A Functional Approach to Generation with TAG[1]

Kathleen F. McCoy, K. Vijay-Shanker, & Gijoo Yang
Department of Computer and Information Sciences
University of Delaware
Newark, Delaware 19716, USA
email: mccoy@udel.edu, vijay@udel.edu

Abstract

It has been hypothesized that Tree Adjoining Grammar (TAG) is particularly well suited for sentence generation. It is unclear, however, how a sentence generation system based on TAG should choose among the syntactic possibilities made available in the grammar. In this paper we consider the question of what needs to be done to generate with TAGs and explain a generation system that provides the necessary features. This approach is compared with other TAG-based generation systems. Particular attention is given to Mumble-86 which, like our system, makes syntactic choice on sophisticated functional grounds.

1 Introduction

Joshi (1987) described the relevance of Tree Adjoining Grammar (TAG) (Joshi, 1985; Schabes, Abeille & Joshi, 1988) to Natural Language Generation. In particular, he pointed out how the unique factoring of recursion and dependencies provided by TAG made it particularly appropriate to derive sentence structures from an input provided by a text planning component. Of particular importance is the fact that (all) syntactic dependencies and function argument structure are localized in TAG trees.

Shieber and Schabes (1991) discuss using Synchronous TAG for generation. Synchronous TAG provides a formal foundation to make explicit the relationship between elementary syntactic structures and their corresponding semantic counterparts, both expressed as elementary TAG trees. This relationship is made explicit by pairing the elementary trees in the syntactic and logical form languages, and associating the corresponding nodes. Shieber and Schabes (1990) describe a generation algorithm which "parses" an input logical form string recording the adjoining and substitution operations necessary to build the string from its elementary components. The corresponding syntactic structure is then generated by doing the same set of operations (in reverse) on the corresponding elementary structures in the grammar describing the natural language.

Note that the generation methodology proposed for synchronous TAG (and the hypothetical generator alluded to in (Joshi, 1987)) takes as input the *logical form* semantic representation and produces a syntactic representation of a natural language sentence which captures that logical form. While the correspondence between logical form and the natural language syntactic form is certainly an important and necessary component of any sentence generation system, it is unclear how finer distinctions can be made in this framework. That is, synchronous TAG does not address the question of *which* syntactic rendition of a particular logical form is most appropriate in a given circumstance. This aspect is particularly crucial from the point of view of generation. A full-blown generation system based on TAG must choose between various renditions of a given logical form on well-motivated grounds.

Mumble-86 (McDonald & Pustejovsky, 1985; Meteer et al., 1987) is a sentence generator based on TAG that is able to take more than just the logical form representation into account. Mumble-86 is one of the foremost sentence generation systems and it (or its predecessors) has been used as the sentence generation components of a number of natural language generation projects (e.g., (McDonald, 1983; McCoy, 1989; Conklin & McDonald, 1982; Woolf & McDonald, 1984; Rubinoff, 1986)). After briefly describing the methodology in Mumble-86, we will point out some problematic aspects of its design. We will then describe our architecture which is based on interfacing TAG with a rich functional theory provided by functional systemic grammar (Halliday, 1970; Halliday, 1985; Fawcett, 1980; Hudson, 1981).[2] We pay particular attention to those aspects which distinguish our generator from Mumble-86.

2 Mumble-86

Mumble-86 generates from a specification of what is to be said in the form of an "L-Spec"

[1] This work is supported in part by Grant #H133E80015 from the National Institute on Disability and Rehabilitation Research. Support was also provided by the Nemours Foundation. We would like to thank John Hughes for his many comments and discussions concerning this work.

[2] The particular suitability of TAG as a grammatical formalism to be used in conjunction with a systemic grammar is discussed in (McCoy, Vijay-Shanker & Yang, 1990).

(Linguistic Specification). An L-Spec captures the content of what is to be generated along with the goals and rhetorical force to be achieved. While the form of the L-Spec is dependent on the particular application, for the purposes of this discussion we can think of it as a set of logical form expressions that describe the content to be expressed. Mumble-86 uses a dictionary-like mechanism to transform a piece of the L-Spec into an elementary TAG tree which realizes that piece. The translation process itself (performed in the dictionary) may be influenced by contextual factors (including pragmatic factors which are recorded as a side-effect of grammar routines), and by the goals recorded in the L-Spec itself. It is in this way that the system can make fine-grained decisions concerning one realization over another.

