Grammar conversion from LTAG to HPSG

Naoki Yoshinaga Yusuke Miyao
Department of Information Science, Graduate School of Science
University of Tokyo
yoshinag@is.s.u-tokyo.ac.jp yusuke@is.s.u-tokyo.ac.jp

Abstract

We propose an algorithm for the conversion of grammars from an arbitrary FB-LTAG grammar into a strongly equivalent HPSG-style grammar. Our algorithm converts LTAG elementary trees into HPSG feature structures by encoding the tree structures in stacks. A set of pre-determined rules manipulate the stack to emulate substitution and adjunction. We have used our algorithm to obtain HPSG-style grammars from existing LTAG grammars. We apply this algorithm to the XTAG English grammar and report some of our findings.

1 Introduction

Recent applications of NLP (Kay et al. 1994; Carroll et al. 1998) have taken advantage of computationally and linguistically motivated approaches to the formalization of grammars, such as the Feature-Based Lexicalized Tree Adjoining Grammar (FB-LTAG[1]: Vijay-Shanker 1987; Vijay-Shanker and Joshi 1988) and the Head-Driven Phrase Structure Grammar (HPSG: Pollard and Sag 1994). Discussion of the correspondences between the two formalisms has accompanied their development; i.e., their linguistic relationships and differences have been investigated (Abeillé 1993; Kasper 1998), as has conversion between two grammars in the two formalisms (Kasper et al. 1995; Tateisi et al. 1998; Becker and Lopez 2000). These ongoing efforts towards *collaboration* have contributed greatly to the development of the two formalisms, and we believe that it is worth continuing in this direction.

This research is thus intended to provide a basis for further collaboration between the communities, and is built around a proposed method of conversion from LTAG to HPSG. This differs from previous methods of conversion in that it guarantees *strong equivalence*[2]; that is, the results of parsing (derivation trees) by an LTAG grammar can be derived from those of the obtained

[1] Unless otherwise noted, we use the term LTAG to refer to FB-LTAG in this paper.

[2] Chomsky first introduced the notion of strong equivalence between grammars, where both grammars generate the same set of structural descriptions (e.g., parse trees: Chomsky (1963)). Kornai and Pullum (1990) and Miller (1999) used the notion of isomorphism between sets of structural descriptions to provide the notion of a strong equivalence across grammar formalisms which we have adopted in this research.

HPSG-style grammar and vice versa. Having strongly equivalent grammars based on different formalisms is valuable for both communities from practical, computational, and theoretical points of view in the following way (Becker and Lopez 2000):

Sharing of resources HPSG-based applications can make use of LTAG resources (lexicons and grammars) such as the large-scale English (Doran et al. 2000) and French (Abeillé and Candito 2000) grammars that have been developed. Our method of conversion can reduce the considerable workload involved in developing large-scale resources from scratch. In this paper, we report on the conversion of a large-scale LTAG grammar.

Parsing comparison Having strongly equivalent grammars allows us to compare two parsers based on two different formalisms. Although most researchers believe that the HPSG parsers are inefficient as compared to LTAG parsers from the viewpoint of the theoretical bounds of worst time complexity, no results have been reported for empirical comparisons of the respective levels of time complexity. This is because a comparison can only be meaningful if it is based on strongly equivalent grammars, which had not been available.

Linguistic correspondence We are also able to explore the linguistic correspondences between formalisms. Since the HPSG-style grammar obtained by our method has both the computational architecture that underlies HPSG and the linguistic specifications that were given in the original LTAG, their difference will be made apparent by comparing the obtained grammar with existing HPSG grammars (Kay et al. 1994; Flickinger 2000).

We used the conversion algorithm which we implemented to successfully convert the latest version of the XTAG English grammar (The XTAG Research Group 2001), which is a large-scale FB-LTAG grammar, into an HPSG-style grammar. Strong equivalence between the original and obtained grammars was empirically attested to by the fact that parsing with these grammars generated equivalent results. Strong equivalence of grammars allows the fair comparison of LTAG and HPSG parsers. As an aside, Yoshinaga et al. (2001) have reported on an efficient HPSG parser that achieved a significantly higher speed of parsing than an existing LTAG parser. In this paper, we investigated the types of linguistic phenomena covered by the XTAG English grammar, and the correspondence to their analysis in the HPSG formalism.

The existing works do not have the above three advantages, which are only attainable when strong equivalence is preserved. Tateisi et al. (1998) have translated the same LTAG into an HPSG. However, their method depended on a translator's intuitive analysis of the original grammar, and the translation was thus manual and grammar dependent. The manual translation demanded considerable efforts on the part of the translator, and this obscures the existence or non-existence of strong equivalence between the original and obtained grammars. Other work has involved the conversion of HPSG into LTAG (Kasper et al. 1995; Becker and Lopez 2000). However, given the greater generative power of HPSG, the conversion required that some restrictions be placed on the input HPSG to suppress its generative capacity. Moreover, Becker and Lopez discussed that there was overgeneration and undergeneration of the LTAG grammars. Thus the results of conversion do not have strong equivalence.

