Constraining Tree Adjoining Grammars by Unification

Karin Harbusch

Deutsches Forschungszentrum für Künstliche Intelligenz
Stuhlsatzenhausweg 3, 6600 Saarbrücken 11
F.R.G.
Phone: (+49 681) 302 5271, Fax: (+49 681) 302 4261
E-Mail: harbusch@dfki.uni-sb.de

Abstract

In a proposal, Vijay-Shanker and Joshi presented a definition for combining the two formalisms Tree Adjoining Grammars and PATR unification. The essential idea for that combination is the separation of the two recursion operations - adjoining and unification - to preserve all properties of both formalisms which is not desirable for natural language applications. In this paper, a definition for the integrated use of both processes is given and the remaining properties of the resulting formalism are discussed - especially weighing the appropriateness of this definition for natural language processing.

1 Introduction

In the field of natural language analysis, *Unification Grammars* are a main research topic. Presently, Unification is defined as extension of context-free grammars.

Knowing the formalism of *Tree Adjoining Grammars* (in the following called *TAGs* in short), which is closely related to context-free grammars (in the following abbreviated *CFG*), the idea arises to replace the context-free grammar in a Unification Grammar by a TAG. The advantage of TAGs is that complete context-free derivation trees or parts of them build the rules of that grammar type (e.g., with the intention of representing a whole linguistic phenomenon). The recursion operation for TAGs allows the replacement of nodes by a tree (defined by a TAG-rule), so that larger structure trees are processed.

In the literature, a first definition for combining these two formalisms was proposed, where the main idea is to separate the two recursion processes - *adjoining* and *unification* - to preserve all properties of both. Here a different approach is chosen, where both recursion processes are integrated. The main point to emphasize here is that the different approaches not only represent a switch between two modes of interpreting the same definition, but a change in the properties of the resulting formalism.

This can be simply demonstrated by the property of monotonicity of the unification. Associated with each tree of a TAG all specification rules for the unification are interpreted at once (e.g., represented as links between the DAG representation of the specification rules at each node in the tree). If now the recursion process of TAGs, the adjoining operation, combines two trees, which both have DAGs, a strategy for reinterpretation of specification information must be defined, because an adjoining modifies inner nodes, where links still are installed.

In the following, the two formalisms are briefly revisited to have a common terminological basis with the reader, before the existing definition of *Tree Adjoining Grammars with Unification* is presented. In contrast to this approach, the new definition is motivated and its properties are discussed. Finally, our experience with an implementation of that definition in the application domain of syntax analysis (e.g., in a natural language dialogue system) is summarized.

2 The Two Basic Formalisms TAG and PATR

In this section, the formalism of TAGs is motivated to be appropriate to replace a context-free grammar in natural language description. Weighing the disadvantages remaining for TAGs, which are the same as for CFGs, the same extending formalism as for CFGs - Unification Grammars - has been chosen to resolve these disadvantages for TAGs.

2.1 A Short Outline of Tree Adjoining Grammars

In 1975, the formalism of Tree Adjoining Grammars (TAGs) was introduced by *Aravind K. Joshi, Leon S. Levy* and *Masako Takahashi* (see [Joshi et al. 75]). Since then, a wide variety of properties - formal properties as well as linguistically relevant ones - were studied (see, e.g., [Joshi 85] for a good overview).

The following example describing the crossed dependencies in Dutch should illustrate the formalism (see Figure 1, where the node numbers written in slanted font should be ignored here, they make sense in combination with Figure 3). A TAG is a tree generation system. It consists of two different sets of trees, which are combinable. Intuitively, the set of *initial trees* can be seen as context-free derivation trees. This means, the start symbol is the root node, all inner nodes are nonterminals and all leaves are terminals (e.g., in Figure 1 tree α). The second set, the *auxiliary trees*, which can replace a node in an initial tree (which is possibly modified by further adjoinings) during the process of adjoining, must have a form that again a derivation tree results. The trees β_1 and β_2 demonstrate that restriction. A special leaf (the *foot node*) must exist, labelled with the same nonterminal as the root node is labelled with. Further, it is obligatory that an auxiliary tree derives at least one terminal. The union of the initial and the auxiliary trees, so to speak the set of rules of a TAG, is called the set of *elementary trees*.

