GTT : A GENERAL TRANSDUCER FOR TEACHING COMPUTATIONAL LINGUISTICS

P. Shann J.L. Cochard
Dalle Molle Institute for Semantic and Cognitive Studies
University of Geneva
Switzerland

ABSTRACT

The GTT-system is a tree-to-tree transducer developed for teaching purposes in machine translation. The transducer is a specialized production system giving the linguists the tools for expressing information in a syntax that is close to theoretical linguistics. Major emphasis was placed on developing a system that is user friendly, uniform and legible. This paper describes the linguistic data structure, the rule formalism and the control facilities that the linguist is provided with.

1. INTRODUCTION

The GTT-system (Geneva Teaching Transducer)[1] is a general tree-to-tree transducer developed as a tool for training linguists in machine translation and computational linguistics. The transducer is a specialized production system tailored to the requirements of computational linguists providing them with a means of expressing information in a format close to the linguistic theory they are familiar with.

GTT has been developed for teaching purposes and cannot be considered as a system for large scale development. A first version has been implemented in standard Pascal and is currently running on a Univac 1100/61 and a VAX-780 under UNIX. At present it is being used by a team of linguists for experimental development of an MT system for a special purpose language (Buchmann et al., 1984), and to train students in computational linguistics.

2. THE UNIFORMITY AND SIMPLICITY OF THE SYSTEM

As a tool for training computational linguists, major emphasis was placed on developing a system that is user friendly, uniform, and which provides a legible syntax.

One of the important requirements in machine translation is the separation of linguistic data and algorithms (Vauquois, 1975). The linguist should have the means to express his knowledge declaratively without being obliged to mix computational algorithms and linguistic data. Production systems (Rosner, 1983) seem particularly suited to meet such requirements (Johnson, 1982); the production set that expresses the object-level knowledge is clearly separated from the control part that drives the application of the productions. Colmerauer's Q-system is the classic example of such a uniform production system used for machine translation (Colmerauer, 1970; Chevalier, 1978: TAUM-METEO). The linguistic knowledge is expressed declaratively using the same data structure during the whole translation process as well as the same type of production rules for dictionary entries, morphology, analysis, transfer and generation. The disadvantage of the Q-system is its quite unnatural rule-syntax for non-programmers and its lack of flexible control mechanism for the user (Vauquois, 1978).

In the design of our system the basic uniform scheme of Q-systems has been followed, but the rule syntax, the linguistic data structure and the control facilities have been modernized according to recent developments in machine translation (Vauquois, 1978; Boitet, 1977; Johnson, 1980; Slocum, 1982). These three points will be developed in the next section.

3. DESCRIPTION OF THE SYSTEM

3.1 Overview

The general framework is a production system where linguistic object knowledge is expressed in a rule-based declarative way. The system takes the dictionaries and the grammars as data, compiles these data and the interpreter then uses them to process the input text. The decoder transforms the result into a digestable form for the user.

3.2 Data structure

The data structure of the system is based on a chart (Varile, 1983). One of the main advantages of using a chart is that the data structure does not change throughout the whole process of translation (Vauquois, 1978).

In the Q-system all linguistic data on the arcs is represented by bracketed strings causing an unclean mixture of constituent structure and other linguistic attributes such as grammatical and semantic labels, etc. With this representation

---

[1] This project is sponsored by the Swiss government.

type checking is not possible. Vauquois proposes two changes :
1) Tree structures with complex labels on the nodes in order to allow interaction between different linguistic levels such as syntax or semantics, etc.
2) A dissociation of the geometry from a particular linguistic level. With these modifications a single tree structure with complex labels increases the power of representation in that several levels of interpretation can be processed simultaneously (Vauquois, 1978; Boîtet, 1977).

In our system each arc of the chart carries a tree geometry and each node of the tree has a complex labelling consisting of a possible string and the linguistic attributes. Through the separation of geometry and attributes, the linguist can deal with two distinct objects: with tree structures and complex labels on the nodes of the trees.

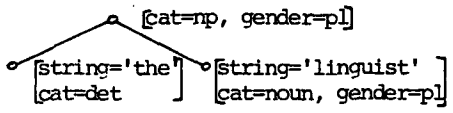


Figure 1. Tree with complex labelling

The range or kind of linguistic attributes possible is not predefined by the system. The linguist has to define the types he wants to use in a declaration part.

e.g.: category = verb, noun, np, pp.
semantic-features = human, animate.
gender = masc, fem, neut.

An important aspect of type declaration is the control it offers. The system provides strong syntactic and semantic type checking, thereby constraining the application range in order to avoid inappropriate transductions. The actual implementation allows the use of sets and subsets in the type definition. Further extensions are planned.