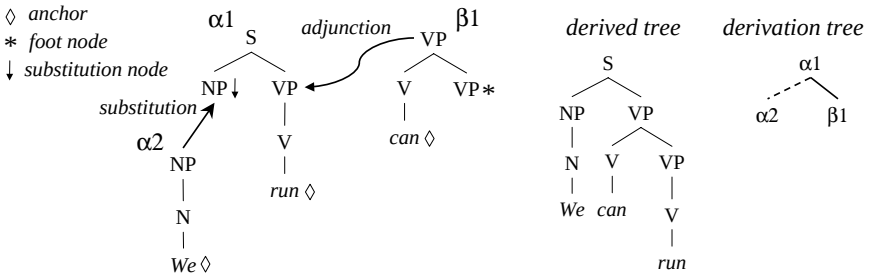


Figure 1: Tree Adjoining Grammar: basic structures and the compose operations used to them

2 Grammar formalisms

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

LTAG (Schabes et al. 1988), an input of our algorithm, provides syntactic analysis of a sentence by using two operations, called *substitution* and *adjunction*, to compose *elementary trees*. This is shown on the left-hand side of Figure 1. An elementary tree has at least one leaf node that is labeled with a terminal symbol (i.e., word) called *an anchor* (marked with $\diamond$). Elementary trees are classified into two types, *initial trees* ($\alpha 1$ and $\alpha 2$) and *auxiliary trees* ($\beta 1$). The label of one leaf node of an auxiliary tree is identical to that of its root node and this is specially marked (here, with $*$) as *a foot node*. In an elementary tree, leaf nodes other than anchors and a foot node are called *substitution nodes* (marked with $\downarrow$).

The left-hand side of Figure 1 illustrates the two operations. In substitution, a leaf node (substitution node) is replaced by an initial tree, while in adjunction, an auxiliary tree with the root node and a foot node labeled x is grafted onto a node with the same symbol x. The results of analysis are described not only by *derived trees* (i.e., parse trees) but also by *derivation trees* (the right-hand side of Figure 1). Derivation trees represent the history of combinations of trees and are the upper-level structural descriptions of LTAG.

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

2.2 Head-Driven Phrase Structure Grammar (HPSG)

We define an HPSG-style grammar, the form of the output of our algorithm, according to the computational architecture which underlies HPSG (Pollard and Sag 1994). It consists of *lexical entries* and *ID grammar rules*, each of which is described with typed feature structures (Carpenter 1992). The greater generative power of the underlying architecture of HPSG allows us to obtain a trivial encoding of an LTAG in HPSG, as described by Keller (1994). However, such a

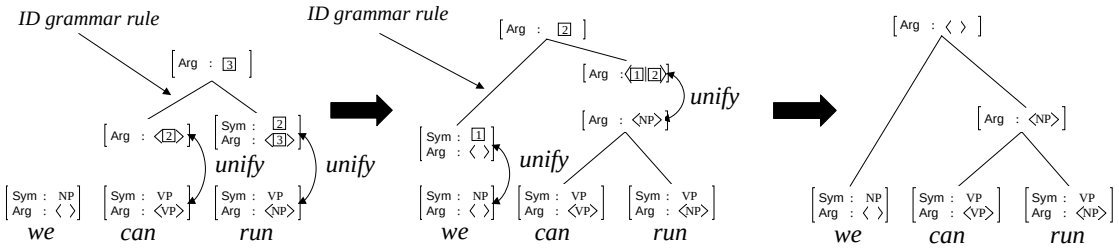


Figure 2: Parsing with an HPSG grammar

conversion cannot answer our purpose because the obtained grammar is far from the one defined by Pollard and Sag (1994). Hence, we restrict the form of an HPSG-style grammar to one which follows the HPSG formalism in its lexicalist framework in the following way. A lexical entry for a word must express the characteristics of the word, such as its subcategorization frame and grammatical category. An ID grammar rule must represent the constraints on the configuration of immediate constituency, and not be a construction-specific rule specified by lexical characteristics. For example, in Figure 2, the boxed numericals '$\boxed{n}$' called *tags* express the sharing of common values between the two sub-feature structures. Note that Pollard and Sag (1994) provide detailed linguistic specifications for the form of feature structures and adopt *principles*, such as *the Head Feature Principle*, to depict linguistic properties. In our definition, we assume that such principles are encoded in ID grammar rules, and do not give further linguistic specifications. The above restrictions enable us to not only define a formal link between an LTAG and an HPSG-style grammar, both of which are lexicalist frameworks, but also clarify the relationships between an HPSG-style grammar and HPSG. We will address this point later, in Section 3.7.

Figure 2 illustrates an example of bottom-up parsing with an HPSG-style grammar. In the HPSG approach, as we can see in this example, a parse tree is generated by incrementally applying ID grammar rules to lexical entries and constructing each of the branching structures one by one. Thus, the key points in the conversion are 1) how to encode the tree structure of an elementary tree as an HPSG lexical entry, and 2) how to emulate substitution and adjunction by ID grammar rules. Note that there is no one-to-one correspondence between an elementary tree and an HPSG lexical entry because one elementary tree can have multiple anchors (i.e., words) to represent compound expressions.