Once a TAG tree is chosen to realize the initial subpiece, that structure is traversed in a left to right fashion. Grammar routines are run during this traversal to ensure grammaticality (e.g., subject-verb agreement) and to record contextual information to be used in the translation of the remaining pieces of the L-Spec. In addition to the grammar routines, as the initial tree is traversed at each place where new information could be added into the evolving surface structure (called attachment points), the remaining L-Spec is consulted to see if it contains an item whose realization could be adjoined or substituted at that position.

In order for this methodology to work, (McDonald & Pustejovsky, 1985) point out that they have to make some strong assumptions about the logical form input to their generator. Notice that the methodology described always starts generating from an *initial tree* and other auxiliary or initial trees are adjoined or substituted into that initial structure.[3] As a result, in generating an embedded sentence, the generator must start with the *innermost* clause in order to ensure that the first tree chosen is an initial (and not an auxiliary) tree. Consider, for example, the generation of the sentence "Who did you think hit John". Mumble-86 must start generating from the clause "Who hit John" which is (roughly) captured in the tree shown in Figure 4. This surface structure would then be traversed. At the point labeled *fr-node* (an attachment point) the auxiliary tree representing "you think" in Figure 2 would be adjoined in.

Notice, however, that if Mumble-86 must work from the inner-most clause out, then the initial L-Spec must be in a particular form which is not consistent with the "logician's usual representation of sentential complement verbs as higher operators" (McDonald & Pustejovsky, 1985)[p. 101] (also noted by (Shieber & Schabes, 1991)). Instead Mumble-86 requires an alternative logical form representation which amounts to breaking the more traditional logical form into smaller pieces which reference each other. Mumble-86 must be told which of these pieces is the embedded piece that the processing should start with.[4]

Notice that this architecture is particularly problematic for certain kinds of verbs that take indirect questions. For instance, it would preclude the proper generation of sentences involving "wonder" (as in "I wonder who hit John"). Verbs which require the question to remain embedded are problematic for Mumble-86 since the main verb (wonder) would not be available when its inclusion in the surface structure needs to be determined.[5]

An additional requirement on the logical form input to the generator is that the lambda expression (representing a wh-question) and the expression containing the matrix trace be present in a single layer of specification. This, they claim, is necessary to generate an appropriate sentence form without the necessity of looking arbitrarily deep into the representation. This would mean that for sentences such as "Who do you think hit John", the lambda expression would have to come with the "hit John" part of the input. We will show that our system does not place either of these restrictions on the logical form input and yet is able to generate the appropriate sentence without looking arbitrarily deep into the input specification.

One can notice a few features of the system just described. First, because the dictionary translation process is context sensitive, the generation methodology is able to take more than just logical form into account. Note, however, that it is unclear what the *theory* is behind the realizations made. In addition, these decisions are encoded procedurally thus the theory is rather difficult to abstract.

It is also the case that Mumble-86 makes no distinction between decisions that are made for functional reasons and those that are made for syntactic reasons. Both kinds of information must be recorded (procedurally) in grammar routines so that they can be taken into account during subsequent translations. While the fact that the grammar is procedurally encoded and that functional

[3]An initial tree is a minimal non-recursive structure in TAG, while an auxiliary tree is a minimal recursive structure. Thus, an auxiliary tree is characterized as having a leaf node (which is termed the foot node) which has the same label as the root node. The tree in Figure 2 is an auxiliary tree. The adjoining operation essentially inserts an auxiliary tree into another tree. For instance, the tree in Figure 5 is the result of adjoining the auxiliary tree shown in Figure 2 into the initial tree shown in Figure 4 at the node labeled *fr-node*.

[4]The task of ordering the elements of logical form is considered by Mumble-86 to be part of a component which is also responsible for ensuring that what is given to mumble is actually expressible in the language (e.g., English). This component is described in (Meteer, 1991).

[5]This is because the logical form for an embedded question and a non-embedded question cannot be distinguished in the kind of input required by Mumble-86 and the main verb (wonder) is not able to pass any information down to the embedded clause since it is realized after the embedded clause.

and syntactic decisions are mixed does not affect the power of the generator, we argue that it does make development and maintenance of the system rather difficult. Functional decisions (e.g., that a particular item should be made prominent) and syntactic decisions (e.g., number agreement) rely on two different bodies of work which should be able to evolve independently of each other. There is no separation of these two different influences in Mumble-86.