Tree γ in Figure 1 shows a TAG *derivation tree*, which is an initial tree with an arbitrary number of adjoinings (here β_1 is adjoined at the node S* in α and β_2 in the node S* in the adjoined tree β_1).

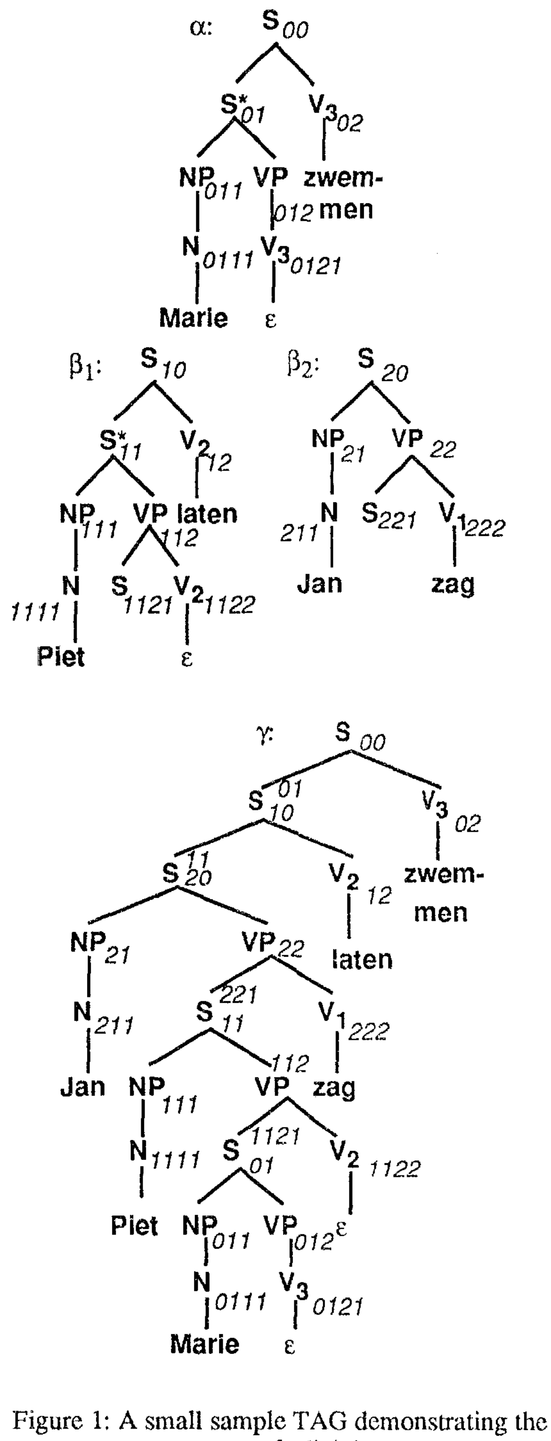

Figure 1: A small sample TAG demonstrating the process of adjoining

The most obvious property of TAG rules (elementary trees), which arises from the close relation with context-free derivation trees - with which linguists are familiar - is the easy way to write and understand such rules. The advantage instead of a context-free grammar producing the derivation trees is that related facts can be described in one rule. E.g., in Figure 1, each tree contains exactly the dependent pieces without any further processing (for more examples see, e.g., [Kroch, Joshi 85]).

With the close relation between TAGs and CFGs, one can think that both formalisms are equivalent. But TAGs are more powerful. In the linguistic community, it is discussed controversially, how powerful a linguistic formalism should be (see, e.g., Pullum 84] or [Shieber 85]). TAGs are *mildly context-sensitive*, which means that they can describe some context-sensitive languages, but not all (e.g., www with $w \in \{a,b\}^*$, but ww is acceptable for a TAG). There is the thesis that natural language can be described very well by a mildly context-sensitive formalism. But this can only be empirically confirmed by describing difficult linguistic phenomenons (here, the example in Figure 1 can only give an idea for the appropriateness of TAGs for natural language description).

