Parsing comparison across grammar formalisms using strongly equivalent grammars

Comparison of LTAG and HPSG parsers: A case study

Naoki Yoshinaga* — **Yusuke Miyao*** — **Kentaro Torisawa****
Jun'ichi Tsujii*,***

* *University of Tokyo*
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-0033, Japan
{yoshinag, yusuke, tsujii}@is.s.u-tokyo.ac.jp

** *Japan Advanced Institute of Science and Technology*
1-1 Asahidai, Tatsunokuchi, Ishikawa, 923-1292, Japan
torisawa@jaist.ac.jp

*** *CREST, JST (Japan Science and Technology Agency)*
4-1-8 Honcho, Kawaguchi-shi, Saitama, 332-0012, Japan

ABSTRACT. *This article presents a novel approach to empirical comparison between parsers for different grammar formalisms such as LTAG and HPSG. The key idea of our approach is to use strongly equivalent grammars obtained by grammar conversion, which generate equivalent parse results for the same input. We validate our approach by giving a formal proof of strong equivalence for an existing grammar conversion from LTAG to HPSG-style grammar. Experimental results using two pairs of LTAG and HPSG parsers with dynamic programming and CFG filtering reveal that the different ways of using the parsing techniques cause significant performance differences, and also suggest a definite way of improving these parsing techniques.*

RÉSUMÉ. *Nous présentons une approche nouvelle pour comparer de manière empirique des analyseurs syntaxiques pour des formalismes grammaticaux comme LTAG et HPSG. L'idée centrale consiste à utiliser des grammaires fortement équivalentes obtenues par conversion et produisant des analyses équivalentes pour la même chaîne d'entrée. Nous validons cette approche en fournissant une preuve formelle d'équivalence forte pour le mécanisme de conversion de grammaires de LTAG vers HPSG. Les résultats expérimentaux obtenus avec deux paires d'analyseurs syntaxiques LTAG et HPSG en utilisant la programmation dynamique et le filtrage CFG ont montré que les différences de performance résultent des différence d'adaptation des techniques d'analyse, suggérant une piste solide pour améliorer celles-ci.*

KEYWORDS: *parsing, dynamic programming, CFG filtering, LTAG, HPSG.*

MOTS-CLÉS : *analyse syntaxique, programmation dynamique, filtrage CFG, LTAG, HPSG.*

1. Introduction

Over the past two decades, a wide range of parsers have been developed for lexicalized grammars such as Lexicalized Tree Adjoining Grammar (LTAG) [SCH 88] and Head-Driven Phrase Structure Grammar (HPSG) [POL 94a]. Parsers that have been proposed independently of one another often share the same parsing techniques that are claimed to be independent of individual grammar formalisms. Examples of such generic techniques are dynamic programming [YOU 67, EAR 70, VIJ 85, HAA 87], left-to-right parsing [TOM 86, BRI 93, NED 98], and two-phase parsing [MAX 93, TOR 95, YOS 99] including CFG filtering [TOR 96, TOR 00, KIE 00]. However, as mentioned in [CAR 94], while these techniques are generic in the sense that they can be used for efficient implementation of parsers for any grammar formalism, their impact often varies from one formalism to another [SCH 95, YOS 99]. It seems that generic techniques actually interact with the characteristics of individual grammar formalisms.

To assess the true effect of algorithmic techniques such as those derived from dynamic programming, CFG filtering, etc., we need a means of abstracting away the ‘surface’ differences between grammar formalisms. Such abstraction should also be useful for adapting techniques found that have been efficient with parsers based on one formalism to parsers based on another.

In this article, we propose grammar conversion as a means of abstracting away the surface differences between grammar formalisms. That is, we show that, by constructing a “*strongly equivalent*” grammar[1] in a particular formalism from one given in another formalism and by measuring the performance of parsers based on the original grammar and ones based on the newly-constructed grammar, one can gain a deeper insight into generic parsing techniques and share techniques developed for parsers for different grammar formalisms. Strongly equivalent grammars obtained by grammar conversion allow us to carry out a meaningful comparison among parsers for different grammar formalisms regardless of their surface differences, because the parsers handle equivalent grammatical constraints (which are preserved by the grammar conversion) and thus produce equivalent parse results for the same input. Strongly equivalent grammars are also very helpful for incorporating techniques that have been found to be efficient from parsers based on one formalism to parsers based on another, because the grammar conversion defines a clear correspondence between those grammars, *i.e.*, a correspondence which enables us to observe how parsers for one formalism handle a grammar in another formalism.