Given that in this system the tree geometry is not bound to a specific linguistic level, the linguist has the freedom to decide which information will be represented by the geometry and which will be treated as attributes on the nodes. This representation tool is thus fairly general and allows the testing of different theories and strategies in MT or computational linguistics.

3.3 The rule syntax

The basic tool to express object-knowledge is a set of production rules which are similar in form to context-free phrase structure rules, and well-known to linguists from formal grammar. In order to have the same rule type for all operations in a translation system the power of the rules must be of type 0 in the Chomsky classification, including string handling facilities.

The rules exhibit two important additions to context-free phrase structure rules:
- arbitrary structures can be matched on the left-hand side or built on the right-hand side, giving the power of unrestricted rules or transformational grammar;
- arbitrary conditions on the application of the rule can be added, giving the power of a context sensitive grammar.

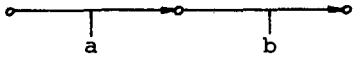
The power of unrestricted rewriting rules makes the transducer a versatile instrument for expressing any rule-governed aspect of language whether this be morphology, syntax, semantics. The fact that the statements are basically phrase structure rules makes this language particularly congenial to linguists and hence well-suited for teaching purposes.

The format of rules is determined by the separation of tree structure and attributes on the nodes. Each rule has three parts: geometry, conditions and assignments, e.g.:

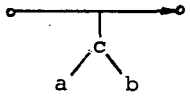
RULE 1

(geometry) a + b ⇒ c(a,b)
(conditions) IF cat(a) = [det] and cat(b) = [noun]
(assignment) THEN cat(c) := [np];

The geometry has the standard left-hand side, production symbol (⇒), and right-hand side of a production rule. a,b,c are variables describing the nodes of the tree structure. The '+' indicates the sequence in the chart, e.g. a+b :



Tree configurations are indicated by bracketing, c(a,b) corresponds to :



Conditions and assignments affect only the objects on the nodes.

3.4 Control structure

The linguist has two tools for controlling the application of the rewriting rules :
1) The rules can be grouped into packets (grammars) which are executed in sequence.
2) Within a given grammar the rule-application can be controlled by means of parameters set by the linguist. According to the linguistic operation envisaged, the parameters can be set to a combination of serial or parallel and one-pass or iterate.
In all, 4 different combinations are possible :

parallel and one-pass
parallel and iterate
serial and one-pass
serial and iterate

In the parallel mode the rules within a grammar are considered as being unordered from a logical point of view. Different rules can be applied on the same piece of data and produce alternatives in the chart. The chart is updated at the end of every application-cycle. In the serial mode the rules are considered as being ordered in a sequence. Only one rule can be fired for a particular piece of data. But the following rules can match the result produced by a preceding rule. The chart is updated after every rule that fired. The parameters one-pass and iterate control the number of cycles. Either the interpreter goes through a cycle only once, or iterates the cycles as long as any rule of the grammar can fire.

The four combinations allow different uses according to the linguistic task to be performed, e.g.:
Parallel and iterate applies the rules non-deterministically to compute all possibilities, which gives the system the power of a Turing Machine (this is the only control mode for the Q-system). Parallel and one-pass is the typical combination for dictionaries that contain alternatives. Two different rules can apply to the same piece of data. The example below (fig. 2) uses this combination in the first GRAMMAR 'vocabulary'.

Serial and one-pass allows rule ordering. A possible application of this combination is a preference mechanism via the explicit rule ordering using the longest-match-first technique. The GRAMMAR 'preference' in the example below (fig. 2) makes use of that by progressive weakening of the selectional restriction of the verb 'drink'. Rule 24 fires without semantic restrictions and rule 25 accepts sentences where the optional argument is missing.

The example should be sufficiently self-explanatory. It begins with the declaration of the attributes and contains three grammars. The result is shown for two sentences (fig. 3). To demonstrate which rule in the preference grammar has fired each rule produces a different top label: rule 21 ⟶ PH1, rule 22 ⟶ PH2, etc.