If an efficient implementation of a parser for TAGs is desired (e.g., in a natural language access system to an expert system), the existence of polynomial time acceptors for the word problem of TAGs becomes relevant (upper time bound $O(n^4 \log n)$, see [Harbusch 89]). On the basis of this efficient algorithm the new definition has been implemented which is mentioned later in the summary.

With this short impression of some advantages of the TAG formalism, the disadvantages should now be tackled. The main property, which has its roots in the close relation to context-free grammars, is the same problem with *subcategorisation*. Further information encoded in the category name leads to combinatory explosion of the grammar. In the framework of CFGs, this disadvantage is removed by defining a Unification Grammar extending a context-free grammar.

2.2 PATR Unification Briefly Revisited

A *Unification Grammar* U (briefly called *UG* or *PATR* grammar) consists of a CFG G, where each rule is extended by a possibly empty set of *specification rules* (for a good introduction to Unification Grammars, e.g., see [Shieber 87]). Such a rule consists of two paths which are unified. A path consists of a number uniquely referring to a constituent in the context-free rule together with a list of *feature names* and/or an atomic *value*.

A pair (context-free rule, list of specification rules) is called *unification rule*. E.g.,((S NP VP)) (((0 fset)(1 fset))((1fset syntax)(2 fset syntax))((2 fset syntax verbform) active)))) is such a rule. Another representation of the specification rules is a *DAG* (directed acyclic graph). Figure 2 shows this representation for the above described example rule. It is built by representing the numbers, feature names and values as nodes which are connected in the way they are put together in the string representation. Common prefixes are represented only once.

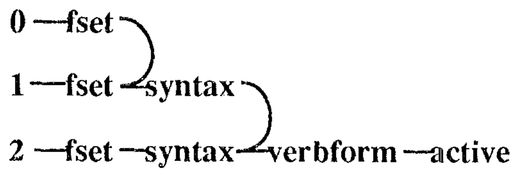

Figure 2: Example of a DAG representation

The process of recursion - called *unification* - is defined as an operation of union on all specification rules according to a context-free derivation, loosely spoken. More formally, the result of the unification of two DAGs x and y (UNIFY(x,y)) is defined inductively as a new DAG z, where

- $z = x$, if $x = y$,
- $z = x$, if x is atomic and y is empty, and vice versa $z = y$, if y is atomic and x is empty,
- if neither x nor y is atomic, then $\forall$ features l such that $|l,u| \in x$, $|l,v| \in y$, $|l,\mathrm{UNIFY}(u,v)| \in z$ and $\forall$ features l such that $|l,w| \in (u \cup v) - (u \cap v)$, $|l,w| \in z$.

It is easy to see that this extension of the context-free formalism allows us to introduce various additional information by specification rules. The main problem of the formalism is that it has Turing capacity - and so all disadvantages inherited with this power. Informally, this paper shows that the combination with TAGs restricts the power of unification.

3 The Two Different Definitions of TAGs with Unification

Although Turing capacity is not a wishful property, unification seems to be a good extension for TAGs as well. In this section, two different definitions for TAGs with Unification (UTAGs) are presented in a common terminology to simplify their comparison.

