Two Recent Developments in Tree Adjoining Grammars: Semantics and Efficient Processing*

Yves Schabes
Aravind K. Joshi

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

ABSTRACT

During the past year there have been two very significant developments in the area of Tree Adjoining Grammars (TAGs).

The first development is a variant of TAGs, called synchronous TAGs, which allows TAG to be used beyond the confines of syntax by characterizing correspondences between languages. The formalism's intended usage is to relate expressions of natural languages to their associated semantics represented by a logical form language in TAG, or to their translates in another natural language. The formalism is incremental and inherently nondirectional. We will show by detailed examples the working of synchronous TAGs and some of its applications, for example in generation and in machine translation.

The second development is the design of LR-style parsers for TAGs. LR parsing strategies evolved out of the original work of Knuth. Even though they are not powerful enough for NLP, they have found use in natural language processing (NLP) by solving by pseudo-parallelism conflicts between multiple choices. This gives rise to a class of powerful yet efficient parsers for natural language. In order to extend the LR techniques to TAGs it is necessary to find bottom-up automaton that is exactly equivalent to TAGs. This is precisely what has been achieved by the discovery of the Bottom-up Embedded Push Down Automaton (BEPDA). Using BEPDA, deterministic left to right parsers for the Tree Adjoining Languages have been developed.

Using TAGs beyond their Role in Syntax

The unique properties of tree-adjoining grammars (TAG) present a challenge for the application of TAGs beyond the limited confines of syntax, for instance, to the task of semantic interpretation or automatic translation of natural language. A variant of TAGs, called synchronous TAGs, has been developed (Shieber and Schabes [1990(a)]). It is used to relate expressions of natural languages to their associated semantics represented in a logical form language or to their translates in another natural language (the work on Synchronous TAG and its applications to language interpretation and generation has been done in collaboration with Stuart Shieber).

Language interpretation and generation with TAGs

The key idea for semantic interpretation is that the logical form language itself can be described by a TAG. The two TAGs (one for the natural language and one for the logical form language) work synchronously, in the sense that the certain correspondences (links) are stated initially between the elementary trees of the two TAGs and then composition operations (such as substitution and adjoining) are carried out synchronously on the linked nodes of the two TAGs. The fact that both the natural language and the logical form language can be described by TAGs is a direct consequence of the extended domain of locality of TAGs as compared to LFG or GPSG.

A sample synchronous TAG is given in Figure 1. Each element of the synchronous TAG is a pair consisting of two elementary trees, one from the source language (English) and one from the target (logical form [LF]). Nodes, one from each tree, may be linked; such links are depicted graphically as thick lines. If we project the pairs onto their first or second components (ignoring the cross links), the projections are TAGs for an English fragment and an LF fragment, respectively. These grammars are themselves written in a particular variant of TAGs; the choice of this base formalism, as we will call it, is free. In the case at hand, we have chosen single-component lexicalized TAGs with adjunction and substitution (Schabes, Abeillé and Joshi [1988]). Other bases (as Multiple Component TAGs) are needed for more complex phenomena.

The elementary operation in a synchronous TAG is supervenient on the elementary operations in the base formalism. A derivation step from a pair of trees $\langle \alpha_1, \alpha_2 \rangle$ proceeds as follows:

1. Nondeterministically choose a link in the pair connecting two nodes (say, n_1 in α_1 and n_2 in α_2).

*This research is partially supported by Darpa grant N0014-85-K0018, ARO grant DAAL03-89-C-0031PRI and NSF grant-IRI84-10413 A02. Parts of this work are results of collaboration with Stuart Shieber (Harvard University), K. Vijay-Shanker (University of Delaware) and Anne Abeillé (University of Paris-7). The specifics of the collaboration are stated in the body of the paper. We are also grateful to Bernard Lang and David Weir for their valuable suggestions on LR-style parsing of TAGs which played an instrumental role in the definition of BEPDA, for example restriction on the moves allowed.