The generation process in Mumble-86 is *syntax driven*. From the input L-Spec an initial (elementary) TAG tree is chosen. This structure is then traversed and grammar routines are initiated. At each possible attachment point during the traversal, the semantic structure (L-Spec) is consulted to see if it contains an item whose realization could be adjoined or substituted at that position. Thus the syntactic surface structure drives the processing.

As a side effect of the above processing strategy, Mumble-86 creates a strictly left-to-right realization of surface structure. While this side-effect is deliberate for reasons of psychological validity, this can be problematic for generating some connectives (as is pointed out in (McKeown & Elhadad, 1991)). This is because Mumble-86 does not have access to the content of the items being conjoined at the time the connective is generated.

In the remainder of this paper we describe a sentence generation system which we have developed. In some ways it is similar to Mumble-86, but there are several major differences:

- The realization of the input in our system is based on systemic functional linguistics (Halliday, 1970; Halliday, 1985; Fawcett, 1980; Hudson, 1981). This is a linguistic theory which states that a generated sentence is obtained as a result of a series of functional choices which are made in a parallel fashion along several different functional domains. The choices are represented as a series of networks with traversal of the networks dependent on the given input along with several knowledge sources which encode information about how various concepts can be linguistically realized. The bulk of the work in systemic linguistics has been devoted to describing what/how functional choice affects surface form. We adopt this work from systemic linguistics, but unlike other implementations, we use a formal syntactic framework (TAG) to express the syntactic constraints.
- Our method is not syntax directed, but follows a functional decomposition called for by the systemic grammar.
- There is a clear separation between the functional and the syntactic aspects of sentence generation which actually allows these two aspects of generation to be developed independently.
- We do not place any constraints on the logical form input. Our methodology calls for nothing different from what is required for a standard systemic grammar (whose input is based on a typical logical form representation).
- The methodology which we describe allows sentence generation to proceed in a semantic head-driven fashion (Shieber, Van Noord, Pereira & Moore, 1990). This is the case even for the embedded sentences discussed earlier which had to be worked "inside out" in Mumble-86.

3 Generator Architecture

There are many different ways of implementing a TAG-based generator. We consider the principles that we take to be common to any TAG generator and indicate how these principles have influenced our architecture. We present various aspects of our architecture and contrast them with choices that have been made in Mumble-86 and Synchronous TAG. Our approach is motivated by arguments presented in (McCoy, Vijay-Shanker & Yang, 1990), but the details of the processing presented there have changed significantly. Our basic processing strategy is detailed in (Yang, McCoy & Vijay-Shanker, 1991); the work presented here is an extension of that strategy.

In order for a TAG generator to be robust, it must have a methodology for deciphering the input and associating various pieces of the input with TAG trees. In Mumble-86 this is accomplished through dictionary look-up along with querying the input at various points during the surface structure traversal. In contrast, we use a systemic grammar traversal for this purpose. In a TAG, each elementary tree lexicalizes a predicate and contains unexpanded nodes for the required arguments. Thus any TAG based generation system should incorporate the notions of semantic head-driven generation. Our approach, based on systemic grammars, does this because the functional decomposition that results from traversal of a systemic grammar at a single rank identifies the head and establishes necessary arguments. Thus it perfectly matches the information captured in an elementary TAG tree.

Once the input has been deciphered, a TAG generator must use this to select a tree. Given that a systemic grammar is being used in our case, we must have a method for associating TAG trees with the network traversal. The traversal of a systemic grammar at a single rank establishes a set of functional choices that can be used to select a TAG tree. The selection process in any TAG-based generator can be considered as providing a classification of TAG trees on functional grounds. We make this explicit by providing a network (called the TAG network)[6] which is traversed to select a TAG tree. The network itself can be thought of as

[6] In fact we view a systemic network in a similar fashion

$$\begin{bmatrix} s-act : wh-question \\ wh-it : n_1 \\ tense : past \\ proc : \text{"}think\text{"} \\ actor : n_2 : \begin{bmatrix} \text{"}you\text{"} \end{bmatrix} \\ phen : \begin{bmatrix} proc : \text{"}hit\text{"} \\ tense : past \\ actee : n_3 = \begin{bmatrix} \text{"}john\text{"} \end{bmatrix} \\ actor : n_1 = \begin{bmatrix} type : person \\ id : quest \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 1. Input for *Who did you think hit John*