We focus on two generic parsing techniques in this article, namely dynamic programming [SAR 00a, HAA 87] and CFG filtering [HAR 90, POL 94b, TOR 96,

1. Chomsky [CHO 63] first introduced the notion of strong equivalence between grammars, where both grammars generate the same set of structural descriptions (*e.g.*, parse trees). Kornai and Pullum [KOR 90] and Miller [MIL 99] used the notion of isomorphism between sets of structural descriptions to provide the notion of strong equivalence across grammar formalisms, which we have adopted in this research.

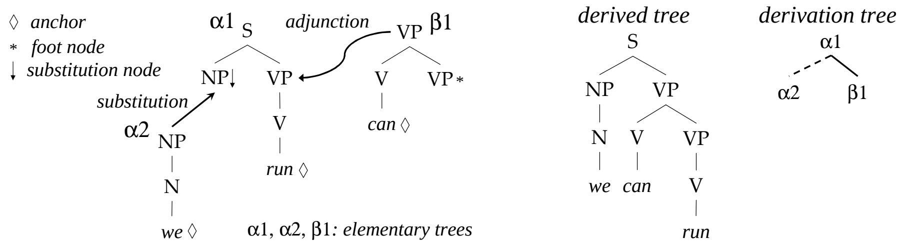


Figure 1. *Lexicalized Tree Adjoining Grammar: basic structures (elementary trees) and compose operations (substitution and adjunction)*

POL 98, TOR 00, KIE 00]. We first see how these techniques have been employed in parsers for two particular grammar formalisms, LTAG and HPSG. Since dynamic programming forms the basis of most current parsing techniques, a comparison of parsers using it allows us to roughly grasp the difference between the performance of LTAG and HPSG parsers. Since the impact of CFG filtering for LTAG is quite different from that for HPSG, a comparison of parsers using it can demonstrate the utility of our methods. Next, we show that grammar conversion yielding an HPSG-style grammar from a given LTAG grammar reveals the true nature of these generic parsing techniques, and results in parsers for LTAG that are more efficient than those implemented for the original LTAG grammar, even though they use the same generic techniques.

To validate our approach, we give a formal proof of strong equivalence for an existing grammar conversion from LTAG to HPSG-style grammar [YOS 02], and use it to obtain strongly equivalent grammars. Empirical comparisons of parser performance were then conducted using two LTAG grammars and their equivalent HPSG-style grammars. One is the XTAG English grammar [XTA 01], which is a large-scale handcrafted Feature-Based LTAG (FB-LTAG) [VIJ 87, VIJ 88], and the other is an LTAG grammar that was automatically extracted from the Penn Treebank [MAR 93].

2. Grammar formalisms and parsing techniques

2.1. *Grammar formalisms*

2.1.1. *Lexicalized Tree Adjoining Grammar*

An LTAG [SCH 88] is defined by a set of *elementary trees* that are composed by two operations called *substitution* and *adjunction*. These are 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 as either *initial trees* ($\alpha 1$ and $\alpha 2$) or *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 the foot node are called *substitution nodes* (marked with $\downarrow$). The left-hand side of Figure 1 also 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). The derivation trees represent the history of combinations of trees.

FB-LTAG [VIJ 87, VIJ 88] is an extension of the LTAG formalism in which each node in the elementary trees has a feature structure, which contains a set of grammatical constraints on the node. The constraints are to be satisfied through unification during adjunction and substitution.

2.1.2. *Head-Driven Phrase Structure Grammar*

We define *an HPSG-style grammar* according to the computational architecture of HPSG [POL 94a]. It consists of *lexical entries* and *Immediate Dominance (ID) grammar rules*, each of which is described with typed feature structures [CAR 92]. The greater generative power of the underlying architecture of HPSG allows us to obtain a trivial encoding of LTAG in the typed feature structure, as described by [KEL 94, pp. 144–151]. However, such a conversion cannot meet our needs because the resulting grammar is far from the one defined in [POL 94a]. Hence, we restrict the form of an HPSG-style grammar to one that follows the HPSG formalism in the following ways. 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 defined by lexical characteristics. These restrictions enable us not only to define a formal link between computational architectures that underlies LTAG and HPSG, but also to clarify the relationships between linguistic accounts given using LTAG and HPSG by comparing the HPSG-style grammar converted from LTAG with HPSG. The interested reader may refer to the discussion in [YOS 02].