3 The algorithm

This section describes our conversion algorithm in detail. As noted in Section 2, the tree structures of LTAG elementary trees should be encoded in HPSG lexical entries, and substitution and adjunction should be emulated by ID grammar rules. Thus, we propose a conversion algorithm which consists of: 1) the conversion of elementary trees into HPSG lexical entries and 2) the

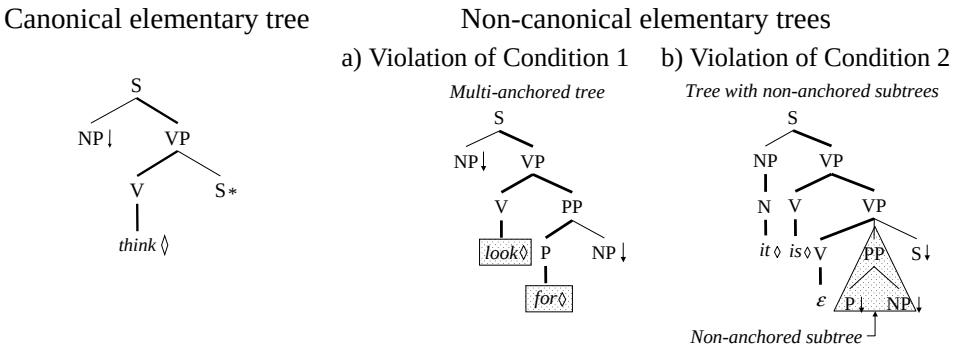


Figure 3: A canonical elementary tree and non-canonical elementary trees

emulation of substitution and adjunction by pre-determined ID grammar rules.

In the following description, we start by defining *canonical elementary trees*, which have a one-to-one correspondence with HPSG lexical entries.[3]

Definition 3.1 (Canonical elementary tree) *Canonical elementary trees are elementary trees which satisfy the conditions below:*

> *Condition 1: A tree has only one anchor.*
> *Condition 2: Every branching structure in a tree contains trunk nodes.*

Trunk nodes are nodes on *a trunk* which is a path from an anchor to the root node (the thick lines in Figure 3) other than the anchor (Kasper et al. 1995). Conditions 1 and 2 respectively guarantee that a canonical elementary tree has only one trunk and that each branching consists of a trunk node, a leaf node, and their mother which is also a trunk node, as seen in the example on the left-hand side of Figure 3. The right-hand side of Figure 3 shows non-canonical trees. We call a subtree of depth $n(\geq 1)$ that includes no anchor *a non-anchored subtree*. Non-canonical elementary trees are converted to canonical trees before conversion into HPSG lexical entries by the algorithm in the next section.

3.1 Conversion of canonical elementary trees

We can directly convert canonical elementary trees to HPSG lexical entries in the way shown in Figure 4. The procedure `convert_tree_into_lexical_entry` in Figure 4 presents an algorithm for converting a canonical elementary tree T into an HPSG lexical entry L. In the algorithm, *arg* is a stack of branchings b_i as described by 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

[3]In this paper, we discuss the conversion of elementary trees which consist of binary branching structures. A unary branching can be regarded as a binary branching whose sibling is the empty category, and n-ary ($n \geq 3$) branchings can be converted into binary branchings. Strong equivalence before and after the conversion is guaranteed, because the conversion are according to a one-to-one (i.e., isomorphic) mapping.

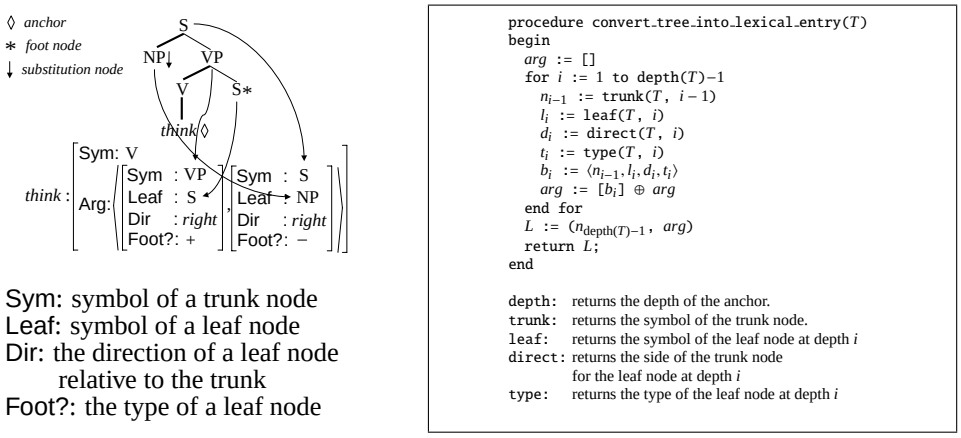


Figure 4: An algorithm for converting a canonical elementary tree T to an HPSG lexical entry L