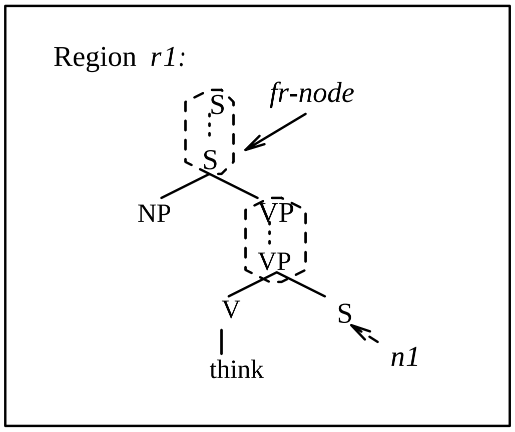

Figure 2. Initial tree selected in region r_1

a decision tree whose choice points are functional features chosen in the systemic network traversal.

So far we have identified how the head can be lexicalized and placed in an appropriate tree with respect to its arguments. This is accomplished by a traversal of a systemic network *at one rank* followed by a TAG network traversal based on the functional choices made. Of course, the arguments themselves must also be realized. This is accomplished by a recursive network (systemic followed by TAG) traversal (focused on the piece of input associated with the particular argument being realized). The recursive network traversals will also result in the realization of a TAG tree. We record information collected during a single (rank) network traversal in a data structure called a *region*. Thus, an initial region will be created and will record all features necessary for the selection of a tree realizing the head and argument placement. The selected tree (and other structures discussed below) will be recorded in the region. Each argument will itself be realized in a subregion which will be associated with the recursive network traversal spawned by the piece of input associated with that argument. Thus we have separate regions for each independent piece of input. This is in contrast to Mumble-86's use of the evolving surface structure in which all grammatical information is recorded.

Once all arguments have been realized as elementary trees in the individual regions, the trees selected in the individual regions must be combined with the tree in the initial region. For this we use the standard TAG operations of adjoining and substitution.

Essentially, our generation methodology consists of two phases:

1. The descent process – where a systemic network traversal is used to collect a set of features which are used to select a TAG tree that realizes the head and into which the arguments can be fit. The traversal is also responsible for spawning the creation of subregions in which the arguments (and modifiers) are realized.

2. The ascent process – where the trees created in the individual subregions are combined with the tree in the mother region resulting in the final realization of the whole.

as a classification of all functional choices expressible in a language.

In our system the systemic network traversal basically replaces the dictionary look-up phase found in Mumble-86[7] which translates the input L-Spec into surface structure. In addition, our system does not walk a surface structure (i.e., the actual tree chosen). In Mumble-86 the surface structure walk spawned grammar routines and caused additional pieces of the L-Spec to be translated into surface structure. Our methodology relies on the systemic network traversal to spawn realizations of the decomposed subpieces. The syntactic aspects of the grammar routines are now incorporated into our TAG network and grammar. Thus our methodology keeps a clearer separation between functional and syntactic aspects of the generation process.

The processing in our system will be explained with an example. Consider the simplified input given in Figure 1.[8] See (Yang, McCoy & Vijay-Shanker, 1991) for a more detailed description of the processing.

[7] The systemic grammar also replaces the grammar routines of Mumble-86 responsible for recording contextual information for subsequent translations. In addition, the part of the dictionary look-up concerned with *syntactic realization* (i.e., the actual tree chosen) is handled by our TAG component.

[8] This input is simplified in that it is basically a standard logical form input with lexical items specified. In general the input is a set of features which drive the traversal of the functional systemic networks.

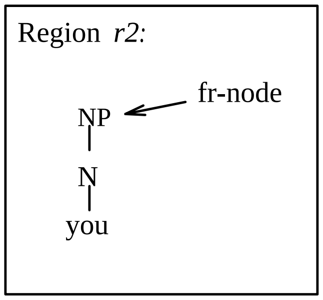

Figure 3. Tree selected in Actor region r_2

3.1 The Descent Process

The input given (along with other knowledge sources traditionally associated with a systemic network) will be used to drive the traversal of a functional systemic network. The purpose of this traversal is two fold: (1) to identify the head/argument structure of the sentence to be realized, and (2) to identify a set of functional features which can be used to choose a tree which appropriately realizes the head/argument structure.