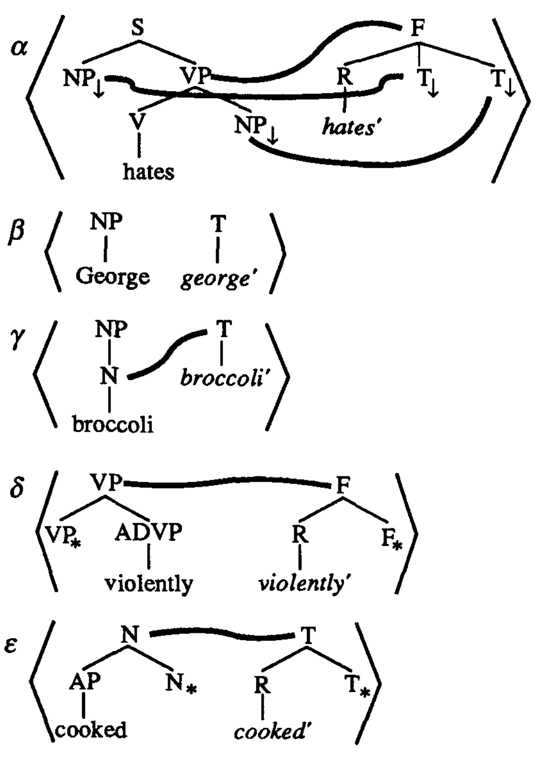
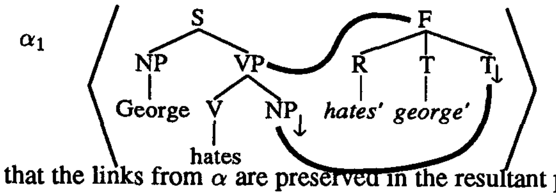


Figure 1: A sample synchronous TAG.

2. Nondeterministically choose a pair of trees $\langle\beta_1, \beta_2\rangle$ in the grammar.
3. Form the resultant pair $\langle\beta_1(\alpha_1, n_1), \beta_2(\alpha_2, n_2)\rangle$ where $\beta(\alpha, n)$ is the result of performing a primitive operation in the base formalism on α at node n using β (e.g., adjoining or substituting β into α at n).[1]

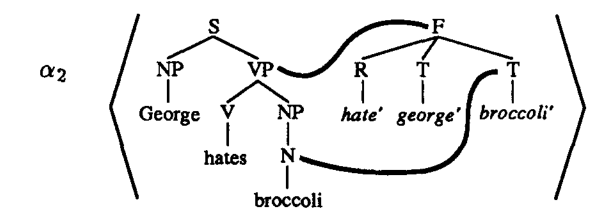
Synchronous TAG derivation then proceeds by choosing a pair of initial trees $\langle\alpha_1, \alpha_2\rangle$ that is an element of the grammar, and repeatedly applying derivation steps as above.

As an example, suppose we start with the tree pair α in Figure 1.[2] We choose the link from the subject NP to T and the tree pair β to apply to its nodes. The resultant, by synchronous substitution, is the tree pair:



Note that the links from α are preserved in the resultant pair α_1 except for the chosen link, which has no counterpart in the result.

Using tree pair γ on the remaining link from NP to T in α_1 yields



This pairing manifests the correspondence between the sentence "George hates broccoli" and its logical form $hates'(george', broccoli')$ (as written in a more traditional notation). Here we see that the links in the operator trees (those in γ) are preserved in the resultant pair, accounting for the sole remaining link. The trees in γ are linked in this way so that other tree pairs can modify the N.

We can continue the derivation, using δ and ϵ to generate the pair given in Figure 2 thereby associating the meaning

$$violently'(hates'(george', cooked'(broccoli')))$$

with the sentence "George hates cooked broccoli violently."

The arguments for factoring recursion and dependencies as TAGs do for the syntax of natural language have their counterparts in the semantics. The structure of TAGs allows syntactic dependencies—agreement, subcategorization, and so forth—to be localized in the primitives of a grammar, the elementary trees. This is most dramatically evident in the case of long-distance dependencies, such as that between a wh-phrase and its associated gap. Similarly, using TAGs to construct logical forms allows the localization of semantic dependencies in the logical forms of natural language expressions, dependencies such as the signature requirements (argument type and arity) of function and relation symbols, and even the long-distance dependencies between a wh-quantifier and its associated bound variable. With other methods of semantics, these dependencies cannot be localized; the semantic aspects of filler-gap dependencies must be passed among the features of various nodes in a parse tree or otherwise distributed over the entire derivation.