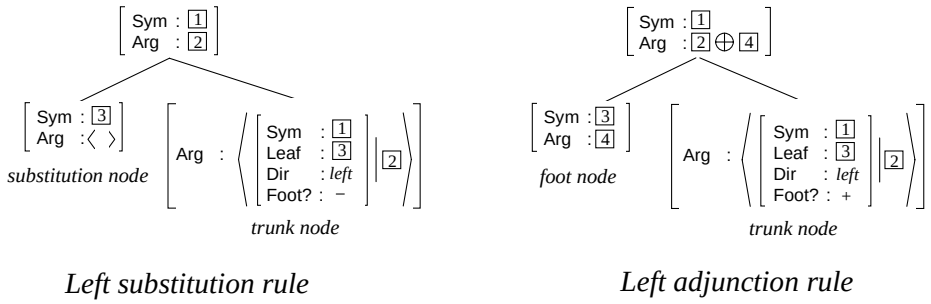


Figure 5: Grammar rules: the substitution rule and adjunction rule

and t_i represent the leaf node at a depth i; respectively, they represent the non-terminal symbol, the direction (the side of the trunk node n_i on which the leaf node is), and the type (whether the node is a foot node or a substitution node). We call this stack *arguments* of the word. Finally, the converted lexical entry L is described by the arguments *arg* and the mother of the anchor, namely, $n_{\texttt{depth(T)}-1}$ where `depth(T)` is the depth of the tree T. In the left-hand side of Figure 4, the value of the Sym feature is the symbol $n_{depth(T)-1}$ and the value of the Arg feature contains the arguments in *arg*, as a stack of feature structures with the four features Sym, Leaf, Dir and Foot?, which correspond to n_{i-1}, l_i, d_i and t_i, respectively. In the recognition process, the parser pops an element from the Arg feature to select a node that is unifiable with that element. It follows that the node with an empty stack as its Arg feature corresponds to the root node of the initial tree.

3.2 Definition of grammar rules

In this section, we provide the definition of the grammar rules which emulate substitution and adjunction respectively and are thus called *substitution rules* and *adjunction rules* (in Figure 5).

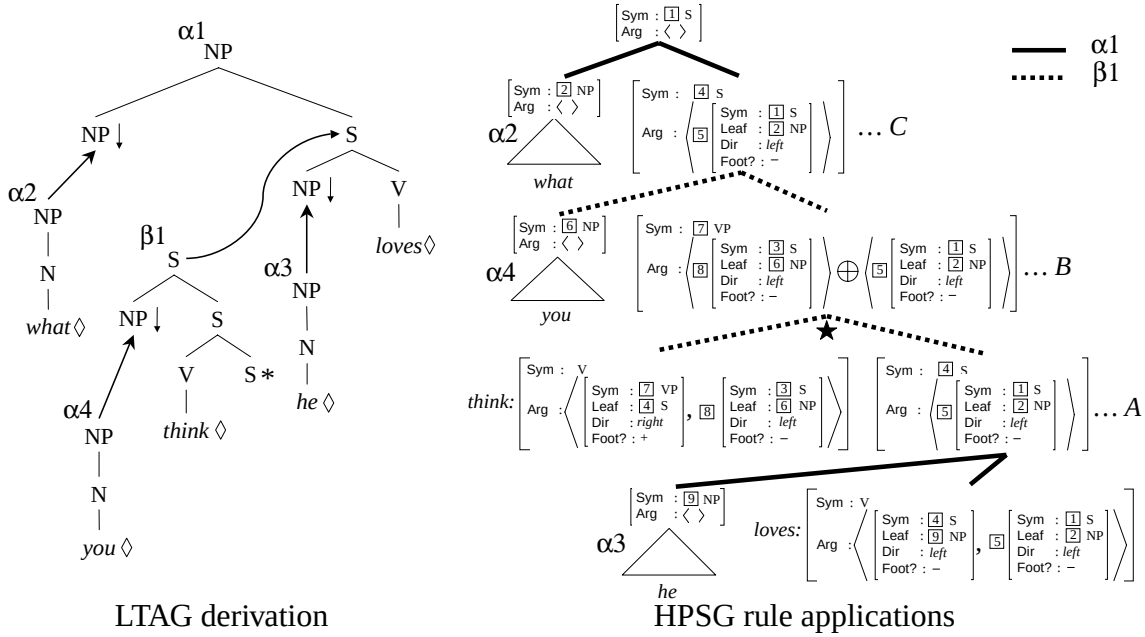


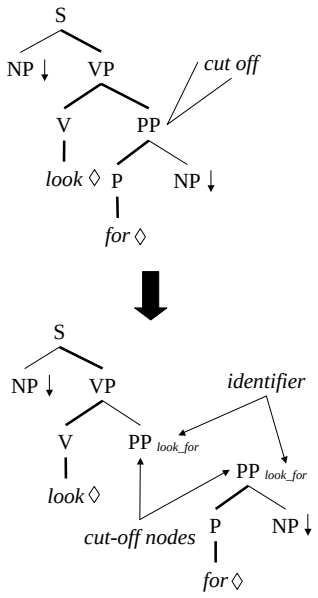
Figure 6: LTAG derivation and HPSG rule applications for the phrase "*what you think he loves*"

In the figure, we give rules for the case where right-hand daughters correspond to the trunk nodes. Of course, there are symmetric rules for the left-hand case. These rules are independent of the input LTAG because they do not specify any given characteristics for the LTAG.