Note that Pollard and Sag [POL 94a] provide detailed linguistic specifications for the form of feature structures and adopt *principles*, such as *the Immediate Dominance Principle*, to express linguistic concepts, for example projection. In our definition, we assume that principles are implicitly encoded in ID grammar rules and when we convert an LTAG grammar to an HPSG-style grammar we do not attempt to translate linguistic specifications in the LTAG into the corresponding HPSG principles.

2.2. ***Conventional parsers***

The LTAG and HPSG parsers with dynamic programming used in our experiments [NOO 94, HAA 87] perform *factoring*, a common-sense parsing technique that avoids generating duplicate equivalent partial parse trees. In the following sections, we briefly describe how factoring is accomplished in parsers for the two formalisms.

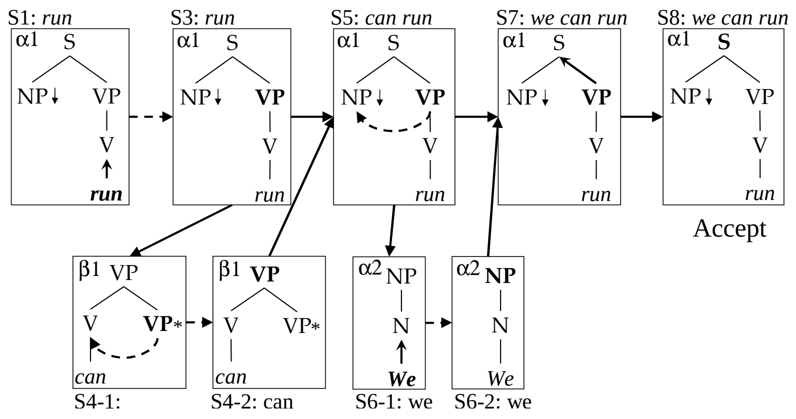


Figure 2. *Example of head-corner parsing for an LTAG grammar*

2.2.1. *Head-corner parser for LTAG*

One of the LTAG parsers used in our experiments, which we call the "conventional" LTAG parser, is a head-corner LTAG parser [SAR 00a]. Its parsing algorithm is a chart-based variant of van Noord's [NOO 94]. The parser uses a data structure called *an agenda*. The agenda stores *states* to be processed. A state is represented by a quadruplet consisting of an elementary tree, a root node, a processing node, and a span over the input, which denote the processed and unprocessed parts of the tree.

Figure 2 depicts the process of head-corner parsing for the sentence "*we can run*." In the figure, nodes in bold face, arrows in states, arrows between states, and strings followed by the state number S# respectively indicate processing nodes, directions of processing, relations between the states, and spans over the input string. In head-corner parsing, the parser traverses a tree from one leaf node called *the head-corner* to the root node. This tree traversal is called *head-corner traversal*. During head-corner traversal, the parser recognizes the siblings of the processing node and possible adjunctions at this node. In Figure 2, the parser first predicts an initial tree $\alpha2$ whose root node matches the symbol S corresponding to the sentence (state S1 in Figure 2). The parser proceeds in a bottom-up manner from the head-corner "*run*" to S. After moving up to the node VP in $\alpha2$ (S3), the parser recognizes the adjunction at the processing node VP and introduces a new state S4-1 for the adjoining tree $\beta1$. After recognizing $\beta1$ (S4-2), the parser tries to recognize the sibling of VP (S5). To recognize the sibling NP, the parser introduces a new state S6-1 for $\alpha1$. Then, the parser proceeds to the root S of $\alpha2$ (S8). Since there is no state to be processed in the agenda, parsing of "*we can run*" ends.

The parser performs factoring when it generates a new state. That is, it pushes a state in the agenda only when an equivalent state does not exist in the agenda. Note that equivalent states are those which have the same elements in the quadruplet.