Figure 2. Example of a grammar file.

```
DECLARE
  cat = det, noun, verb, val_node, np, ph1, ph2, ph3, ph4, ph5;
  number = sg, pl;
  marker = human, liquid, notdrinkable, physobj, abstr;
  valency = v1, v2, v3;
  argument = arg1, arg2, arg3;

GRAMMAR vocabulary      PARALLEL ONEPASS

 RULE 1   a => a
   IF       string(a)= 'the'
   THEN     cat(a) := [det];

 RULE 2   a => a
   IF       string(a)= 'man'
   THEN     cat(a) := [noun]; number(a) := [sg];
            marker(a) := [human];

 RULE 3   a => a
   IF       string(a)= 'beer'
   THEN     cat(a) := [noun]; number(a) := [sg];
            marker(a) := [liquid];

 RULE 4   a => a
   IF       string(a)= 'car'
   THEN     cat(a) := [noun]; number(a) := [sg];
            marker(a) := [physobj];

 RULE 5   a => a
   IF       string(a)= 'gazoline'
   THEN     cat(a) := [noun]; number(a) := [sg];
            marker(a) := [notdrinkable];

 RULE 6   a => a
   IF       string(a)= 'drinks'
   THEN     cat(a) := [noun]; number(a) := [pl];
            marker(a) := [liquid];

 RULE 7   a => a(b,c)
   IF       string(a)= 'drinks'
   THEN     cat(a):=[verb]; valency(a):=[v2];
            cat(b):=[val_node];  cat(c):=[val_node];
            argument(b):=[arg1]; marker(b):=[human];
            argument(c):=[arg2]; marker(c):=[liquid];

GRAMMAR  nounphrase    SERIAL ONEPASS

 RULE 11  a + b => c(a,b)
   IF     cat(a) = [det] and cat(b) = [noun]
   THEN   cat(c) := [np]; marker(c) := marker(b);

GRAMMAR preference     SERIAL ONEPASS

 RULE 21  a + b(#1,c,#2,d,#3) + e  => x(b,a,e)
   IF   cat(a)=[np] and cat(b)=[verb] and cat(e)=[np]
        and valency(b)=[v3]
        and argument(c)=[arg1] and marker(c)=marker(a)
        and argument(d)=[arg2] and marker(d)=marker(e)
   THEN cat(x) := [ph1];

 RULE 22  a + b(#1,c,#2) + e  => x(b,a,e)
   IF   cat(a)=[np] and cat(b)=[verb] and cat(e)=[np]
        and valency(b)=[v2]
        and argument(c)=[arg1] and marker(c)=marker(a)
   THEN cat(x) := [ph2];

 RULE 23  a + b(#1,c,#2) + e  => x(b,a,e)
   IF   cat(a)=[np] and cat(b)=[verb] and cat(e)=[np]
        and valency(b)=[v2]
        and argument(c)=[arg2] and marker(c)=marker(e)
   THEN cat(x) := [ph3];

 RULE 24  a + b + e  => x(b,a,e)
   IF   cat(a)=[np] and cat(b)=[verb] and cat(e)=[np]
        and valency(b)=[v2]
   THEN cat(x) := [ph4];

 RULE 25  a + b  => x(b,a)
   IF   cat(a)=[np] and cat(b)=[verb]
        and valency(b)=[v2]
   THEN cat(x) := [ph5];

ENDFILE
```

Figure 3. Output of upper grammar file.

```
Input sentence :
    (1)  The man drinks the beer.

Result :
PH1 CAT=[PH1]
|-'DRINKS' CAT=[VERB] VALENCY=[V2]
| |-VAL_NODE CAT=[VAL_NODE] MARKER=[HUMAN] ARGUMENT=[ARG1]
| |-VAL_NODE CAT=[VAL_NODE] MARKER=[LIQUID] ARGUMENT=[ARG2]
|-NP CAT=[NP] MARKER=[HUMAN]
| |-'THE' CAT=[DET]
| |-'MAN' CAT=[NOUN] NUMBER=[SG] MARKER=[HUMAN]
|-NP CAT=[NP] MARKER=[LIQUID]
  |-'THE' CAT=[DET]
  |-'BEER' CAT=[NOUN] NUMBER=[SG] MARKER=[LIQUID]

Input sentence :
    (2)  The man drinks the gazoline.

Result :
PH2 CAT=[PH2 ]
|-'DRINKS' CAT=[VERB] VALENCY=[V2]
| |-VAL_NODE CAT=[VAL_NODE] MARKER=[HUMAN] ARGUMENT=[ARG1]
| |-VAL_NODE CAT=[VAL_NODE] MARKER=[LIQUID] ARGUMENT=[ARG2]
|-NP CAT=[NP] MARKER=[HUMAN]
| |-'THE' CAT=[DET]
| |-'MAN' CAT=[NOUN] NUMBER=[SG] MARKER=[HUMAN]
|-NP CAT=[NP] MARKER=[NOTDRINKABLE]
  |-'THE' CAT=[DET]
  |-'GAZOLINE' CAT=[NOUN] NUMBER=[SG] MARKER=[NOTDRINKABLE]
```