Traditionally a systemic network consists of a number of networks of functional choices which are traversed in parallel. Each network considers choices along one functional domain. One such network is the *mood* network which is responsible for, among other things, determining what kind of speech act should be generated for the top-level element. This network must notice, for example, that the speech-act specified is wh-questioning, but that the item being questioned is not one of the arguments to the top level process. Thus a standard declarative form should be chosen for the realization of this top level element.

Standard implementations of systemic grammar (Davey, 1978; Mann & Matthiessen, 1985; Patten, 1988; Fawcett, 1990), upon traversal of the mood network to this point, would evaluate a set of realization operations which manipulate an eventual surface string. For instance, upon identifying that a declarative form is needed, the subject would be *ordered* before the finite. We argue in (McCoy, Vijay-Shanker & Yang, 1990) that it is more practical to replace the use of such realization operators with a more formal grammatical system (and that the use of such a system is perfectly consistent with the tenets of systemic linguistics). Thus during the network traversal, our system simply collects the chosen features and these are used to drive the traversal of a TAG network whose traversal results in the selection of a tree.

At the same time the mood network is traversed, so would be other networks. The *transitivity* network is concerned with identifying the head argument structure of the item being realized. In

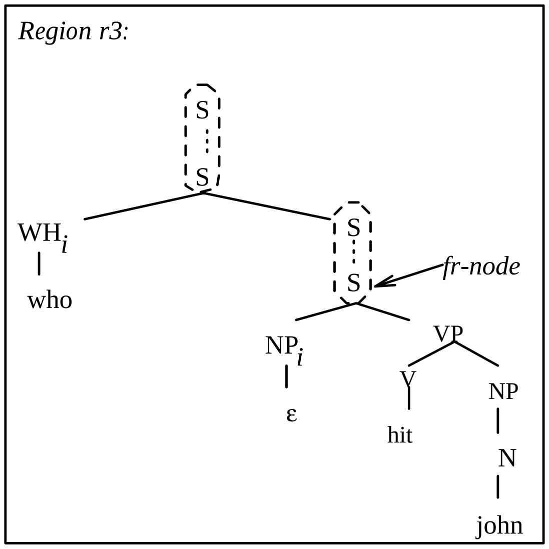

Figure 4. Tree selected in Phenomenon region r_3

this case, it would consider the fact that the item to be realized has a "process" which is mental. This identification results in the expectation of two arguments – an actor (doing the mental process) and a phenomenon (that thing the process is about). Each of these identified arguments must be realized individually. This is accomplished via the *preselect* operation.[9] This operation causes a recursive network traversal (whose results are recorded in a subregion) to be done focused on the input for the identified sub-element.

The features collected during the functional systemic network traversal are used to drive the traversal of the TAG network which results in the selection of a tree realizing the indicated features. Features such as that the process is mental and that the speech act is declarative would cause the selection of a tree for the mother region such as the tree in Figure 2.

Similar processing would then take place in the two subregions, each eventually resulting in the trees such as those shown in Figures 3 and 4.

3.2 The Ascent Process

In a TAG generator, after the input has been decomposed and elementary trees associated with each subpiece of the input, the chosen trees must be put together. Therefore, every TAG generator must provide a means to determine where

[9] From the realization operations used in systemic grammars (particularly Nigel), we need only the preselect and the conflate operations because all structure building operations are incorporated into TAG. The conflation operation is used to map functional features (e.g., agent, phenomenon) into grammatical functions (e.g., subject, complement). Note that in the networks from systemic grammars, we take only the functional part and thus avoid having choice points that exist for purely syntactic reasons.