The use of the synchronous TAG augmentation allows an even more radical reduction in the role of features in a TAG grammar. Because of the extended domain of locality that TAGs possess, the role of features and unification is reduced from its role in context-free based systems. Only finite-valued features are needed, with the possible exception of a feature whose value encodes an expression's logical form. In removing the construction of logical forms from the duties delegated to features, we can maintain a strictly finite-valued—and therefore formally dispensable—feature system for TAGs.

Applications

Synchronous TAGs suggest elegant solutions to the semantics of idioms, quantifier scoping (Shieber and Schabes, [1990a]) and provide an elegant framework for generation (Shieber and Schabes, [1990b]) and machine translation (Abeillé, Schabes and Joshi [1990]).

[1] The definition allows for the operations performed on the first and second trees to differ, one being a substitution and the other an adjunction, for example.

[2] We use standard TAG notation, marking foot nodes in auxiliary trees with '*' and nodes where substitution is to occur with '↓'. The nonterminal names in the logical form grammar are mnemonic for Formula, Relation (or function) symbol, Term, and Quantifier.

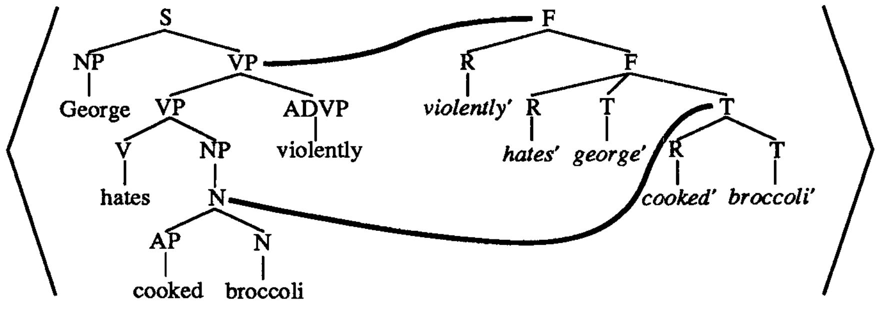
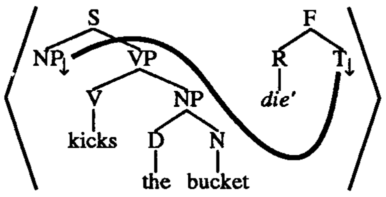


Figure 2: Derived tree pair for "George hates cooked broccoli violently."

Semantics Idioms

All of the arguments for the TAG analysis of idioms and light verb constructions (Abeillé and Schabes, 1989) can then be maintained in a formalism that allows for semantics for them as well. In particular, discontinuous syntactic constituents can be semantically localized nonstandard long-distance dependencies are statable without resort to reanalysis, both frozen and flexible idioms can be easily characterized.

For example, the idiomatic construction "kick the bucket" cashes out as the following tree pair, under its idiomatic interpretation:

α_3 

whereas the literal usage of "kick" is associated with a tree pair similar to that of "hates" in Figure 1.

Quantifiers

In order to characterize quantifier scoping possibilities, multi-component TAGs (as defined by Joshi, 1987) is used as the base formalism for synchronous TAG (see Shieber and Schabes [1990(a)] for more details on quantifiers scoping with Synchronous TAG). In particular, an NP will be linked both to a formula in the semantics (the quantifier's scope) and a term (the position bound by the quantifier).

Generation

The nondirectionaly of Synchronous TAGs enables us to use it for semantic interpretation as well as for generation (see Shieber and Schabes [1990b]).

Machine Translation

The transfer between two languages, such as French and English, can be done by putting directly into correspondence large elementary units without going through some interlingual representation and without major changes to the source and target grammars (Abeillé, Schabes and Joshi [1990]). The underlying formalism for the transfer is Synchronous Tree Adjoining Grammars. Transfer rules are stated as correspondences between nodes of trees of large domain of locality which are associated with words. We can thus define lexical transfer rules that avoid the defects of a mere word-to-word approach but still benefit from the simplicity and elegance of a lexical approach (this work has been done in collaboration with Anne Abeillé).

As an example, consider the fragment of the transfer lexicon given in Figure 3.