3.1 The Definition of the Grammar for TAGs with Unification

Same as to define specification rules according to a context-free rule, where the relation between both sets is represented by unique node numbers to refer to the different constituents in a rule. Here, according to an initial or an auxiliary tree, these specifications are defined between father and sons or between brothers via unique node numbers all over the trees. The trees α, β_1 and β_2 in Figure 1 (now interpreting the unique node numbers of the elementary trees) together with the according specifications in Figure 3 describe an example TAG with Unification, which produces the propagation of some syntactic and semantic information from the lexical items to the root node of the different subsentences. Note that the unique node numbers are also helpful to identify the individual adjoined trees in the derivation tree. To prevent the ambiguity resulting from adjoining the same tree more than once, the node number of the eliminated node is taken as a prefix for all new nodes.

```
specifiation rules :                  (((00  fset)(01  fset))
                                       ((01  fset)(02  fset))
                                       ((01  fset) (012  set))
                                       ((011  fset)(012  fset))
                                       ((012  fset)(0121  fset))
                                       ((011  fset)(0111  fset)))
(((10  fset sem_role dep_sent action) (((20  fset sem_role dep_sent action)
  (11  fset sem_role action))           (22  fset sem_role action))
 ((11  fset)(12  fset))                 ((20  fset sem_role dep_sent dep_sent action)
 ((11 fset)(112  fset))                  (22 fset sem_role dep_sent_action))
 ((111  fset) (112  fset))              ((21 fset)(22 fset))
  ((111  fset)(1111  fset))             ((211  fset)(211  fset))
  ((11 2 fset)(1121  fset)              ((22 fset)(222 fset))
 ((112  fset sem_role dep_sent actor)   ((22 fset sem_role dep_sent dep_sent actor)
  (1121  fset sem_role actor)))          (221  fset sem_role dep_sent actpr))
                                        ((22  fset sem_role dep_sent actor)
                                         (221  fset sem_role actor)))
Lexicon : (Marie ((N ((syntax  num) sing)((syntax case) nom)((syntax pers) 3)
                      ((synt_role subject) Marie)(sem_role actor) Marie)))
                 ((N ((syntax  num) sing)((syntax case) dat)((syntax pers) 3)
                      ((synt_role d_obj) Marie) ((sem_role recipient) Marie)) )
                 ((N ((syntax  num) sing)((syntax case) acc)((syntax pers) 3)
                      ((synt_role  i_d_obj) Marie)((sem_role recipient) Marie))))
          ... same information for Jan and Piet ...
          (zwemmen ((V ((syntax verbform) inf)((synt_role verb) zwemmen)
                        ((synt_role subject))((synt_role d_obj) NONE)
                        ((synt_role i_d_obj) NONE)((sem_role action ) zwemmen))
          (laten ((V ((syntax verbform) inf)
                     ((synt_role verb) laten)synt_role d_obj) NONE)
                     ((synt_role i_d_obj) NONE)(((sem_role action) laten)))
          (zag  ((V ((syntax verbform) fin) ((syntax pers) 3)((syntax num) sing)
                    ((synt_role verb) zagen)((synt_role d_obj) NONE)
                    ((synt_role i_d_obj) NONE)((sem_role action) zagen)))
```

Figure 3: Specification rules for α, β_1 and β_2 in Figure 1

To get an impression of what that grammar does, one can read all relations between father and sons in all initial and auxiliary trees as context-free rules annotated with the corresponding specification rules.
To realize this partial interpretation of specification rules, the following sets for each node x are defined:
$\uparrow$x := the set of all specification rules with x father or brother of the other mentioned node in that rule,
$\downarrow$x := the set of all specification rules, where x is a son of the further mentioned node in that rule and
$\Diamond$x := the set of all specification rules, where a value of x is defined.
It is easy to see that for each node these sets can be automatically computed. E.g, for the node 01 in α, $\uparrow$01 := {(((00 fset)(01 fset)), ((01 fset)(02 fset))}, $\downarrow$01 := {(((01 fest)(012 fset))}, $\Diamond$01 := $\emptyset$. *Vijay-Shanker* and *Joshi* prefer to write the grammar in the $\uparrow$ (TOP, or briefly called t) and $\downarrow$ (BOTTOM, b) terminology, which allows slide differences in expressiveness, e.g., a non-empty TOP set of the root node of an auxiliary tree can be specified.
The Unification Grammar, consisting of all rules built by interpreting each father and all its sons in all elementary trees as context-free rules, together with all TOP sets, can be interpreted as described in section 2.2. But it is important to note, the two resulting grammars are not equivalent, because the context-free grammar doesn't require the derivation of the whole tree, if one context-free rule out of it is in use.
Considering this simple and intuitive interpretation of a UTAG, the problem of defining adjoining and unification directly for such a grammar becomes obvious. Only the relations between father and sons or brothers are interpreted directly, although there are links defined over a whole tree. This is exactly the point, where the two definitions differ.