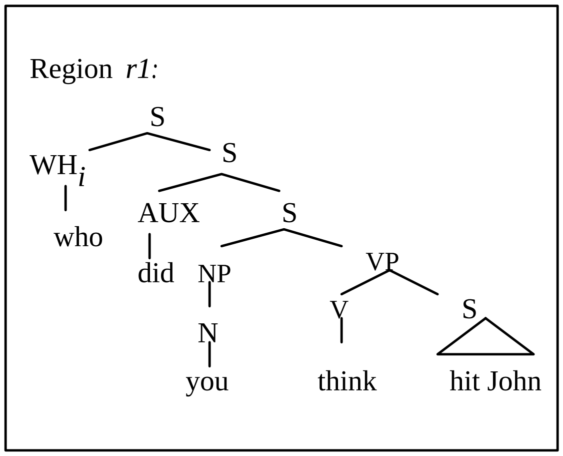

Figure 5: Final tree: *Who did you think hit John?*

the substitution or adjunction must take place. In order to do this, with each tree there must be a mapping of grammatical functions to nodes in the tree. In our case, we associate a *mapping table* with each tree. For instance, the mapping table associated with the tree shown in Figure 2 would indicate that the phenomenon (which would have been conflated with complement) is associated with the node labeled n_1 in the tree. In the simplest case the tree which realizes the phenomenon would be substituted at the node labeled n_1 in the tree in the mother region.

A data structure similar to a mapping table is used by the other TAG generators as well. In synchronous TAG the mapping table corresponds to the explicit node for node mapping between elementary logical form and syntactic trees. The mapping table in Mumble-86 is implicit in the schemas which create the surface structure tree (during the dictionary look-up phase) since they place L-spec elements in the appropriate place in the surface structure they create.

A more complex case arises when an argument node is a footnode of an auxiliary tree. Suppose an auxiliary tree, β, was chosen in a region and a tree, γ, was chosen in a subregion to realize the argument specified by the footnode of β. Rather than substituting γ in β, β is adjoined into a node in γ. This node is the node in γ that heads the subtree realizing the function specified for the subregion. For this reason, each tree in a region also has associated with it a pointer we call an *fr-node* which points to the node heading this subtree (functional root). In Regions r_1 and r_2 the functional root is also the root of the tree. Notice in Region r_3 that the functional root is the embedded S node. This *fr-node* is chosen because the tree chosen in the region is a wh-question tree due to the fact that (according to the input) the phenomenon is being questioned. There is nothing in the phenomenon itself, however, that specifies that

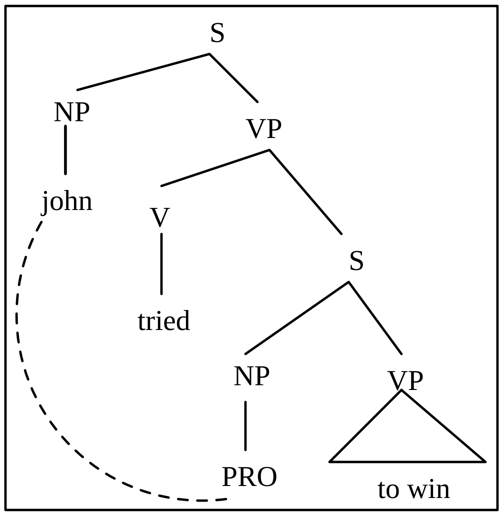

Figure 6. Standard tree for "John tried to win"

its speech-act should be wh-questioning. Thus the portion of the tree under the embedded S node captures the predicate argument structure which realizes the phenomenon as is specified in the input. If it were the case that the phenomenon was specified to be a wh-question (as in "Mary wondered who hit John") then the root node would be chosen as the *fr-node*. The *fr-node* comes into play when the trees in the individual regions are combined via adjunction during the ascent process.

Other TAG generators have analogues to our fr-node. In synchronous TAG it is implicit in the mapping between the nodes in the two kinds of trees. In Mumble-86, it is the attachment points on surface structure. The point is that if trees might be adjoined into, any TAG generator must specify where adjoining might take place and this specification depends (at least in part) on the functional content that the tree is intended to capture.

Going back to our example, in combining trees in the subregions with the tree chosen in the initial region $r1$, the agent tree would be combined with the tree in region $r1$ using straight substitution. The location of the substitution would be determined by the address given for the agent in the mapping table for the tree in region r_1.

The mapping table also indicates that the phenomenon should be placed at $n1$ in the tree in Figure 2. Notice, however, that $n1$ is the foot node. This is an indication to the processor that the final tree in region $r1$ should result from *adjoining* the tree in $r1$ into the tree in the subregion $r3$ (Figure 4). The place of adjoining is specified by the fr-node in the phenomenon tree in region r_3. The result of this adjoining is shown in Figure 5.[10]