Substitution rule: The Sym feature of the node to which is applied substitution must be identical to the Leaf feature '3' of the trunk node. The substitution rule percolates the tail elements '2' of the Arg feature of the trunk node to the mother in order to continue constructing the tree. The value of the Arg feature of the node for substitution must be an empty stack $\langle\ \rangle$, because this node must be unified only with the node that corresponds to the root node of the initial tree. The value "−" or "+" of the Foot? feature explicitly determines whether the next rule to be applied is the substitution rule or the adjunction rule.

Adjunction rule: The Sym feature of a foot node must be identical to the Leaf feature '3' of the trunk node. The value of the Arg feature of the mother node is a concatenated stack of the Arg features, '2' and '4', of both of its daughters. This allows the parser to construct the tree which corresponds to the adjoining tree and then to continue constructing the tree which corresponds to the adjoined tree.

Figure 6 shows examples of rule application. The thick lines indicate the adjoined tree ($\alpha 1$) and the dashed lines indicate the adjoining tree ($\beta 1$). The adjunction rule is applied to construct




```
procedure divide_tree_into_subtrees(MT)
begin
  if (number(MT) = 0)
    return {MT}
  else
  A := select(MT)
  ⟨CT,T⟩ := divtree(MT, A)                                  ··· (1)
  foreach T (T)
    SST := divide_tree_into_subtrees(T)
    ST := SST ∪ ST
  end foreach
  ST := ST ∪ {CT}
  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
  CT := MT
  return ⟨CT,T⟩
end
```

`select:`	returns one of anchors (default: the left-most one).
`depth:`	returns the depth of the anchor.
`trunk:`	returns the trunk node at depth *i*
`arg:`	returns the sibling node of the trunk node.
`cut:`	cuts off the tree at the sibling node of the trunk node and returns a subtree whose root node is the sibling node.
`nonleaf:`	returns true if not a leaf node.
`address:`	returns address in the elementary tree.
`mark:`	marks an address for each cut-off node.

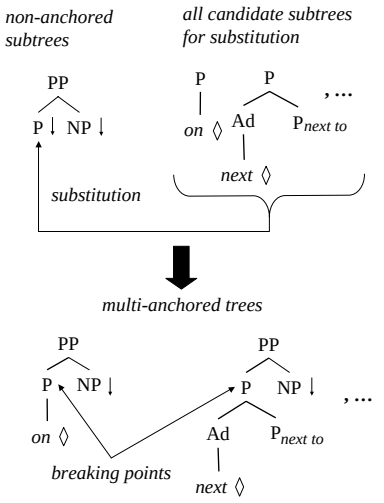
Figure 7: An algorithm for dividing a non-canonical elementary tree MT into a set of subtree $\mathcal{ST}$, each of which has at most one anchor.

the branching marked with ⋆, where "*think*" takes as its argument the node whose Sym feature's value is S. By applying the adjunction rule, the Arg feature of the mother node B becomes a concatenated stack of the Arg features of both $\beta 1$, '8', and $\alpha 1$, '5.' Note that when the construction of $\beta 1$ is completed, the Arg feature of the trunk node C will return to its former state A. We can continue constructing $\alpha 1$ in the same way as for the case where no adjunction had been applied.

3.3 Division of multi-anchored trees

Non-canonical elementary trees are initially divided into multiple subtrees, each of which has at most one anchor, as is shown in Figure 7. 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 the co-occurrence relation among the multiple anchors.

