Grammar conversion from LTAG into HPSG

NAOKI YOSHINAGA
Department of Information Science, Graduate school of Science, University of Tokyo
yoshinag@is.s.u-tokyo.ac.jp

YUSUKE MIYAO
Department of Information Science, Graduate school of Science, University of Tokyo
yusuke@is.s.u-tokyo.ac.jp

ABSTRACT. We propose a grammar conversion algorithm from an arbitrary Feature-Based Lexicalized Tree Adjoining Grammar (FB-LTAG) grammar into a strongly equivalent Head-Driven Phrase Structure Grammar (HPSG)-style grammar. Our algorithm converts LTAG elementary trees into HPSG feature structures by encoding a tree structure in a list. A set of pre-determined rules manipulate the list to emulate substitution and adjunction. By using our algorithm, we can obtain HPSG-style grammars from existing LTAG grammars. We apply this algorithm to the XTAG English grammar and report some findings.

1 Introduction

Recent intelligent NLP applications (Kay, Gawron, and Norvig 1994) take advantages of computationally and linguistically motivated grammar formalisms, such as Feature-Based Lexicalized Tree Adjoining Grammar (FB-LTAG[1]) (Vijay-Shanker 1987; Vijay-Shanker and Joshi 1988) and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag 1994). Although both formalisms have the same motivation to model natural languages, the correspondence between the formalisms is not well discussed. We believe that collaboration between the communities will be beneficial for their further improvement.

This research aims at providing a basis for the collaboration, and the proposal is a grammar conversion from LTAG into HPSG. Our conversion guarantees the strong equivalence, that is, parsing results (derivation trees) of an LTAG grammar can be derived from parsing results of the obtained

[1] If not confusing, we use the term LTAG to refer to FB-LTAG in this paper.

HPSG-style grammar. Strongly equivalent grammars based on the different formalisms are valuable for both communities in practical, computational, and theoretical aspects (Becker and Lopez 2000):

Resource sharing HPSG-based applications can make use of LTAG resources (lexicons and grammars) such as large-scale English (Doran, Hockey, Sarkar, Srinivas, and Xia 2000) and French grammars (Abeillé and Candito 2000). Our algorithm can reduce the considerable workload to develop huge resources from scratch.

Parsing efficiency We can apply HPSG parsing techniques to LTAG parsing by applying them to the obtained HPSG-style grammar. Recent development of the HPSG parsing techniques allows the use of HPSG-based processing in practical application contexts (Flickinger, Oepen, and Tsujii 2000). Experiments using state-of-the-art parsers show the efficiency of an HPSG parser compared to an LTAG parser.

Linguistic correspondence We can explore the linguistic correspondence between the formalisms. Since the obtained grammar has computational features of HPSG and linguistic features of LTAG, their difference will be apparent by comparing the obtained grammar with existing HPSG grammars.

We implemented the conversion algorithm, and successfully converted the latest version of the XTAG English grammar (The XTAG Research Group 2001), which is a large-scale FB-LTAG grammar. The strong equivalence was empirically attested by the fact that parsing with the original and the obtained grammars generated equivalent parse results. Strongly equivalent grammars enable the fair comparison of LTAG and HPSG parsers, and another work reports that an efficient HPSG parser achieved a parsing speed that was significantly higher than an existing LTAG parser (Yoshinaga, Miyao, Torisawa, and Tsujii 2001). In this paper, we investigated the types of linguistic phenomena covered by the XTAG English grammar, and correspondence to their analysis in the HPSG formalism.

Tateisi et al. also translated LTAG into HPSG (Tateisi, Torisawa, Miyao, and Tsujii 1998). However, their method depended on translator's intuitive analysis of the original grammar, and thus the translation was manual and grammar dependent. The manual translation demanded considerable efforts from the translator, and obscures the strong equivalence between the original and obtained grammars. Other works converted HPSG into LTAG (Kasper, Kiefer, Netter, and Vijay-Shanker 1995; Becker and Lopez 2000). Given the greater generative power of HPSG, the conversion required some restrictions on HPSG to suppress its generative capacity. As a result, the conversion loses the strong equivalence of the grammars. The existing works cannot gain the above advantages, which are attainable only when the strong equivalence is preserved.