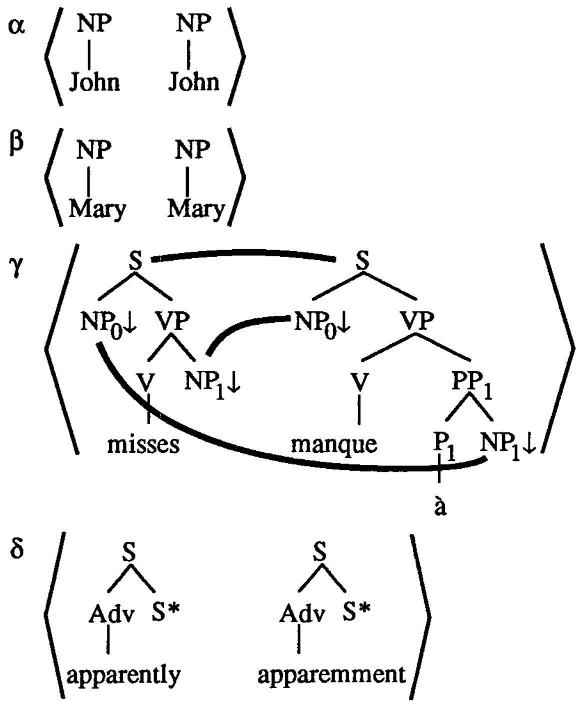
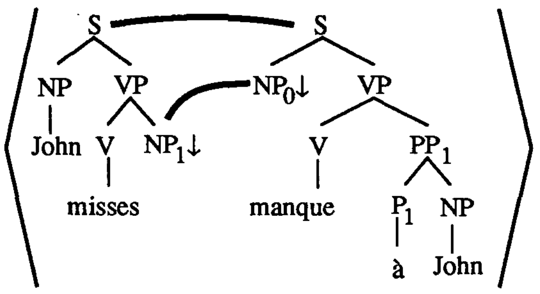


Figure 3: Fragment of the English-French transfer lexicon

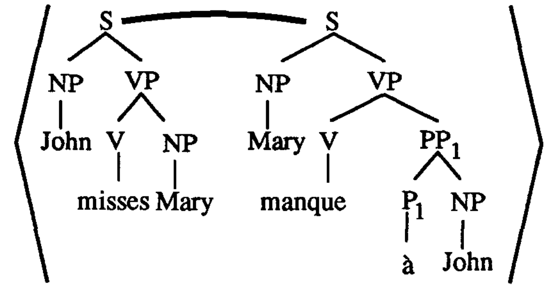
For example, suppose we start with the pair γ and we operate the pair α on the link from the English node NP_0 to the French node NP_1. This operation yields the derived pair α_4.

α_4



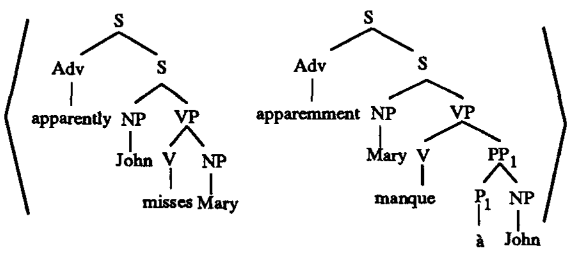
Then, if the pair β operates on the NP_1-NP_0 in α_4, the following pair α_5 is generated.

α_5



Finally, when the pair δ operates on the S-S link in α_5, the pair α_6 is generated.

α_6



The fragment of the transfer lexicon given in Figure 3 therefore enables us to translate:

Apparently, John misses Mary
$\leftrightarrow$ *Apparemment, Mary manque à John*

In most cases, translation can be performed incrementally as the input string is being parsed.

By virtue of their extended domain of locality, Tree Adjoining Grammars allow regular correspondences between larger structures to be stated without a mediating interlingual representation. The mapping of derivation trees from source to target languages, using the formalism of synchronous TAGs, makes possible to state such direct correspondences. By doing so, we are able to match linguistic units with quite different internal structures. Furthermore, the fact that the grammars are lexicalized enables capturing some idiosyncrasies of each language.

The simplicity and effectiveness of the transfer rules in this approach shows that lexicalized TAGs, with their extended domain of locality, are very well adapted to machine translation.

Efficient Processing of TAGs

The second development is the design of LR-style parsers for TAGs. LR parsing strategies evolved out of the original work of Knuth. LR(k) parsers for Context Free Grammars (Knuth, 1965) consist of a finite state control (constructed given a CFG) that drives deterministically with k lookahead symbols a push down stack, while scanning the input from left to right. It has been shown that they recognize exactly the set of languages recognized by deterministic push down automata. LR(k) parsers for CFGs have been proven useful for compilers as well as recently for natural language processing. For natural language processing, although LR(k) parsers are not powerful enough, conflicts between multiple choices are solved by pseudo-parallelism (Lang, 1974, Tomita, 1987). This gives rise to a class of powerful yet efficient parsers for natural languages. It is in this context that deterministic (LR(k)-style) parsing of TAGs is studied (this work has been done in collaboration with Vijay-Shanker).