The procedure `divide_tree_into_subtrees` in Figure 7 represents an algorithm for dividing a non-canonical elementary tree MT into a set of subtrees $\mathcal{ST}$. It starts by selecting one anchor A, and the single-anchored tree CT of that anchor A, which consists of the trunk nodes




```
procedure expand_tree_into_anchored_tree(NT)
begin
  BR := na_br(NT)
  MT := {NT}
  foreach BR (BR)
    S := select(BR)                                   ... (1)
    IT := initial(S)
    MMT := φ
    foreach MT (MT)
      TMT := substitute(MT, S, IT)                    ... (2)
      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 candidate trees whose root node is the same as the leaf node.
`select:`	returns one of leaf nodes (default: the left-most one).
`substitute:`	causes substitution to the leaf node and returns a set of resulting multi-anchored trees.
`mark:`	marks the substituted node

Figure 8: An algorithm for converting a non-anchored subtree NT to a set of multi-anchored trees $\mathcal{MT}$

and their sibling nodes, is then picked up (`(1)` in Figure 7). We traverse the path from the root node to the anchor *A*, cut off the sibling node[4] `arg(trunk(`*i*`))` if it is not a leaf node, and store *the address* of the elementary tree in the cut-off node as an identifier (`(2)` and `(3)` in Figure 7).

Using this algorithm, multi-anchored elementary trees which only violates Condition 1 are converted into canonical trees, while trees with non-anchored subtrees are converted into canonical trees and non-anchored subtrees, which violate Condition 2. Non-anchored subtrees are converted into canonical trees by the algorithm given in the next section.

3.4 Substitution in non-anchored subtrees

The non-canonical elementary trees which violate Condition 2 have non-anchored subtrees. These non-anchored subtrees are first extracted by the algorithm in the previous section, and are then converted into multi-anchored trees by substituting a substitution node on the deepest branching by every candidate tree for substitution. The candidate trees for the application of this process are selected from among the all canonical elementary trees and the ones obtained by the algorithm in the previous section. Substituted nodes are marked as *breaking points* to record the origination of these nodes. Note that non-anchored subtrees are not selected as candidates for substitution, because their root nodes originate from internal nodes of the elementary trees. This guarantees that the multi-anchored trees obtained by this process will satisfy Condition 2.

[4]We should mention that the path from the foot node to the root node (*spine*: Kasper et al. 1995) 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 trees.

These trees can be converted into single-anchored trees, to which we can apply the algorithm in Section 3.1, by the algorithm in the previous section.

The procedure `expand_tree_into_anchored_tree` in Figure 8 represents an algorithm for converting a non-anchored subtree NT into multi-anchored trees $\mathcal{MT}$. For each branching structure that consists of substitution nodes or foot nodes, one substitution node S is selected (`(1)` in Figure 8). The function `substitute` applies a substitution to the node S of every candidate tree which is substitutable for S (`(2)` in Figure 8).

3.5 Strong equivalence

In this section, we discuss how our algorithm guarantees strong equivalence between the grammar it obtains and the original grammar. In the obtained grammar, the grammar rules are applied only to those feature structures which correspond to nodes which are substitutable for/adjoinable with the canonical elementary trees of the original LTAG because the branchings encoded in the respective values of Arg specify the nodes to be subcategorized next. Strong equivalence also holds for the conversion of non-canonical elementary trees. For trees that violate Condition 1, we can distinguish the cut-off nodes from substitution nodes thanks to the identifiers, which allow recovery of the co-occurrence relation between the divided trees. For trees that violate Condition 2, we can identify those nodes to which substitution is to be applied in a combined tree because they are marked as breaking points, and thus consider the combined tree as two trees in the LTAG derivation. We can thus avoid overgeneration by having the identifiers checked in the substitution rules, and avoid undergeneration by substituting *all* candidate trees for substitution nodes in the algorithm in Section 3.4.

Strong equivalence enables us to recover an LTAG derivation tree from an HPSG parse tree by following the history of rule applications and mapping each of them to substitution or adjunction. Let us take the case of Figure 6 as an example. We start by following the trunk node when the substitution rule was applied, or the foot node when the adjunction rule was applied. We then reach "*love*," and recognize it as the anchor of an elementary tree whose root node is identical to that of the parse tree. We then follow the path from the anchor to the root node to recognize $\alpha 1$ and combinations between $\alpha 1$ and other elementary trees. Since we start by finding an application of the substitution rule, we can map it to the substitution of $\alpha 3$ to $\alpha 1$ by recognizing the sibling node of the trunk node as the root node of $\alpha 3$ and by recursively recovering the partial derivation from the sibling subtree. Then, the next rule is the adjunction rule (marked with $\star$), and we find that the node A takes adjunction. We thus remember the length of the value of the Arg feature of the node A, and follow the trunk with handling rule applications as ones for the adjoining tree $\beta 1$ until the length of the Arg feature is equal to that for the node A. This is the case at the node C. This implies that the construction of the adjoining tree $\beta 1$ is completed at the node C. We restart the recognition of $\alpha 1$. After handling another application of the substitution rule, we reach the root node S, and complete the recognition of $\alpha 1$ and thus the whole derivation tree.

3.6 Extension to FB-LTAG

The above algorithms produce the conversion of an LTAG, and are easily extensible to handle an FB-LTAG grammar by merely storing a feature structure for each node, together with the symbol, in the Sym feature and Leaf feature. The grammar rules are also extended to execute the feature structure unification done in FB-LTAG.

3.7 Correspondence between an HPSG-style grammar and HPSG

The above algorithms provide a formal link between HPSG and LTAG via an HPSG-style grammar. In this section, we discuss the linguistic correspondence between an HPSG-style grammar and HPSG according to the syntactic *head*, which is a central notion in HPSG. In the following discussion, we assume that the reader is familiar with HPSG (Pollard and Sag 1994).