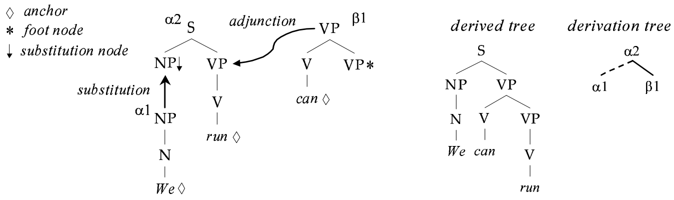


Figure 1.1: Tree Adjoining Grammar: basic structures and their composing operations

2 Grammar formalisms

2.1 Feature-Based Lexicalized Tree Adjoining Grammar (FB-LTAG)

LTAG (Schabes, Abeille, and Joshi 1988), an input of our algorithm, is a grammar formalism that provides syntactic analyses for a sentence by composing *elementary trees* with two operations called *substitution* and *adjunction*. Elementary trees are classified into two types, *initial trees* ($\alpha 1$ and $\alpha 2$ in Figure 1.1) and *auxiliary trees* ($\beta 1$ in Figure 1.1). An elementary tree has at least one leaf node labeled with a terminal symbol called an *anchor* (marked with $\diamond$). In an auxiliary tree, one leaf node is labeled with the same symbol as the root node and is specially marked as a *foot node* (marked with $*$). In an elementary tree, leaf nodes with the exception of anchors and a foot node are called *substitution nodes* (marked with $\downarrow$).

The left-hand side of Figure 1.1 illustrates the two operations. *Substitution* replaces a leaf node (substitution node) with an initial tree and *adjunction* grafts an elementary tree with the root node and a leaf node (foot node) labeled x onto a node with the same symbol x. Results of analysis are described not only by *derived trees* (i.e., parse trees) but also by *derivation trees* (Figure 1.1). A derivation tree represents the history of combinations of trees and is a structural description in LTAG.

FB-LTAG (Vijay-Shanker 1987; Vijay-Shanker and Joshi 1988) is an extension of the LTAG formalism. In FB-LTAG, each node in elementary trees has a feature structure, containing grammatical constraints on the node.

2.2 Head-Driven Phrase Structure Grammar (HPSG)

We define HPSG, an output of the algorithm, following the computational specifications in (Pollard and Sag 1994). It consists of *lexical entries* and *ID grammar rules*, each of which is described with typed feature structures (Carpenter 1992). A lexical entry for each word expresses characteristics of the word, such as subcategorization frame and grammatical category.

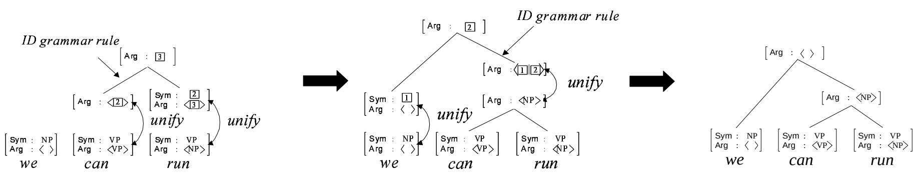


Figure 1.2: Parsing with an HPSG grammar

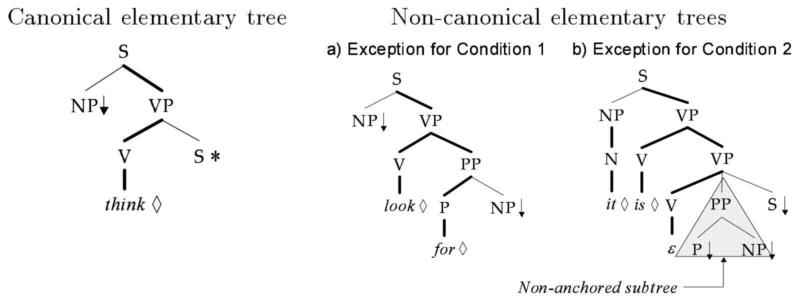


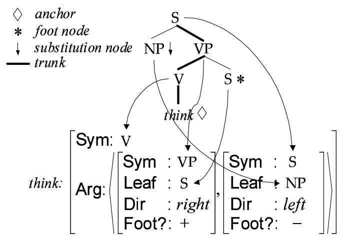
Figure 1.3: A canonical elementary tree and exceptions

An ID grammar rule represents a grammatical relation between a mother and its daughters, and is independent of lexical characteristics.

Figure 1.2 illustrates an example of bottom-up parsing with an HPSG grammar. As we can see in this example, in HPSG a parse tree is generated by incrementally applying ID grammar rules to lexical entries and constructing each branching one by one, while in LTAG it is done by composing elementary trees with the two operations. Thus, the key points of the conversion are 1) how to encode a tree structure of an elementary tree as an HPSG feature structure, and 2) how to emulate substitution and adjunction. We should notice that there is not one-to-one correspondence between an elementary tree and an HPSG lexical entry because one elementary tree can have multiple anchors to represent compound expressions.