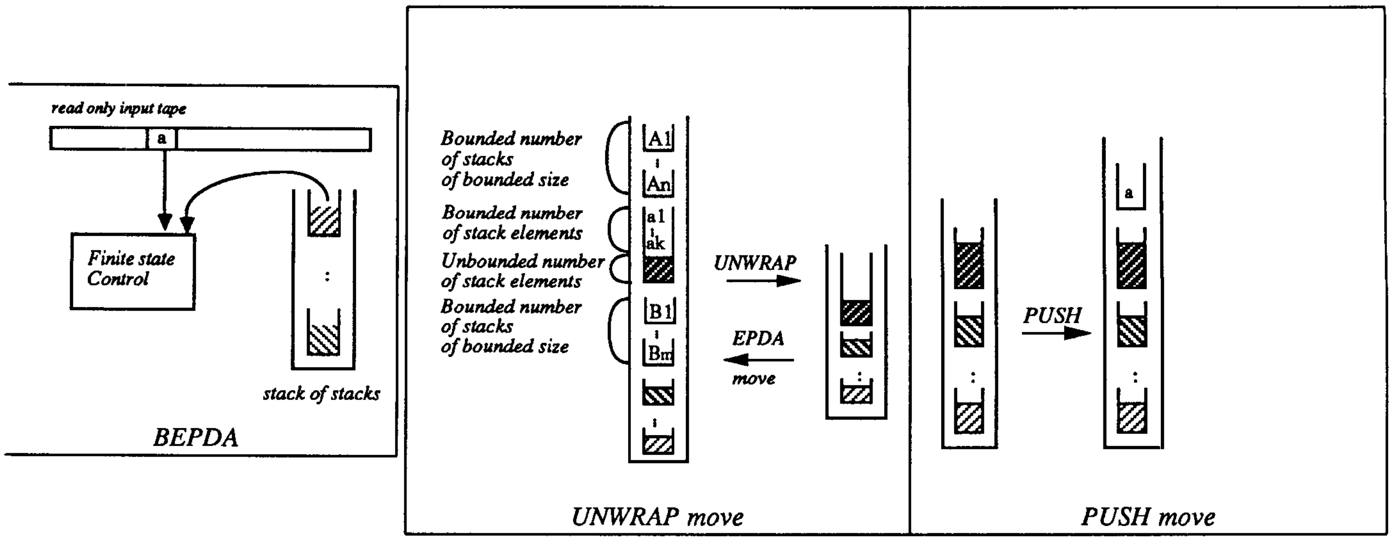
The set of Tree Adjoining Languages is a strict superset of the set of Context Free Languages (CFLs). For example, the cross serial dependency construction in Dutch can be generated by a TAG. Walters (1970), Révész (1971), Turnbull and Lee (1979) investigated deterministic parsing of the class of context-sensitive languages. However they used Turing machines which recognize languages much more powerful than Tree Adjoining Languages. So far no deterministic bottom-up parser has been proposed for any member of the class of the so-called "mildly context sensitive" formalisms (Joshi, 1985) in which Tree Adjoining Grammars fall.[3] Since the set of Tree Adjoining Languages (TALs) is a strict superset of the set of Context Free Languages, in order to define LR-type parsers for TAGs, we need to use a more powerful configuration then a finite state automaton driving a push down stack. The design of deterministic left to right bottom up parsers for TAGs in which a finite state control drives the moves of a Bottom-up Embedded Push Down Stack has been investigated. The class of corresponding non-deterministic automata recognizes exactly the set of TALs.

Due to the lack of space, we focus our attention on the bottom-up embedded pushdown automaton. The moves of the parser are sequences of moves of the automaton. The complete construction of LR-style parser for TAGs can be found in Schabes and Vijay-Shanker (1990).