4. FACILITIES FOR THE USER

There is a system user-interaction in the two main programs of the system, the compiler and the interpreter. The following example (fig. 4) shows how the error messages of the compiler are printed in the compilation listing. Each star with a number points to the approximate position of the error and a message explains the possible errors. The compiler tries to correct the error and in the worst case ignores that portion of the text following the error.

```
GRAMMAR errortest
  PARALEL ITERATE
   *0
pos. 0 : -ES- /SERIAL/ ou /PARALLEL/ attendu

RULE 1
  a+b => c(a,b)
  IF STRING(a)='blabla' AND cat(b)=[nom THEN cat(d) := [nom];
                                    *0*1      *2
pos. 0 : -ES- /,/ attendue
pos. 1 : -ES- /]/ attendue
pos. 2 : -SEM- id. pas defini dans la geometrie (cote droit)

RULE 2
  a(a) => c(a,b)
    *0
pos. 0 : -SEM- id. deja utilise sur partie gauche

  IF cat(a)=[det] THEN categ(b) := [noun];
                    *0          *1
pos. 0 : -SEM- id. ne represente pas un ensemble
pos. 1 : -SEM- id. ne represente pas un element
```

Figure 4. Compilation listing with error message.

The interpreter has a parameter that allows the sequence of rules that fired to be traced. The trace in figure 5 below corresponds to the execution of the example (1) in figure 3.

```
interpreteur de Q-codes  83.1   fev-14-84

 application de la regle 1
 application de la regle 1
 application de la regle 2
 application de la regle 3
 application de la regle 6
 application de la regle 7
VOCABULARY execute(e)

 application de la regle 11
 application de la regle 11
NOUNPHRASE execute(e)

 application de la regle 21
PREFERENCE execute(e)

temps d'interpretation : 0.516 sec. CPU
                         3.583 sec. utilisateur
```

Figure 5. Trace of execution.

5. CONCLUSION

The transducer is implemented in a modular style to allow easy changes to or addition of components as the need arises. This provides the possibility of experimentation and of further development in various directions:

- integration of a lexical database with special editing facilities for lexicographers;
- developments of special interpreters for transfer or scoring mechanisms for heuristics;
- refinement of linguistically motivated type checking.

In this paper we have mainly concentrated on syntactic applications to illustrate the use of the transducer. However, as we hope to have shown, the formalism of the system is general enough to allow interesting applications in various domains of linguistics such as morphology, valency matching and preference mechanisms (Wilks, 1983).

ACKNOWLEDGEMENTS

Special thanks should go to Roderick Johnson of CCL, UMIST, who contributed a great deal in the original design of the system presented here, and who, through frequent fruitful discussion, has continued to stimulate and influence later developments, as well as to Dominique Petitpierre and Lindsay Hammond who programmed the initial implementation. We would also like to thank all members of ISSCO who have participated in the work, particularly B. Buchmann and S. Warwick.

REFERENCES

Buchmann, B., Shann, P., Warwick, S. (1984). Design of a Machine Translation System for a Sublanguage. Proceedings, COLING '84.

Chevalier, M., Dansereau, J., Poulin, G. (1978). TAUM-METEO : description du système. T.A.U.M., Groupe de recherche en traduction automatique, Université de Montréal, janvier 1978.

Colmerauer, A. (1970). Les systèmes-Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur. Université de Montréal.

Johnson, R.L. (1982). Parsing - an MT Perspective. In: K. Spark Jones and Y. Wilks(eds.), Automatic Natural Language Parsing, Memorandum 10, Cognitive Studies Centre, University of Essex.

Rosner, M. (1983). Production Systems. In: M. King (ed.), Parsing Natural Language, Academic Press, London.

Slocum, J. and Bennett, W.S. (1982). The LRC Machine Translation System: An Application of State-of-the-Art Text and Natural Language Processing Techniques to the Translation of Technical Manuals. Working paper LRC-82-1, Linguistics Research Center, University of Texas at Austin.

Vauquois, B. (1975). La traduction automatique à Grenoble. Documents de Linguistique Quantitative, 24. Dunod, Paris.

Vauquois, B. (1978). L'évolution des logiciels et des modèles linguistiques pour la traduction automatisée. T.A. Informations, 19.

Varile, G.B. (1983). Charts: A Data Structure for Parsing. In: M. King (ed.), Parsing Natural Language, Academic Press, London.

Wilks, Y. (1973). An Artificial Intelligence Approach to Machine Translation. In: R.C. Schank and K.M. Colby (eds.), Computer Models of Thought and Language, W.H. Freeman, San Francisco., pp. 114-151.