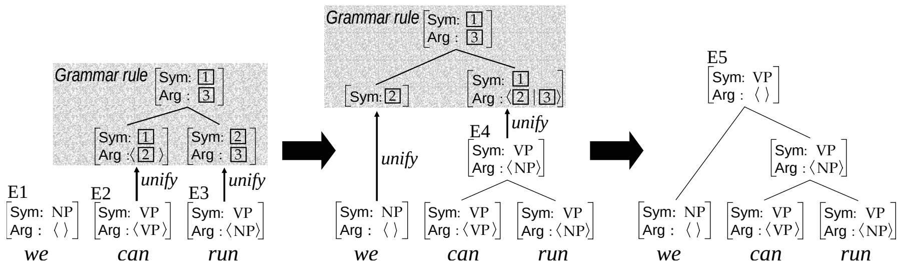


Figure 3. *Example of CKY-style parsing for an HPSG grammar*

2.2.2. *CKY-style HPSG parser*

One of the HPSG parsers used in our experiments, which we call the "conventional" HPSG parser, is a CKY-style HPSG parser [HAA 87]. The parser uses a data structure called *a triangular table*. The triangular table stores *edges*, which correspond to partial parse trees. An edge is described with a tuple consisting of a feature structure that represents the root node of a partial parse tree and a span over the input.

Figure 3 illustrates an example of the CKY-style parsing for an HPSG grammar. First, lexical entries for "*we*," "*can*," and "*run*" are stored as edges E1, E2, and E3 in the triangular table. Next, E2 and E3 are each unified with the daughter feature structures of an ID grammar rule. The feature structure of the mother node is determined as a result of this unification and is stored in the triangular table as a new edge E4. An ID grammar rule is then applied to E1 and E4, and a new edge E5 is generated. Since the parse tree spans the whole input string, parsing of "*we can run*" ends.

The parser performs factoring when it generates a new edge. That is, it stores an edge in the triangular table unless an equivalent edge exists in the cell. Note that equivalent edges are those which have the same elements in the tuple.

2.3. *CFG filtering techniques*

CFG filtering is a parsing scheme that predicts possible parse trees by using a CFG extracted from a given grammar. An initial offline step of CFG filtering is performed to *approximate* a given grammar with a CFG, in other words, to extract a CFG backbone from a given grammar (*Context-Free (CF) approximation*). The resulting CFG is used as an efficient device for computing the necessary conditions for parse trees.

After the initial step, CFG filtering generally comprises two phases. In phase 1, the parser first constructs possible parse trees by using the CFG obtained in the initial offline step, and then filters out CFG edges unreachable by top-down traversal starting from roots of successful context-free derivations. In phase 2, it eliminates invalid parse trees by using full constraints in the given grammar. We call the remaining CFG edges that are used for phase 2 *essential edges*.

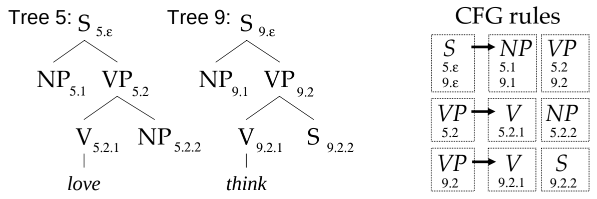


Figure 4. *Extraction of CFG from LTAG*

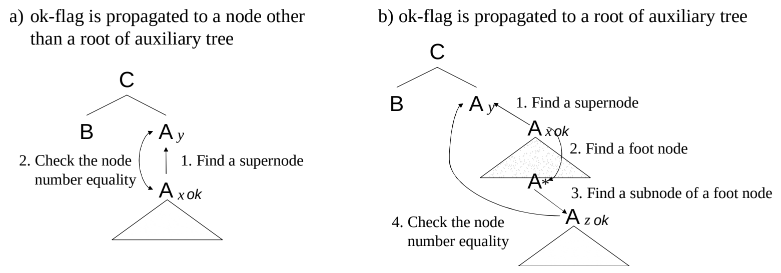

a) ok-flag is propagated to a node other than a root of auxiliary tree
C
B
A y
1. Find a supernode
2. Check the node number equality
A x ok
b) ok-flag is propagated to a root of auxiliary tree
C
B
A y
1. Find a supernode
A x ok
2. Find a foot node
A*
3. Find a subnode of a foot node
4. Check the node number equality
A z ok

Figure 5. *Ok-propagation from an essential edge to another*

The parsers with CFG filtering for LTAG and HPSG follow the above parsing strategy, but differ in their ways of approximating a grammar with CFG and eliminating impossible parse trees in phase 2. In the following sections, we briefly describe the CF approximation and the elimination of impossible parse trees for each formalism.

