valley, California, 1988.

[Mart83] MARTIN, P., APPELT, D., PEREIRA, F.: Transportability and Generality in a Natural Language Interface System, Proceedings of IJCAI—83. Vol. 1, 573—581 pp.

[Schr88] SCHRÖDER, M.: Evaluating User Utterances in Natural Language Interfaces to Databases, Computers and Artificial Intelligence 1988 (7), 317—337 pp.