In order to derive HPSG from an HPSG-style grammar that we have obtained, we must elaborate on the *sign* of the HPSG-style grammar. HPSG provides a modular specification of linguistic generalization by using *principles* and *rule schemata* in the context of the lexicalist framework.[5] On the other hand, our HPSG-style grammar implicitly captures some of the principles and ID schemata of the definition in Section 2.2 in the following way. *The Immediate Dominance Principle* is satisfied by the use of the ID grammar rules. In the ID grammar rules, *the Subcategorization Principle* is expressed by the structure-sharing of the Sym and Leaf features which correspond to the HEAD feature in HPSG. We should notice that a non-empty value for the Arg feature of the foot node in the adjunction rules roughly corresponds to the SLASH feature in HPSG, which supports *the Nonlocal Feature Principle*. The Arg feature thus corresponds to concatenation of the SUBCAT and SLASH features of HPSG. Other principles, such as *the Head Feature Principle*, are implicitly encoded in the original FB-LTAG. We will extract such principles in the LTAG context and subdivide the ID grammar rules into HPSG rule schemata by analyzing feature percolation (Tateisi et al. 1998). We should mention the following two issues in order to elaborate an HPSG-style grammar.

The distinction between predicative and modifier auxiliary trees We should mention that auxiliary trees in LTAG are of a predicative or a modifier type (Kroch 1989; Schabes and Shieber 1994). The former introduces a predicate that subcategorizes for a phrase of the category of its foot node, while the latter introduces a modifying, dislocated phrase, or a complement. This distinction is, in rough terms, made by determining which daughter is the head, a foot node or a trunk node. Tateisi et al. (1998) distinguished these trees by manually analyzing feature percolation in auxiliary trees and by assigning HPSG rule schemata separately to each auxiliary tree. In this paper, we consider that all trunk nodes are heads, that is, treat all auxiliary trees as predicative trees, but we can manually or semi-automatically determine the above categories by providing some linguistic cues or by analyzing feature percolation.

[5] Note that HPSG-specific linguistic theories such as binding theory must be implemented in the obtained HPSG-style grammar by defining additional features or special mechanisms.

Table 1: The classification of elementary trees in the XTAG English grammar (LTAG) and lexical entries converted from (HPSG): $\mathcal{A}$: canonical elementary trees, $\mathcal{B}$: trees that are non-canonical only because they violate Condition 1, $\mathcal{C}$: violate Condition 2, $\mathcal{D}$: trees that violate 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

Head selection How we should implement the function `select` in Figure 7 that selects the anchor A is not entirely clear. 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 the problem of which word of a phrase is the syntactic/semantic head. We must also consider a similar issue to do with how we should implement the function `select` in Figure 8 that selects the leaf node S to be substituted. Since elementary trees with non-anchored subtrees represent constructions that require a specification beyond immediate dominance, such as *it-clefts* and *equative be*, this problem may be rephrased as one of finding which leaf node takes the dominant syntactic role and should be substituted in carrying out HPSG analysis. We currently simply select an anchor or a substitution node from the left-most node, though we can solve these problems by using linguistic ideas such as projection.

4 Experiments

We applied the algorithm to the latest version of the XTAG English grammar (The XTAG Research Group 2001),[6] a large-scale FB-LTAG grammar for English. We successfully converted all elementary trees[7] in the XTAG English grammar to HPSG lexical entries. Table 1 shows the classification of the elementary trees of the XTAG English grammar according to the conditions we introduced in Section 3. The table also shows the number of corresponding HPSG lexical entries.

In the experiment with the parsing of 457 sentences from the ATIS corpus[8] (Marcus et al. 1994: the average length is 6.32 words), we acquired exactly the same number of derivation trees by using the original and obtained grammars. This result is empirical attestation of the strong equivalence of our algorithm. We had thus made the XTAG English grammar available

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

[7]*Elementary trees* should in fact be denoted as *elementary tree templates*. That is, elementary tree templates are abstracted from lexicalized trees, which defines one syntactic lexicon.

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

Table 2: The classification of the non-canonical elementary trees in Table 1: multi-anchored trees (corresponding to $\mathcal{B}$) (left), and trees 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 HPSG community by converting the grammar to HPSG-style. In the experimental result, we see that the efficient HPSG parser (Torisawa et al. 2000) achieved a speed in parsing that was higher by a factor of 25.5 than the LTAG parser (Sarkar 2000). We further investigated the reason for this speed-up in an earlier work (Yoshinaga et al. 2001) where we obtained the following results. Contrary to expectations from the viewpoint of the theoretical bound of worst time complexity, we showed that the empirical time complexity of the HPSG parser is lower than that of the LTAG parser. By inspecting the differences between the algorithms of both parsers, we finally concluded that suppressing the ambiguity of partial parse trees by the operation called *factoring* is the most important factor in achieving high performance in parsing, and that existing HPSG parsers are more beneficial than LTAG parsers in this respect.[9]

The left-hand side of Table 2 shows how multi-anchored elementary trees are used in the XTAG English grammar. The table shows that they are mainly used for compound expressions or idioms. Although such expressions seem to be difficult to explain in terms of the HPSG formalism, the obtained grammar is able to handle them when they are divided into multiple lexical entries. Another case of multi-anchored trees is *the multi-anchor PP complement verb*, whose prepositional node is anchored. Such a case is shown in the center of Figure 3. The obtained grammar expresses this construction by cut-off nodes to require specified subtrees. In linguistic specifications in terms of HPSG, on the other hand, such a constraint is expressed by having a PFORM feature, which takes values that represent the type of the corresponding prepositional phrase. This analysis seems to be consistent with the obtained grammar, that is, the LTAG analysis.