[10]The details of how the AUX is inserted can be found in

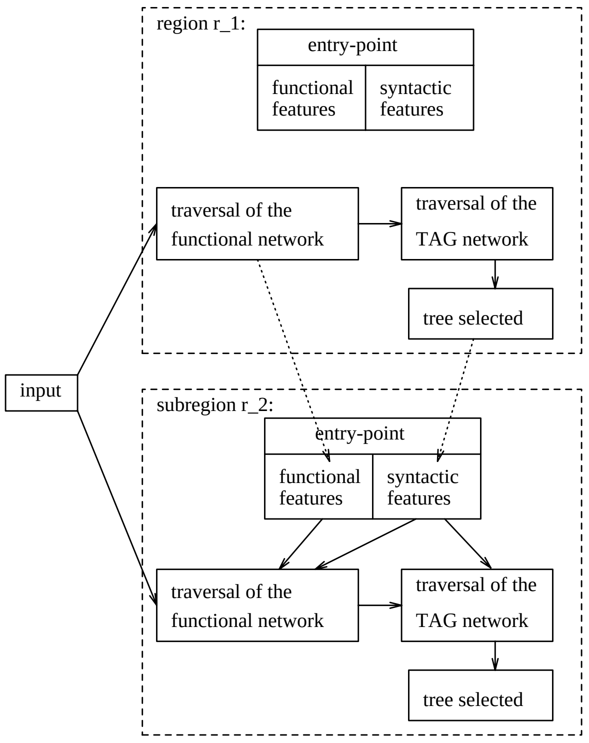

Figure 7: Flow of Information in Processing Model

4 Passing Features

So far we have established that any TAG-based generator, once an elementary tree has been chosen, would need to realize the arguments of the predicate by recursively calling the same procedure. The resulting trees chosen would be combined with the original elementary tree at the appropriate place by substitution and adjunction. In this recursive process, we have indicated the need for only functional information to be passed down from the mother region to the subregions (at the very least, in the form of the functional input associated with the piece being realized in the region). We now consider an example where syntactic information must be passed down as well.

Consider the generation of a sentence such as "John tried to win". The standard structure for this sentence is given in Figure 6. The problem is that in TAG this tree must be derived from the combination of two separate sentential trees: one headed by the verb "tried" and the other by the verb "win". However we must capture the constraint that the subject of the "win" tree is John (which is the same as the subject of the "tried" tree) but that it is realized only as a (null) pro. Note that this constraint cannot be localized in TAG but cuts across two elementary trees.

(Yang, 1991). It is inserted in the region r_1 as a result of a feature disparity on the nodes of the tree resulting from the adjoining operation just described. The same disparity would not occur in indirect questions (e.g., "I wonder who hit John").

While generating this sentence, when we choose the "tried" tree in the mother region, we must pass down the information that among the trees associated with win, the one with "pro" in the subject position must be chosen. Notice that this is a purely syntactic constraint based on the choice of the verb "try". The choosing of this tree has ramifications on both the functional network traversal (since the agent of "win" should not be expanded) and the TAG network traversal.

In addition, any syntactic constraint that is placed on the arguments (perhaps by the choice of the head) must be passed down to the subregion to influence the realization of the arguments. In general, the passed down features may influence either the functional or the TAG network traversal (see Figure 7). Such passing of syntactic and functional features must occur in any TAG generator where the realization of the head is done prior to the realization of its arguments.

5 Conclusions

In this paper we started with considering the principles underlying the design of any TAG-based generator. We have shown how these principles have been incorporated in our generation system and have compared it with other TAG-based generators.

The architecture of our generation system incorporates both functional aspects of generation and syntactic aspects. Each of these aspects is handled separately, by two different formalisms which are uniquely combined in our architecture. The result is a sentence generation system which has the advantage of incorporating two bodies of knowledge into one system. Our system has several advantages over Mumble-86. In addition to the use of systemic grammar as a theory for realization and a function (rather than syntactic) directed generation process, we have shown that our methodology does not place any special requirements on the input logical form. Our methodology can proceed in a head-driven manner using notions such as the mapping table and the functional root to decide how trees should be combined. These notions allow fine distinctions in form which are not possible in Mumble-86. In addition, our system separates functional from syntactic decisions thus allowing these two bodies to be expanded independently.

A prototype of our system has been implemented in Lucid Common Lisp on a Sun Workstation. Details of the implementation can be found in (Yang, 1991).

References

Conklin, E. & McDonald, D. (1982). Salience: The key to the selection problem in natural language generation. In *Proceedings of* *the 20th Annual Meeting*, (pp. 129–135)., Toronto, Canada. Association for Computational Linguistics.