2.3.1. *CF approximation of LTAG*

In the CFG filtering techniques for LTAG [HAR 90, POL 98], every branching of elementary trees in a given grammar is extracted as a CFG rule as shown in Figure 4.

Because the obtained CFG can reflect only local constraints given in each local structure of the elementary trees, it generates invalid parse trees that connect local trees extracted from different elementary trees. To eliminate such illegal parse trees, a link between branchings is preserved as *a node number* which records a unique node address (a subscript attached to each node in Figure 4). As depicted in Figure 5, we can eliminate such parse trees by traversing essential edges in a bottom-up manner and recursively propagating *an ok-flag* from node number x to node number y when a connection between x and y is allowed in the LTAG grammar. We call this propagation *ok-propagation* [POL 94b].

2.3.2. *CF approximation of HPSG*

In the CFG filtering techniques for HPSG [TOR 96, TOR 00, KIE 00], a CFG is extracted from a given HPSG grammar by recursively instantiating daughters of a grammar rule with lexical entries and generated feature structures, as shown in Fig-

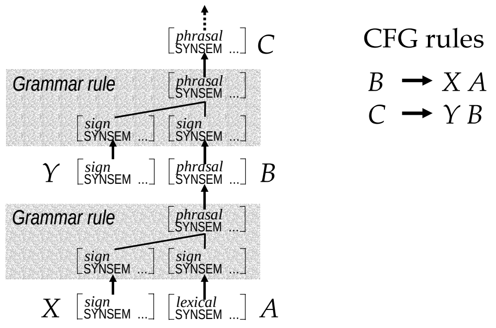


Figure 6. *Extraction of CFG from HPSG*

ure 6. This procedure stops when new feature structures are not generated. We must impose restrictions on the features (*i.e.*, ignore them) or on the number of rule instantiations or both in order to guarantee termination of the rule instantiation. A CFG is obtained by regarding the initial and the generated feature structures as nonterminals and bottom-up derivation relationships as CFG rules.

Although the resulting CFG reflects the local and global constraints of the whole structure of lexical entries, it generates invalid parse trees that do not reflect the constraints given by the features that were ignored in the CFG. These parse trees are eliminated in phase 2 by applying HPSG grammar rules that correspond to the applied CFG rules. We call this rule application *rule-application*.

3. Grammar conversion

This section describes an algorithm for converting from LTAG to a strongly equivalent HPSG-style grammar [YOS 02]. The conversion algorithm consists of two phases of conversion: i) a conversion from LTAG into *canonical LTAG*, LTAG which consists only of *canonical elementary trees*, and ii) a conversion from the canonical LTAG into an HPSG-style grammar. Substitution and adjunction are emulated by pre-determined ID grammar rules.[2] This algorithm can easily handle an FB-LTAG grammar by merely extending the grammar rules to execute the feature structure unification in the same way as in FB-LTAG. We give a formal proof of strong equivalence for the above two phases of conversion in Appendices A.2 and A.3, respectively.

We define *canonical elementary trees*, which have one-to-one correspondences with HPSG lexical entries. Canonical elementary trees are elementary trees which

2. In this article, we assume that elementary trees consist of binary branching structures. A unary branching can be regarded as a binary branching in which one daughter is the empty category, and n-ary ($n \geq 3$) branchings can similarly be converted into binary branchings. This conversion guarantees strong equivalence by virtue of being a one-to-one mapping.

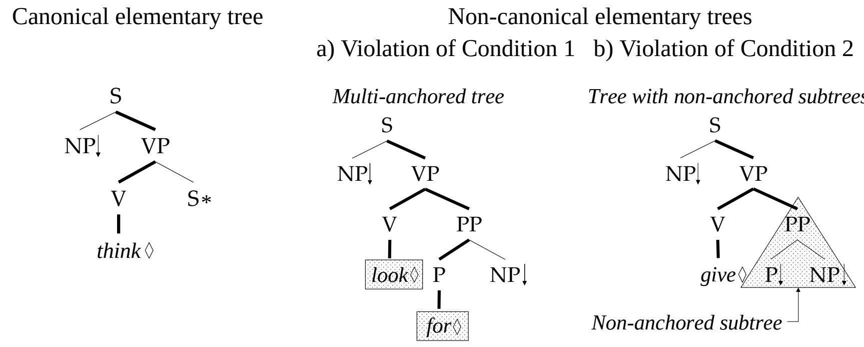

Canonical elementary tree
Non-canonical elementary trees
a) Violation of Condition 1
b) Violation of Condition 2
Multi-anchored tree
Tree with non-anchored subtrees
S
NP↓
VP
V
S∗
think◊
PP
look◊
P
NP↓
for◊
give◊
P↓
Non-anchored subtree