3 Algorithm

This section describes an algorithm for grammar conversion in detail. As described in Section 2, tree structures of LTAG elementary trees should be encoded in HPSG lexical entries, and the tree composing operations should be emulated by ID grammar rules. Thus, we propose a conversion algorithm which consists of: 1) conversion of elementary trees into HPSG lexical entries and 2) emulation of substitution and adjunction by pre-determined ID grammar rules.



Sym: symbol of a trunk node
Leaf: symbol of a leaf node
Dir: the direction of a leaf node against the trunk
Foot?: the type of a leaf node

```
procedure convert_tree_into_lexical_entry(T)
begin
  arg := □
  for i := 1 to depth(T)−1
    n_{i−1} := trunk(T, i − 1)
    l_i := leaf(T, i)
    d_i := direct(T, i)
    t_i := type(T, i)
    b_i := ⟨n_{i−1}, l_i, d_i, t_i⟩
    arg := [b_i] ⊕ arg
  end for
  L := (n_{depth(T)−1}, arg)
  return L;
end

depth:  returns the depth of the anchor.
trunk:  returns the symbol of the trunk node.
leaf:   returns the symbol of the leaf node in depth i
direct: returns which side of the trunk node
        the leaf node is in depth i
type:   returns the type of the leaf node in depth i
```

Figure 1.4: A conversion algorithm from a canonical elementary tree T into an HPSG lexical entry L

In the following, we first state the notion of *canonical*, and show *canonical elementary trees*, which have a one-to-one correspondence to an HPSG lexical entry[2]. Canonical elementary trees are elementary trees which satisfy the conditions below:

Condition 1 A tree must have only one anchor.

Condition 2 All branchings in a tree must contain trunk nodes.

Trunk nodes are nodes on a *trunk*, which is a path from an anchor to the root node (the thick line in Figure 1.3) (Kasper, Kiefer, Netter, and Vijay-Shanker 1995), with the exception of the anchors. Condition 1 guarantees that a canonical tree has only one trunk, and Condition 2 guarantees that each branching consists of a trunk node, a leaf node, and their mother (also a trunk node) (the left-hand side of Figure 1.3). The right-hand side of Figure 1.3 shows non-canonical ones. We call a subtree which is a depth 1 or above with no anchor a *non-anchored subtree*. Non-canonical elementary trees are first converted to canonical ones, and then applied the algorithm for canonical ones.

3.1 Conversion of canonical elementary trees

This section describes a conversion algorithm of an LTAG canonical elementary tree into an HPSG lexical entry. Canonical elementary trees can be directly converted to HPSG lexical entries as shown in the left-hand side of Figure 1.4.

[2] In this paper, we discuss a conversion of elementary trees which consist of binary branchings. A conversion of trees with unary branchings is straightforward and trees with n-ary ($n \geq 3$) branchings can be converted into trees with binary branchings.

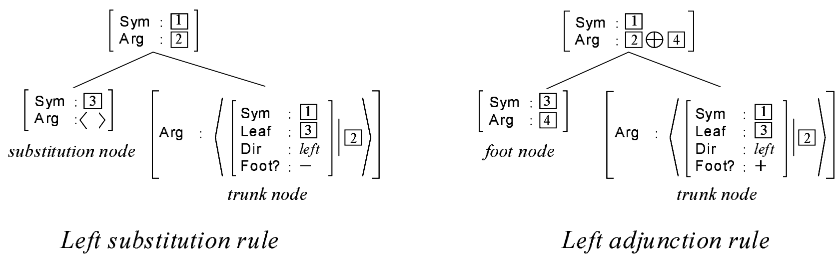


Figure 1.5: Grammar rules: substitution rule and adjuction rule

The procedure `convert_tree_into_lexical_entry` in Figure 1.4 depicts an algorithm for converting an elementary tree T into an HPSG lexical entry L. In the algorithm, *arg* is a list of branchings b_i described with a quadruplet $\langle n_{i-1}, l_i, d_i, t_i \rangle$ along the trunk. The parameter n_{i-1} represents the mother node of the trunk node n_i. The parameters l_i, d_i and t_i represent a leaf node with depth i; they represent the non-terminal symbol, the direction (on which side of the trunk node n_i the leaf node is), and the type (whether a foot node or a substitution node) respectively. We call this list the *arguments* of the word. Finally the converted lexical entry L is described with the arguments *arg* and the mother of the anchor, namely, $n_{\mathtt{depth(T)}-1}$ where `depth(T)` is the depth of the tree T.