Automata Models of Tags

Before we discuss the Bottom-up Embedded Pushdown Automaton (BEPDA) which is used by parser, we will explain the Embedded Pushdown Automaton (EPDA). An EPDA is similar to a pushdown automaton (PDA) except that the storage of an EPDA is a sequence of pushdown stores. A move of an EPDA (see Figure 5) allows for the introduction of bounded pushdowns above and below the current top pushdown. Informally, this move can be thought of as corresponding to the adjoining operation move in TAGs with the

[3]Tree Adjoining Grammars, Modified Head Grammars, Linear Indexed Grammars and Categorial Grammars (all of which generate the same subclass of context-sensitive languages) fall in the class of the so-called "mildly context sensitive" formalisms. The Embedded Push Down Automaton recognizes exactly this set of languages (Vijay-Shanker 1987).



read only input tape
a
Finite state Control
stack of stacks
BEPDA
Bounded number of stacks of bounded size
Bounded number of stack elements
Unbounded number of stack elements
Bounded number of stacks of bounded size
A1
An
a1
ak
B1
Bm
a
UNWRAP
EPDA move
PUSH
UNWRAP move
PUSH move

Figure 4: Bottom-up Embedded Pushdown Automaton

pushdowns introduced above and below the current pushdown reflecting the tree structure to the left and right of the foot node of an auxiliary being adjoined. The spine (path from root to foot node) is left on the previous stack.

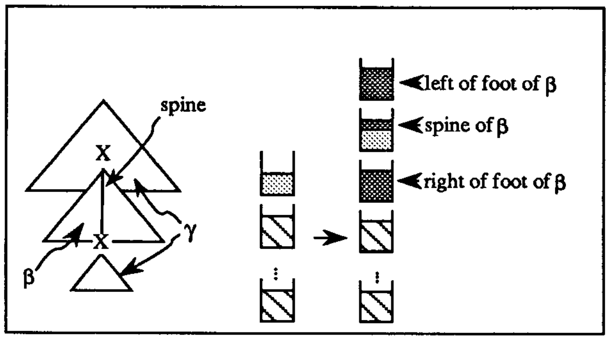


Figure 5: Embedded Pushdown Automaton

The generalization of a PDA to an EPDA whose storage is a sequence of pushdowns captures the generalization of the nature of the derived trees of a CFG to the nature of derived trees of a TAG. From Thatcher (1971), we can observe that the path set of a CFG (i.e. the set of all paths from root to leaves in trees derived by a CFG) is a regular set. On the other hand, the path set of a TAG is a CFL. This follows from the nature of the adjoining operation of TAGs, which suggests stacking along the path from root to a leaf. For example, as we traverse down a path in a tree γ (in Figure 5), if adjunction, say by β, occurs then the spine of β has to be traversed before we can resume the path in γ.

Bottom-up Embedded Pushdown Automaton

For any TAG G, an EPDA can be designed such that its moves correspond to a top-down parse of a string generated by G (EPDA characterizes exactly the set of Tree Adjoining Languages, Vijay- Shanker, 1987). If we wish to design a bottom-up parser, say by adopting a shift reduce parsing strategy, we have to consider the nature of a reduce move of such a parser (i.e. using EPDA storage). This reduce move, for example applied after completely considering an auxiliary tree, must be allowed to 'remove' some bounded pushdowns above and below some (not necessarily bounded) pushdown. Thus (see Figure 4), the reduce move is like the dual of the wrapping move performed by an EPDA.

Therefore, the Bottom-up Embedded Pushdown Automaton (BEPDA), whose moves are dual of an EPDA, has been introduced. The two moves of a BEPDA are the unwrap move depicted in Figure 4 – which is an inverse of the wrap move of an EPDA – and the introduction of new pushdowns on top of the previous pushdown (push move). In an EPDA, when the top pushdown is emptied, the next pushdown automatically becomes the new top pushdown. The inverse of this step is to allow for the introduction of new pushdowns above the previous top pushdown. These are the two moves allowed in a BEPDA, the various steps in our parsers are sequences of one or more such moves.

Due to space constraints, we do not show the equivalence between BEPDA and EPDA apart from noting that the moves of the two machines are dual of each other.

Using the BEPDA, the parser recognizes the derived tree inside out: it extracts recursively the innermost auxiliary tree that has no adjunction performed in it. Schabes and Vijay-Shanker (1990) give a complete explanation of the parser moves and its construction. The accuracy of the parsing table can also be improved by computing lookaheads for TAGs.

Similar to the work of Lang (1974) and Tomita (1987) extending LR parsers for arbitrary CFGs, the LR parsers for TAGs can be extended to solve by pseudo-parallelism the conflicts of moves.