3.2 The Definition of Vijay-Shanker and Joshi

The definition of *Vijay-Shanker* and *Joshi* separates this local information (for a description of their approach see, e.g., [Vijay-Shanker 87]). It reminds us of the interpretation of attributes after computing the context-free derivation tree for an *Attribute Grammar* (e.g., see [Aho et al. 86]).
In their approach, the $\uparrow$ and $\downarrow$ sets remain isolated until all adjoinings are made. With this strategy it is clear that the unification cannot be used to reduce the number of structure trees, which are unificationally ill-formed. More formally spoken, the adjoining is defined as described in Figure 4 (for the reason of uniquely referencing to the $\uparrow$ and $\downarrow$ sets at each node X, t and b with different bar levels are used). After all adjoinings were made, all $\uparrow$ and $\downarrow$ sets at each node are unified.
The disadvantage of this sequential interpretation of adjoining and unification can be demonstrated by the example grammar. In the lexicon, all names, "Marie", "Piet" and "Jan", have three different cases (nominative, dative and accusative). Therefore, three structurally equivalent trees are produced for each tree α, β_1 and β_2. These are structurally combined by adjoining. Out of this collection, by the specification rule, which demands a subject, unification selects one correct reading. But this is checked after building nine different derivation trees.

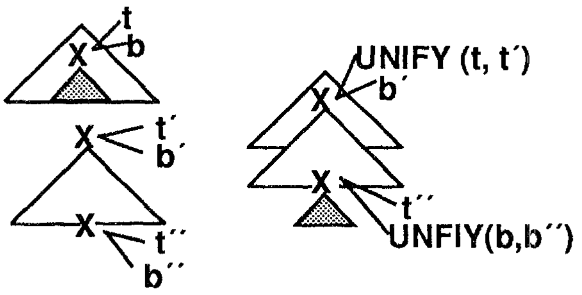

Figure 4: The adjoining definition for the approach of *Vijay-Shanker* and *Joshi*

One way to handle that problem is to use unification with disjunction to reduce the set of structurally equivalent trees. But this doesn't tackle the problem fundamentally, because it cannot reduce ambiguities, which only can be eliminated by interpreting the specification rules at once. This integration is realized in our new approach. The different approaches are abbreviated with SUTAG for the more sequential approach of *Vijay-Shanker* and *Joshi* , and IUTAG for a more integrated approach, which is presented now.

3.3 An Integrated Definition for TAG and Unification

The basis for our definition is a UTAG given in the notation as described in section 3.1. For each elementary tree, a set of specification rules is defined, which can be interpreted as unified DAGs over the whole tree. E.g., the path 'fset sem_role action' of node 01 has the value "zwemmen".
If an adjoining should happen in a node X, for this node the sets $\uparrow$X, $\downarrow$X and $\Diamond$X are computed because this node and with it all links from its DAG to other DAGs are replaced by an auxiliary tree. Structurally, the adjoining looks like the original one for TAGs (and same as for SUTAGs). In the case of an adjoining, a node in an (possibly modified) initial tree should be replaced by a whole auxiliary tree β, and all links that node had have to be modified by the information of β. You can imagine the new linked DAGs all over the adjoined tree as being a filter for the former propagated information. E.g., information passing a node X, where an adjoining will take place, must not be supported by the path from the root to the foot node of the adjoined auxiliary tree. So the propagation is stopped somewhere in the tree.
More formally spoken, the definition of adjoining can be given as described in Figure 5. The DAG of the node, in which the adjoining will take place, is represented by $\uparrow$ and $\downarrow$ sets.
Here, r and f stand for the whole DAG of the root and foot node of the auxiliary tree, which will be adjoined. But it is obvious that the $\uparrow$ set of r is empty as well as the $\downarrow$ set of f. This is clear because there exists no father of the root and no son of the foot node where these links can end.
Using the same terminology for the auxiliary tree as in Figure 4 (r is separated in t´ and b´, and f in t´´ and b´´), one can write instead of "UNIFY(t,r)" as well