[9]The compaction of substructures in elementary trees has been asserted to have a great impact on performance in parsing (Evans and Weir 1998; Lopez 2000). In this research, several elementary trees for each word were converted into finite-state automata, and merged into a single finite-state automaton. In the HPSG parsers, some of this compaction is dynamically executed by factoring. Furthermore, the factoring method enables another kind of compaction which merges the equivalent edges headed by different words. Automaton-based compaction is incapable of performing this kind of compaction, since their techniques are basically closed in elementary trees for each word.

The right-hand side of Table 2 shows the usages represented by elementary trees with non-anchored subtrees. These elementary trees express constructions requiring specifications beyond immediate dominance. As we can see in Table 1, these trees are expanded to include quite a large number of lexical entries. This result leads us to expect that these constructions might be difficult to handle in the HPSG formalism. The major case of such constructions is *the PP complement verb*, which has the expanded PP structure (see *it-cleft* in the right-hand side of Figure 3). This construction allows the extraction of the object of the preposition, and expresses the verb taking the object of the preposition as its argument rather than taking the whole PP. In the HPSG formalism, on the other hand, this NP extraction is explained by using the SLASH feature, and the predicate-argument relation between the verb and the object of the preposition is expressed by linking the argument in the predicative-argument structure (the CONTENT feature) with the object of the preposition. This construction is hence explained differently in LTAG and HPSG. Further case is the *it-cleft*. It is not clear how to handle this in the HPSG formalism, which might be a subject worth discussing.

5 Conclusion

We have proposed a method for grammar conversion from an FB-LTAG grammar into an HPSG-style grammar. Strong equivalence between the original and converted grammars is guaranteed; hence, we can obtain the parsing results of an LTAG grammar from the parsing results of the HPSG-style grammar obtained by conversion. Our method thus enables the sharing of LTAG resources with the HPSG community, the application of HPSG parsing techniques to LTAG grammars, and the clarification of the differences between linguistic analysis according to the two grammar formalisms. We implemented this algorithm and obtained the following results. (1) We used our grammar conversion algorithm to successfully convert the latest version of the XTAG English grammar; that is, we made the XTAG English grammar available to the HPSG community. (2) We used the strong equivalence of the grammars to perform the first ever fair comparison of the performance of each of the approaches in parsing, and showed the relative efficiency of an HPSG parser. (3) By analyzing the obtained HPSG-style grammar, we observed the difference between the two formalisms. We are going to formally prove the strong equivalence between the two grammars. This will be possible because of the formulation of the conversion algorithms.

Acknowledgment The authors wish to thank Anoop Sarkar for his help in applying his parser as part of our experiments. The authors are also indebted to the two anonymous reviewers for their valuable comments on and criticisms of this paper. Special thanks to Kristina Striegnitz for her good work since the Sixth ESSLLI Student Session.

References

Abeillé, A. (1993). *Les nouvelles syntaxes: grammaires d'unification et analyse du français*. Armanda Colin. in French.

Abeillé, A. and M.-H. Candito (2000). FTAG: A Lexicalized Tree Adjoining Grammar for French. In A. Abeillé and O. Rambow (Eds.), *Tree Adjoining Grammars: Formal, Computational and Linguistic Aspects*, pp. 305–329. CSLI publications.

Becker, T. and P. Lopez (2000). Adapting HPSG-to-TAG compilation to wide-coverage grammars. In *Proc. of TAG+5*, pp. 47–54.

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

Carroll, J., N. Nicolov, O. Shaumyan, M. Smets, and D. Weir (1998). The LEXSYS project. In *Proc. of TAG+4*, pp. 29–33.

Chomsky, N. (1963). Formal properties of grammar. In R. D. Luce, R. R. Bush, and E. Galanter (Eds.), *Handbook of Mathematical Psychology*, Volume II, pp. 323–418. John Wiley and Sons, Inc.

Doran, C., B. A. Hockey, A. Sarkar, B. Srinivas, and F. Xia (2000). Evolution of the XTAG system. In A. Abeillé and O. Rambow (Eds.), *Tree Adjoining Grammars: Formal, Computational and Linguistic Aspects*, pp. 371–403. CSLI publications.

Evans, R. and D. Weir (1998). A structure-sharing parser for lexicalized grammars. In *Proc. of COLING–ACL '98*, pp. 372–378.

Flickinger, D. (2000). On building a more efficient grammar by exploiting types. *Natural Language Engineering 6*(1), 15–28.

Kasper, R. (1998). TAG and HPSG. Talk given in the tutorial session at TAG+ 4.

Kasper, R., B. Kiefer, K. Netter, and K. Vijay-Shanker (1995). Compilation of HPSG to TAG. In *Proc. of ACL '95*, pp. 92–99.

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

Keller, B. (1994). *Feature Logics, Infinitary Descriptions and Grammar*. CSLI publications.

Kornai, A. and G. K. Pullum (1990). The X-bar theory of phrase structure. *Language 66*, 24–50.

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*, pp. 66–98. University of Chicago Press.

Lopez, P. (2000). Extended partial parsing for lexicalized tree grammars. In *Proc. of IWPT 2000*, pp. 159–170.

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

Miller, P. (1999). *Strong Generative Capacity: The Semantics of Linguistic Formalism*. CSLI Publications.

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 *Proc. of TAG+5*, pp. 193–198.

Schabes, Y., A. Abeillé, and A. K. Joshi (1988). Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. In *Proc. of 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 *Proc. of 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 6*(1), 63–80.

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

Vijay-Shanker, K. and A. K. Joshi (1988). Feature structures based Tree Adjoining Grammars. In *Proc. of COLING '92*, pp. 714–719.

Yoshinaga, N., Y. Miyao, K. Torisawa, and J. Tsujii (2001). Efficient LTAG parsing using HPSG parsers. In *Proc. of PACLING 2001*, pp. 342–351.