Conclusion

During the past year there have been two very significant developments in the area of Tree Adjoining Grammars (TAGs): synchronous TAGs and efficient processing of TAGs.

A variant of TAGs called Synchronous TAGs has been developed, which is used to relate expressions of natural languages to their associated semantics represented in a logical form language. The key idea is that the logical form language itself can be described by a TAG. The two TAGs work synchronously, in the sense that the certain correspondences (links) are stated initially between the elementary trees of the two TAGs and then universal composition operations (such as substitution and adjoining) are carried out synchronously on the linked nodes of the two TAGs. Synchronous TAGs are used for language interpretation, generation and machine translation.

The second development is the design of LR-style parsers for TAGs. The existence of the push down automata for context-free grammars is crucial for the development of these techniques for the parsing of context-free languages. In order to extend the LR techniques to TAGs it is necessary to find bottom-up automaton that is exactly equivalent to TAGs. This is precisely what has been achieved by the discovery of the Bottom-up Embedded Push Down Automaton (BPDA). Using BPDA the first deterministic left to right parsers for the Tree Adjoining Languages were developed.

References

Abeillé, Anne and Schabes, Yves, 1989. Parsing Idioms in Tree Adjoining Grammars. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics (EACL'89)*. Manchester.

Abeillé, Anne, Schabes, Yves, and Joshi, Aravind K., 1990. Using Lexicalized Tree Adjoining Grammars for Machine Translation. In *Proceedings of the* 13th *International Conference on Computational Linguistics (COLING'90)*. Helsinki.

Joshi, Aravind K., 1985. How Much Context-Sensitivity is Necessary for Characterizing Structural Descriptions—Tree Adjoining Grammars. In Dowty, D., Karttunen, L., and Zwicky, A. (editors), *Natural Language Processing—Theoretical, Computational and Psychological Perspectives*. Cambridge University Press, New York. Originally presented in a Workshop on Natural Language Parsing at Ohio State University, Columbus, Ohio, May 1983.

Joshi, Aravind K., 1987. An Introduction to Tree Adjoining Grammars. In Manaster-Ramer, A. (editor), *Mathematics of Language*. John Benjamins, Amsterdam.

Knuth, D. E., 1965. On the translation of languages from left to right. *Inf. Control* 8:607–639.

Lang, Bernard, 1974. Deterministic Techniques for Efficient Non-Deterministic Parsers. In Loeckx, Jacques (editor), *Automata, Languages and Programming, 2nd Colloquium, University of Saarbrücken*. Lecture Notes in Computer Science, Springer Verlag.

Révész, G., 1971. Unilateral context sensitive grammars and left to right parsing. *J. Comput. System Sci.* 5:337–352.

Schabes, Yves and Vijay-Shanker, K., 1990. Deterministic Left to Right Parsing of Tree Adjoining Languages. In 28th *Meeting of the Association for Computational Linguistics (ACL'90)*. Pittsburgh.

Schabes, Yves, Abeillé, Anne, and Joshi, Aravind K., August 1988. Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars. In *Proceedings of the* 12th *International Conference on Computational Linguistics (COLING'88)*. Budapest, Hungary.

Shieber, Stuart and Schabes, Yves, 1990 (a). Synchronous Tree Adjoining Grammars. In *Proceedings of the* 13th *International Conference on Computational Linguistics (COLING'90)*. Helsinki.

Shieber, Stuart and Schabes, Yves, 1990 (b). Generation and Synchronous Tree Adjoining Grammars. In *Proceedings of the fifth International Workshop on Natural Language Generation*. Pittsburgh.

Thatcher, J. W., 1971. Characterizing Derivations Trees of Context Free Grammars through a Generalization of Finite Automata Theory. *J. Comput. Syst. Sci.* 5:365–396.

Tomita, Masaru, 1987. An Efficient Augmented-Context-Free Parsing Algorithm. *Computational Linguistics* 13:31–46.

Turnbull, C. J. M. and Lee, E. S., 1979. Generalized Deterministic Left to Right Parsing. *Acta Informatica* 12:187–207.

Vijay-Shanker, K., 1987. *A Study of Tree Adjoining Grammars*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania.

Walters, D.A., 1970. Deterministic Context-Sensitive Languages. *Inf. Control* 17:14–40.