"UNIFY(t,b')" and for "UNIFY(b,f)" as well "UNIFY(b,t'')". What becomes obvious is that in the resulting tree each node has a DAG, which is connected with the DAGs of its neighbors in the tree. This was the aim of the new definition, always to produce linked DAGs all over the derivation tree to test for failure of unification at once.

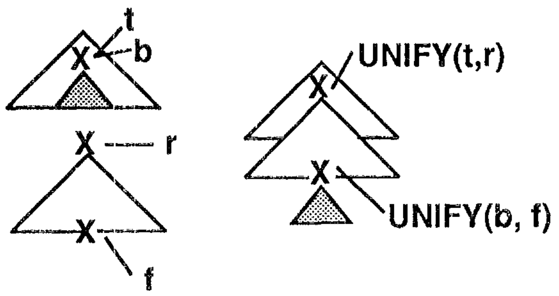

Figure 5: The adjoining definition for IUTAGs

But this is not the complete definition; there are two things to be added. One is the concrete handling of the elimination of links and the second is the reintroduction of the value definition of the eliminated node. Beginning with the elimination of links, the separation into ↑ and ↓ at a node X, an adjoining will take place, means that all information propagated along the node X is eliminated (e.g., one can imagine a reason maintenance system to keep track of that task, so that unification is no more commutative and independent from the time of introducing information). E.g., information from a leaf is propagated to the root of the whole tree. This propagation can be interrupted, modified or kept untouched if an adjoining takes place. Intuitively, the adjoined tree can be imagined as a filter for the propagation.
Some extra computation must be done for the reintroduction of the value information (◊X) in the following way. Since the node, in which a value is defined, will be eliminated in the case of an adjoining, the question arises, what should happen to the value definition? Here, it was decided to find a point for the reintroduction of that information. The first idea can be to say, by definition add it to the root node. But now a fail can be produced in the case that the adjoined tree adds parts of the path to reach that value (e.g., if the value definition in node X with the node number 01 is ((01 fset) val) and in the root node (10) of the adjoined tree, the specification rule exists ((10 fset next) (11 fset)), a fail is produced because fset has at the same time a feature and a value as successor). To allow this property to be interpreted without failure, which is desirable for the idea of defining a filter via adjoining, a computation of such parts of paths in the adjoined tree is done to find the maximal extension of the path, behind which the value can be added without producing a fail.
In this process, called computation of the *inheritance history*, all maximal prefixes of paths in the adjoined auxiliary tree are computed. Out of this set, those candidates are chosen, which have the value definition as prefix p. Behind these maximal paths the value is reintroduced. Because this selection does not always have a unique path at exactly one node as result (e.g., if root and foot node add two different feature names behind p, but between both no propagation occurs), at the end of both paths the value is reintroduced by definition. It is clear that the computation of the inheritance history can be done once for all paths in all auxiliary trees, so that this part of the definition doesn't extend the execution time very much. The most elaborate work has to be done in reconstructing the correct links all over the derivation tree after an adjoining. In the worst case, changes of propagation all over the derivation tree are required.
To give an idea of how this definition works, in Figure 6 the changes during the adjoining of β_1 and β_2 in α are represented. Here only the nominatve reading is in use, because the lexical reading with case dative and accusative produces a fail in unification with the valency description of the verbs. Concentrating on the feature path "fset sem_role" (note, we don't claim that this is a serious semantics of the sentence!), first the meaning of "Marie zwemmen" is produced (action is "zwemmen" and actor is "Marie"), which is modified during the adjoining by "Piet laten" and "Jan zag".