Davey, A. (1978). *Discourse Production.* Edinburgh: Edinburgh University Press.

Fawcett, R. (1980). *Cognitive linguistics and social interaction.* Heidelberg: Julius Groos Verlag Heidelberg and Exeter University.

Fawcett, R. P. (1990). The communal project: two years old and going well. *Network*, (13).

Halliday, M. A. K. (1970). Language structure and language function. In J. Lyons (Ed.), *New Horizons in Linguistics.* Harmondsworth, England: Penguins Books.

Halliday, M. A. K. (1985). *An introduction to functional grammar.* London England: Edward Arnold.

Hudson, R. A. (1981). Systemic generative grammar. In M. A. K. Halliday & J. R. Martin (Eds.), *Readings in Systemic Linguistics.* North Pomfret, Vermont: Batsford.

Joshi, A. K. (1985). How much context-sensitivity is necessary for characterizing structural descriptions : Tree adjoining grammar. In D. Dowty, L. Karttunen, & A. Zwicky (Eds.), *Natural Language Processing : Theoretical, Computational and Psychological Perspectives.* New York: Cambridge University Press.

Joshi, A. K. (1987). The relevance of tree adjoining grammar to generation. In G. Kempen (Ed.), *Natural Language Generation: New Results in Artificial Intelligence, Psychology, and Linguistics* (pp. 233–252). Dordrecht/Boston: Martinus Nijhoff Publishers (Kluwer Academic Publishers).

Mann, W. & Matthiessen, C. (1985). Nigel: A systemic grammar for text generation. In O. Freedle (Ed.), *Systemic Perspectives on Discourse.* NJ: Norwood.

McCoy, K. F. (1989). Generating context sensitive responses to object-related misconceptions. *Artificial Intelligence*, *41*, 157–195.

McCoy, K. F., Vijay-Shanker, K., & Yang, G. (1990). Using tree adjoining grammars in the systemic framework. In *Proceedings of* 5^{th} *International Workshop on Natural Language Generation.*, Dawson, PA.

McDonald, D. (1983). Dependency directed control: Its implications for natural language generation. In N. Cercone (Ed.), *Computational Linguistics* (pp. 111–130). Pergamon Press.

McDonald, D. & Pustejovsky, J. D. (1985). Tags as a formalism for generation. In *Proceedings of the 23rd Annual Meeting*, Chicago, IL. Association for Computational Linguistics.

McKeown, K. R. & Elhadad, M. (1991). A contrastive evaluation of functional unification grammar for surface language generation: A case study in choice of connectives. In C. Paris, W. Swartout, & W. Mann (Eds.), *Natural Language Generation in Artificial Intelligence and Linguistics* (pp. 351–396). Boston/Dordrecht/London: Kluwer Academic Publishers.

Meteer, M. (1991). Bridging the 'generation gap'. *Computational Intelligence*, *7*(4).

Meteer et al., M. (1987). Mumble-86: Design and implementation. COINS Tech Report 87-87a, University of Massachusetts.

Patten, T. (1988). *Systemic Text Generation as Problem Solving.* Cambridge: Cambridge University Press.

Rubinoff, R. (1986). Adapting mumble: Experience with natural language generation. In *Proceedings of the 1986 National Conference on Artificial Intelligence*, (pp. 1063–1068)., Philadelphia, Pa. AAAI.

Schabes, Y., Abeille, A., & Joshi, A. (1988). Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of COLING' 88*, Budapest, Hungary.

Shieber, S. M. & Schabes, Y. (1991). Generation and synchronous tree-adjoining grammars. *Computational Intelligence*, *7*(4).

Shieber, S. M., Van Noord, G., Pereira, F., & Moore, R. C. (1990). Semantic-head-driven generation. *Computational Linguistics*, *16*(1).

Woolf, B. & McDonald, D. (1984). Context-dependent transitions in tutoring discourse. In *Proceedings of the 1984 National Conference on Artificial Intelligence*, Washington, D.C. AAAI.

Yang, G. (1991). *An Integrated Approach to Generation Using Systemic Grammars and Tree Adjoining Grammars.* PhD thesis, University of Delaware.

Yang, G., McCoy, K. F., & Vijay-Shanker, K. (1991). From functional specification to syntactic structures: Systemic grammar and tree adjoining grammar. *Computational Intelligence*, *7*(4).