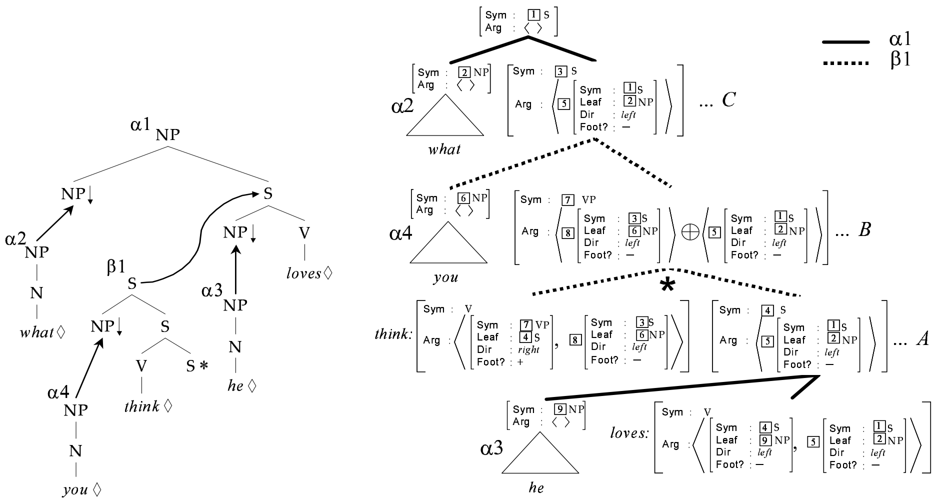
In the left-hand side of Figure 1.4, the value of the Sym feature is the symbol $n_{depth(T)-1}$, the value of the Arg feature contains the arguments in *arg*, as a list of feature structures with four features Sym, Leaf, Dir and Foot? corresponding to n_{i-1}, l_i, d_i and t_i respectively.

3.2 Definition of grammar rules

In this section, we give the definition of grammar rules to emulate substitution and adjunction (Figure 1.5) [3]. We should note that these grammar rules are independent of the original grammar because they do not specify any characteristics given in the original grammar.

Substitution rule: The Sym feature of the substitution node must have the value of the Leaf feature $\boxed{3}$ of the trunk node. The Arg feature of the substitution node must be a null list, because the substitution node must be unified only with the node corresponding to the root node of the initial tree. The substitution rule percolates the tail elements $\boxed{2}$ of the Arg feature of the trunk node to the mother in order to continue constructing the tree.

[3] In the figure, we give rules whose right daughter is a trunk node. Of course, there are symmetric rules whose left daughter is a trunk node.



LTAG derivation HPSG rule applications

Figure 1.6: LTAG derivation and HPSG rule applications for a phrase "*what you think he loves*"

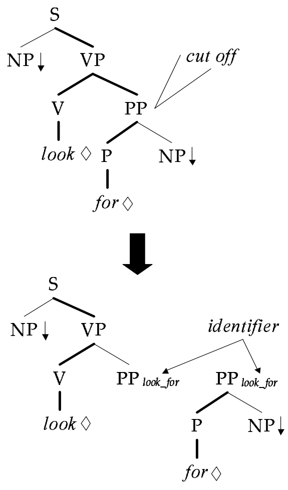
Adjunction rule: The Sym feature of a foot node must have the same value as the Leaf feature $\boxed{3}$. The value of the Arg feature of the mother node is a concatenation list of both Arg features $\boxed{2}$ and $\boxed{4}$ of its daughters because we first construct the tree corresponding to the adjoining tree and next continue constructing the tree corresponding to the adjoined tree. The value "+" or "−" of the Foot? feature explicitly determines whether the next rule application is the adjunction rule or the substitution rule.

Figure 1.6 shows an example of rule applications. The thick line indicates the adjoined tree ($\alpha 1$) and the dashed line indicates the adjoining tree ($\beta 1$). The adjunction rule is applied to construct the branching marked with $\star$, where "think" takes as an argument a node whose Sym feature's value is S. By applying the adjunction rule, the Arg feature of the mother node (B) becomes a concatenation list of both Arg features of $\beta 1$ ($\boxed{8}$) and $\alpha 1$ ($\boxed{5}$). Note that when the construction of $\beta 1$ is completed, the Arg feature of the trunk node (C) will be its former state (A). We can continue constructing $\alpha 1$ as if nothing had happened.