3.4 The Comparison of the Properties of SUTAG and IUTAG

Finally, let's weigh the two approaches. A first impression in reading both definitions can be that the differences are marginal. It is simply a decision in ordering processes. But a second look offers that there are more differences. The ideas of what should be represented by an adjoining are different. A SUTAG supports the monotone idea behind unification and the parallel and absolute presence of propagated knowledge (what is written in the ↑ and ↓ sets should be propagated exactly from that node all over the tree without any more changes). In a UTAG, the idea of filtering propagated information has the highest priority. Specification information should be revisable by new adjoinings. Each of the two approaches works more efficient for different problem classes, which cannot be characterized in more detail here.
For SUTAGs, the main disadvantage is the late interpretation of the specification rules. Especially here lies the motivation for the definition of IUTAGs. In the above mentioned example, the focus in describing the advantage of online unification is demonstated by the representation of the intermediate states of the DAGs, but it is clear as well, how the lexical readings with case equal dative or accusative for "Marie","Piet" and "Jan" are eliminated.
Another distinguishing feature of IUTAGs is that unification is not further a monotone process as it is in SUTAGs (same as for UGs). Links, which have been installed, can be eliminated. This is the more formally circumscribed effect of a filter, which is the intuitive idea behind that definition. It is obvious that the realization of that fact is more expensive than to wait until all adjoinings are realized. But it comes up with a reduction of execution time for invalid readings. Actually, this is a trade off, which has more or less effect on the time complexity in the relation in which specification rules are used to subcategorize and restrict the structural descriptions. Therefore, both definitions probably have more perfect application domains.
A point which was mentioned as a disadvantage of the Unification formalism, was the Turing capacity. For both definitions of UTAGs, the power is restricted in a

way that no path can be introduced via specifications without adding a piece of structure, because the definition of an elementary tree requires a non-empty leaf. Therefore, the defined unification process underlies the constant growth property. At the moment, it is not yet clear, what class of languages can exactly be described with that formalism, but it is obvious that it is less powerful than Turing machines (e.g., languages such as a^{2^n} are not describable). But a result for an upper time bound of an algorithm for that formalism is yet unknown.