3.3 Division of multi-anchored trees

Multi-anchored elementary trees, which violate Condition 1, are divided into multiple canonical elementary trees. We call the cutting nodes in the divided trees *cut-off nodes*. Note that a cut-off node is marked by an *identifier* to preserve a co-occurrence relation among the multiple anchors (Figure 1.7).

The procedure `divide_tree_into_subtrees` in Figure 1.7 depicts an




```
procedure divide_tree_into_subtrees(MT)
begin
  A := select(MT)
  ⟨ST, 𝒯⟩ := divtree(MT, A)                       ··· (1)
  foreach T (𝒯)
    S𝒮𝒯 := divide_tree_into_subtrees(𝒯)
    𝒮𝒯 := S𝒮𝒯 ∪ 𝒮𝒯
  end foreach
  𝒮𝒯 := 𝒮𝒯 ∪ {ST}
  return 𝒮𝒯;
end

procedure divtree(MT, A)
begin
  𝒯 := φ
  for i := 1 to depth(MT, A)-1
    if (nonleaf(arg(trunk(i))))
      ⟨MT′, T⟩ := cut(MT, arg(trunk(i)))           ··· (2)
      address(trunk(i), Address)
      mark(Address, MT′, T)                        ··· (3)
      𝒯 = 𝒯 ∪ T
      MT := MT′
    end if
  end for
  ST := MT
  return ⟨ST, 𝒯⟩
end

select:  returns one of anchors.
depth:   returns the depth of the anchor.
trunk:   returns the trunk node in depth i
arg:     returns the sister of the trunk node.
cut:     cuts off the tree at the sister of the trunk node and
         returns a subtree whose root node is the sister node.
nonleaf:returns true if not a leaf node.
address:returns address in the elementary tree.
mark:    marks the address to each cut-off node.
```

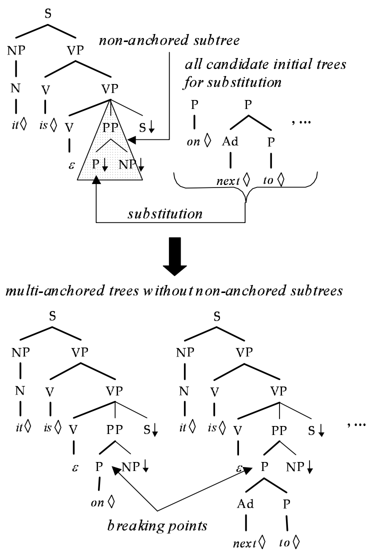
Figure 1.7: An algorithm for dividing a multi-anchored elementary tree MT into single-anchored trees $\mathcal{ST}$

algorithm for dividing a multi-anchored elementary tree MT into a set of single-anchored subtrees $\mathcal{ST}$. One anchor A is selected, and a single-anchored tree ST which can be constructed from the anchor A is picked up (`(1)` in Figure 1.7). We check the path from the root node to the anchor A, cut off the sister node [4] `arg(trunk(`i`))` if it is not a leaf node, and store the *address* of an elementary tree into the cut-off node as an identifier (`(2)` and `(3)` in Figure 1.7).

3.4 Substitution to non-anchored subtrees

Non-canonical elementary trees violating Condition 2 have non-anchored subtrees. A non-anchored subtree is converted into multi-anchored trees by substituting the deepest node with initial trees (Figure 1.8). Substituted nodes are marked as *breaking points* to remember that the nodes originate from the substitution nodes. In the resulting trees, all subtrees are anchored

[4] We should mention that the path from the foot node to the root node (*spine*) in an auxiliary tree must not be cut because the spine represents the chain of head signs between the root node and the foot node, which are unified with the same internal node in the other elementary trees.




```
procedure expand_tree_into_anchored_tree(NT)
begin
  𝓑𝓡 := na_br(NT)
  𝓜𝓣 := {NT}
  foreach BR (𝓑𝓡)
    S := select(BR)                                 ··· (1)
    𝓘𝓣 := initial(S)                                ··· (2)
    𝓜𝓜𝓣 := φ
    foreach MT (𝓜𝓣)
      𝓣𝓜𝓣 := substitute(MT, S, 𝓘𝓣)                  ··· (3)
      mark(S)
      𝓜𝓜𝓣 := 𝓣𝓜𝓣 ∩ 𝓜𝓜𝓣
    end foreach
    𝓜𝓣 := 𝓜𝓜𝓣
  end foreach
  return 𝓜𝓣
end

na_br:       returns the deepest branchings, whose
             daughters do not consist of an anchor.
initial:     returns all initial tree whose root node is the
             same as the leaf node.
select:      returns one of leaf nodes.
substitute:  causes substitution to the leaf node and
             returns a set of resulting multi-anchored trees.
mark:        marks the substituted node
```

Figure 1.8: An algorithm for converting a non-anchored subtree NT into anchored trees $\mathcal{MT}$

so that we can apply the above conversion algorithms.

The procedure `expand_tree_into_anchored_tree` in Figure 1.8 depicts an algorithm for combining a non-anchored subtree NT with initial trees into anchored trees $\mathcal{MT}$. For each branching consisting of substitution nodes or foot nodes, one substitution node S is selected (`(1)` in Figure 1.8). The function `substitute` causes a substitution to the node S with all initial trees which can substitute to S (`(3)` in Figure 1.8).

We should consider the termination of this algorithm. Since the function `initial` returns all initial trees whose root node is the same as the leaf node (`(2)` in Figure 1.8), the initial trees might include a tree violating Condition 2, to which this algorithm should also be applied. This seems to cause an infinite recursion of application of this algorithm, but it can be avoided by removing non-anchored subtrees from such an initial tree before applying this algorithm.

3.5 Strong equivalence

Our algorithm guarantees the strong equivalence of the original and obtained grammars. In the obtained grammar, the above grammar rules are applied only to feature structures corresponding to nodes which can substitute/be adjoined in the LTAG canonical elementary trees because encoded branchings in **Arg** specify the nodes to be subcategorized next. The strong equivalence holds also for conversion of non-canonical elementary trees. For trees violating Condition 1, we can distinguish the cut-off nodes from the substitution nodes owing to identifiers, which recover the co-occurrence relation in the original elementary trees between the divided trees. For trees violating Condition 2, we can identify substitution nodes in a combined tree because they are marked as breaking points, and we can consider the combined tree as two trees in the LTAG derivation.

By following a history of rule applications and mapping each of them to substitution or adjunction, we can recover an LTAG derivation tree from an HPSG parse tree. First, we find the syntactic head of a parse tree by following a trunk node when the substitution rule was applied, or a foot node when the adjunction rule was applied. We then follow the path from the syntactic head to the root node. If we find an application of the substitution rule, we can readily map it to LTAG substitution. If we find an application of the adjunction rule, the node takes adjunction. We remember the length of the Arg feature of the node to identify the adjoining tree. If the length of the Arg feature of a trunk node is longer than the one we remembered, we are in the adjoining tree. When the length becomes equal to the one we remembered, the construction of the adjoining tree reaches to its root node and the construction of the adjoined tree restarts. Thus we can bring down the adjoining tree. Let us consider the case in Figure 1.6. First we find that "*love*" is a syntactic head of the parse tree. We then follow the path from the anchor to the root node of $\alpha 1$. Since we first find an application of the substitution rule, we can map it to the substitution of $\alpha 3$ to $\alpha 1$. Then, the next rule is the adjunction rule (marked with $\star$), and we can find the node A takes adjunction. We thus remember the length of the Arg feature of the node A, and follow the trunk until the length of the Arg feature is equal to that of the node A. At the node C, the length of the Arg feature is equal to the node A. This implies that the construction of the adjoining tree $\beta 1$ is completed at the node C.

3.6 Extension to FB-LTAG

The above algorithm gives the conversion of LTAG, and it can be easily extended to handle an FB-LTAG grammar by merely storing a feature structure of each node into the Sym feature and Leaf feature together with the non-terminal symbol. ID grammar rules execute feature structure unification done in LTAG substitution and adjunction.

3.7 Remaining issue

The above algorithm gives a formal link between HPSG and LTAG, but the linguistic aspects are not well discussed. In this section, we discuss the linguistic aspects according to the notion of the syntactic head, which is a central notion in HPSG.

Distinction between predicative/modifying auxiliary trees We should mention that there are two kinds of auxiliary trees in LTAG (Kroch 1989; Schabes and Shieber 1994). One is a predicative auxiliary tree, and the other is a modifier auxiliary tree. The former introduces a predicate that subcategorizes for a phrase of the category of its foot node and assigns a thematic role to the phrase, while the latter introduces a modifying, complement or dislocated phrase. This distinction between the two kinds of auxiliary trees is given roughly by determining which daughter is head, a foot node or a trunk node. Tateisi et al. (Tateisi, Torisawa, Miyao, and Tsujii 1998) distinguished these trees by manually analyzing feature percolation in auxiliary trees and by assigning HPSG rule schemata separately to them. In this paper, we consider that all trunk nodes are head, but it can be manually or semi-automatically determined by giving some linguistic cues or by analyzing feature percolation.