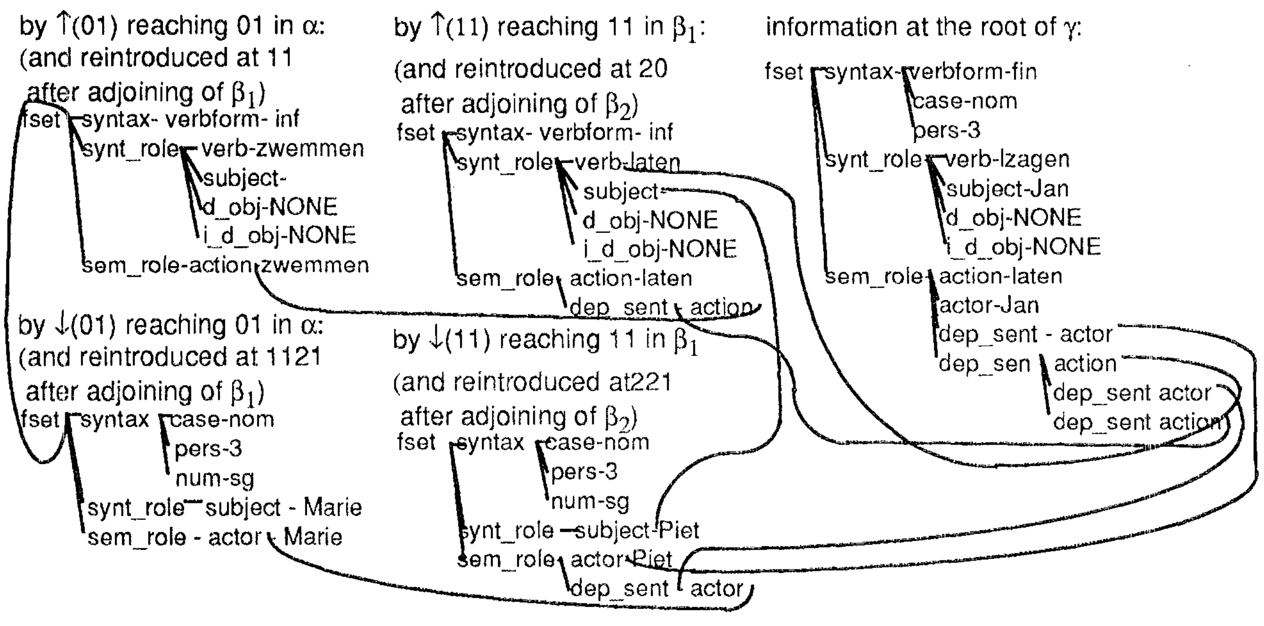

Figure 6: Some intermediate states for an example demonstrating the adjoining definition for IUTAGs

4 Summary

Also the UTAG definition can be seen purely as a theoretical result, syntax description as the most prominent application domain has influenced the design. Therefore, our experience with an implementation of the IUTAG definition running a natural language grammar is mentioned here (for all technical details see [Buschauer et al. 89]). The system is written in Common LISP on a Hewlett Packard machine of the 9000 series with emphazise on efficiency. E.g., the response time for a small test grammar and sentences of a length of about 10 words is less than 10 milliseconds. Presently, this pure parser is extended by tools (e.g., consistency check) to build a workbench for linguists designing IUTAGs.

On this basis, more empirical results should be produced for that domain. Currently, we work on further extensions of the definition (e.g., ID/LP for free word order in German). Our main emphazise lies on the aspect of *incrementality*. In this paper the interpretation direction of the definitions was analysis. But the other direction, the problems in *generation* are faced now, with the ambitious aim to verify that IUTAGs are appropriate for a bidirectional and integrated description of syntactic, semantic and pragmatic facts.

Literature

[Aho et al. 86] **A. V. Aho, R. Sethi, J. D. Ullman:** *Compilers: Principles, Techniques, and Tools,* Addison-Wesley, Reading, Massachusetts, 1986.

[Buschauer et al. 89] **B. Buschauer, P. Poller, A. Schauder, K. Harbusch:** *Parser für TAGs mit Unifikation,* "AI-Laboratory" Memo in press, Department of Computer Science, University of the Saarland, Saarbrücken, FRG, 1989.

[Harbusch 89] **K. Harbusch:** *Effiziente Strukturanalyse natürlicher Sprache mit Tree Adjoining Grammars,* PhD Thesis, University of Saarland, Saarbrücken, FRG, 1989.

[Joshi 85] **A. K. Joshi:** *An Introduction to Tree Adjoining Grammars,* Technical Report MS-CIS-64, University of Pennsylvania, Philadelphia, Pennsylvania, 1985.

[Joshi et al. 75] **A. K. Joshi, L. S. Levy, M. Takahashi:** *Tree Adjunct Grammars,* Journal of the Computer and System Sciences 10, 1975.

[Kroch, Joshi 85] **A. S. Kroch, A. K. Joshi:** *The Linguistic Relevance of Tree Adjoining Grammars,* Technical Report MS-CIS-16, University of Pennsylvania, Philadelphia, Pennsylvania, 1985.

[Pullum 84] **G. Pullum:** *On Two Recent Attempts to Show That English is Not a CFL,* Computational Linguistics 10:4, 1984.

[Shieber 85] **S. M. Shieber:** *Evidence against the Context-Freeness of Natural Langage,* Linguistics and Philosophy 8, 1985.

[Shieber 87] **S. M. Shieber:** *An Introduction to Unification-Based Approaches to Grammar,* CSLI-Lecture Notes 4, Stanford, California, 1987.

[Vijay-Shanker 87] **K. Vijay-Shanker:** *A Study of Tree Adjoining Grammars,* PhD Thesis, University of Pennsylvania, Philadelphia, Pennsylvania, 1987.