Head selection It is not entirely clear how we should `select` the anchor A in Figure 1.7. Since most multi-anchored trees represent compound expressions or idioms, such as "*in front of*" and "*kick the bucket*", this problem can be replaced with another problem that which word of a phrase should be syntactic/semantic head. In the current implementation, we simply select an anchor from the left most one, though we can select one by using the idea of projection or some other linguistic approaches. There is room for argument on how we should `select` the leaf node S to be substituted in Figure 1.8. Since elementary trees with non-anchored subtrees represent constructions requiring a specification beyond immediate dominance, such as *it-clefts* and *equative be*, this problem can be rephrased as the problem that which leaf node takes the most important syntactic role in the constructions and should be expanded for HPSG analysis. We can solve this problem by using the same idea as we have mentioned above, but for now we simply select a leaf node from the left most one.

4 Experiments

The algorithm was applied to the latest version of the XTAG English grammar (The XTAG Research Group 2001)[5], which is a large-scale FB-LTAG grammar for English. We successfully converted all the elementary trees[6] in the XTAG English grammar to HPSG lexical entries. Table 1.1 shows the classifications of elementary trees of the XTAG English grammar, according

[5]We used the grammar attached to the latest distribution of an LTAG parser which we used for the parsing experiment. The parser is available at ftp://ftp.cis.upenn.edu/pub/xtag/lem/lem-0.13.0.i686.tgz

[6]*Elementary trees* should be in fact denoted as *elementary tree templates*. That is, elementary trees are abstracted from lexicalized trees, and one elementary tree template is defined for one syntactic construction, which is assigned to a number of words.

Table 1.1: The classification of elementary trees in the XTAG English grammar (LTAG) and converted lexical entries corresponding to them (HPSG): $\mathcal{A}$: canonical elementary trees, $\mathcal{B}$: elementary trees violating only Condition 1, $\mathcal{C}$: elementary trees violating only Condition 2, $\mathcal{D}$: elementary trees violating both conditions

Grammar	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	**Total**
LTAG	326	764	54	50	1,194
HPSG	326	1,992	1,083	2,474	5,875

Table 1.2: The classification of elementary trees violating the conditions in Table 1.1: multi-anchored ones (corresponding to $\mathcal{B}$) (left), and ones with non-anchored subtrees (corresponding to $\mathcal{C} \cup \mathcal{D}$) (right)

Construction	**# of trees**
Compound expressions	414
Verb with PP	194
Idioms	140
Others	16
Total	**764**

Construction	**# of trees**
Verb with PP	85
It-cleft	12
Others	7
Total	**104**

to the conditions we introduced, and also shows the number of corresponding HPSG lexical entries.

We acquired exactly the same number of the derivation trees by using the original and the obtained grammar in the parsing experiment with 457 sentences from the ATIS corpus (Marcus, Santorini, and Marcinkiewicz 1994)[7] (the average length is 6.32 words). This result empirically attested the strong equivalence of our algorithm. The experimental result also shows that an efficient HPSG parser (Torisawa, Nishida, Miyao, and Tsujii 2000) achieves a drastic speed-up against the LTAG parser (Sarkar 2000). The speed-up is because of not only the careful implementation but also the difference of the parsing scheme in the HPSG and LTAG parsing (Yoshinaga, Miyao, Torisawa, and Tsujii 2001).

The left-hand side of Table 1.2 shows how multi-anchored elementary trees are used in the XTAG English grammar. The table shows they are mainly used for compound expressions or idioms. Although such expressions seem to be difficult to explain in the HPSG formalism, the obtained grammar can handle them with a proper size of lexical entries by dividing them into

[7] We eliminated 59 sentences because of a time-out of the parsers, and 61 sentences because the LTAG parser does not produce correct derivation trees because of bugs in its preprocessor.

multiple lexical entries. Another case for multi-anchored trees is *verb with PP*. The obtained grammar expresses this construction by cut-off nodes to require specified subtrees. On the other hand in linguistic specifications of HPSG, such a constraint is expressed by having the FORM feature, which requires the type of a phrase. This analysis seems to be consistent with the obtained grammar, i.e., the LTAG analysis.

The right-hand side of Table 1.2 shows the usage of elementary trees with non-anchored subtrees. These elementary trees express constructions requiring specifications beyond immediate dominance. As we can see in Table 1.1, these trees are expanded to quite large number of lexical entries. This result expects that these constructions might be difficult to handle in the HPSG formalism. The major case of such constructions is *verb with PP*, which causes a Wh-extraction of NP from PP. Since this construction is explained by using SLASH feature in HPSG, the difference between the formalisms appears here. Another case is *it-cleft*, which is not obvious to handle in HPSG, and it might be worth discussing.

5 Conclusion

We proposed a grammar conversion from an FB-LTAG grammar into an HPSG-style grammar. The grammar conversion guarantees the strong equivalence, and hence we can obtain parsing results of an LTAG grammar from those of the obtained HPSG-style grammar. Thus, our grammar conversion enables to share LTAG resources in the HPSG community, to apply HPSG parsing techniques to LTAG grammars, and to clarify the difference of linguistic analysis in both grammar formalisms. We implemented this algorithm, and successfully converted the latest version of the XTAG English grammar. Although we did not give a formal proof of strong equivalence, it will be possible owing to the formulation of our conversion algorithm.

Acknowledgment The authors wish to thank Mr. Anoop Sarkar for his help in using his parser in our experiment. The authors are also indebted to two anonymous reviewers for their valuable comments on this paper.

Bibliography

Abeillé, A. and M.-H. Candito (2000). Evolution of the XTAG system. In A. Abeillé and O. Rambow (Eds.), *Tree Adjoining Grammars: Formal, Computational and Linguistic Aspects*. CSLI publications.

Becker, T. and P. Lopez (2000). Adapting HPSG-to-TAG compilation to wide-coverage grammars. In *Proceedings of the 5th of International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+5)*, pp. 47–54.

Carpenter, B. (1992). *The Logic of Typed Feature Structures*. Cambridge University Press.

Doran, C., B. A. Hockey, A. Sarkar, B. Srinivas, and F. Xia (2000). FTAG: A Lexicalized Tree Adjoining Grammar for French. In A. Abeillé and O. Rambow (Eds.), *Tree Adjoining Grammars: Formal, Computational and Linguistic Aspects*. CSLI publications.

Flickinger, D., S. Oepen, and J. Tsujii (Eds.) (2000). *Natural Language Engineering – Special Issue on Efficient Processing with HPSG: Methods, Systems, Evaluation*. Cambridge University Press.

Kasper, R., B. Kiefer, K. Netter, and K. Vijay-Shanker (1995). Compilation of HPSG to TAG. In *Proceedings of 33rd Meeting of the Association for Computational Linguistics (ACL '94)*, pp. 92–99.

Kay, M., J. Gawron, and P. Norvig (1994). *Verbmobil: A Translation System for Face-to-Face Dialog*. CSLI Publications.

Kroch, A. (1989). Asymmetries in long-distance extraction in a Tree-Adjoining Grammar. In M. Baltin and A. Kroch (Eds.), *Alternative Conceptions of Phrase Structure*. University of Chicago Press.

Marcus, M., B. Santorini, and M. A. Marcinkiewicz (1994). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics 19*(2), 313–330.

Pollard, C. and I. A. Sag (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications.

Sarkar, A. (2000). Practical experiments in parsing using Tree Adjoining Grammars. In *Proceedings of the 5th of International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+5)*, pp. 193–198.

Schabes, Y., A. Abeille, and A. K. Joshi (1988). Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. In *Proceedings of 12th International Conference on Computational Linguistics (COLING '92)*, pp. 578–583.

Schabes, Y. and S. M. Shieber (1994). An alternative conception of tree-adjoining derivation. *Computational Linguistics 20*(1), 91–124.

Tateisi, Y., K. Torisawa, Y. Miyao, and J. Tsujii (1998). Translating the XTAG English grammar to HPSG. In *Proceedings of the 4th of International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pp. 172–175.

The XTAG Research Group (2001). A Lexicalized Tree Adjoining Grammar for English. http://www.cis.upenn.edu/~xtag/.

Torisawa, K., K. Nishida, Y. Miyao, and J. Tsujii (2000). An HPSG parser with CFG filtering. *Natural Language Engineering – Special Issue on Efficient Processing with HPSG: Methods, Systems, Evaluation 6*(1), 63–80.

Vijay-Shanker, K. (1987). *A Study of Tree Adjoining Grammars*. Ph. D. thesis, Department of Computer & Information Science, University of Pennsylvania.

Vijay-Shanker, K. and A. K. Joshi (1988). Feature structures based Tree Adjoining Grammars. In *Proceedings of 12th International Conference on Computational Linguistics (COLING '92)*, pp. 714–719.

Yoshinaga, N., Y. Miyao, K. Torisawa, and J. Tsujii (2001). Efficient LTAG parsing using HPSG parsers. In *Proceedings of Pacific Association for Computational Linguistics (PACLING 2001)*. To appear.