Figure 7. *A canonical elementary tree and non-canonical elementary trees*

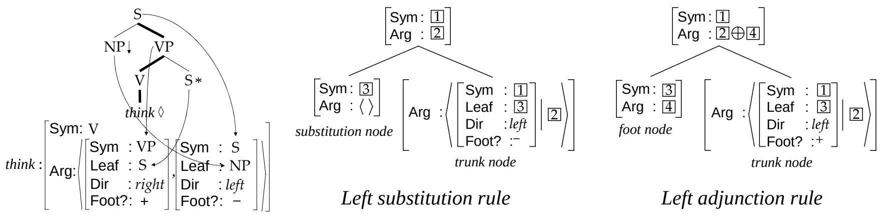


Figure 8. *Conversion of LTAG elementary trees into HPSG lexical entries (left) and the grammar rules: the substitution rule (center) and the adjunction rule (right)*

satisfy two conditions. Condition 1: A tree has only one anchor, and Condition 2: Every branching structure in a tree contains *trunk nodes*. Trunk nodes (nodes in bold face in Figure 7) are nodes on *a trunk* (thick branches in Figure 7), which is a path from an anchor to a root node other than the anchor [KAS 95]. Condition 1 guarantees that a canonical elementary tree has only one trunk, while Condition 2 guarantees that each branching consists of a trunk node, a leaf node, and their mother, as seen in the example on the left-hand side of Figure 7. The right-hand side of Figure 7 shows non-canonical trees. We call a subtree of depth $n\,(n \geq 1)$ that includes no anchor *a non-anchored subtree*. Non-canonical elementary trees are first converted into canonical trees, and then converted into HPSG lexical entries.

3.1. *Conversion of canonical elementary trees*

The left-hand side of Figure 8 depicts an example of conversion of *canonical elementary trees*. A canonical elementary tree is converted into an HPSG lexical entry by regarding leaf nodes as arguments of the anchor and storing them in a stack (the Arg feature in the left-hand side of Figure 8) where each leaf node is expressed by a triplet consisting of the symbol, the direction against the trunk, and the type (the Leaf, Dir,

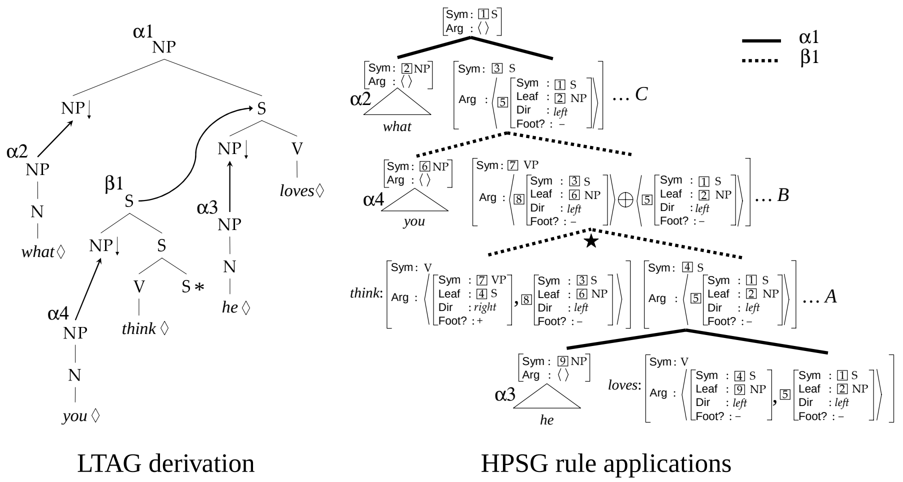


Figure 9. *LTAG and HPSG parsing of the phrase "what you think he loves"*

and Foot? features in the left-hand side of Figure 8, respectively). The trunk symbol is stored as the Sym feature in order to explicitly determine the symbol of the mother node in the rule applications.

Substitution and adjunction are emulated by grammar rules such as those shown in the center and the right-hand side of Figure 8, respectively. A co-indexing box $\boxed{n}$ indicates that two sub-feature structures share their values with each other. Both rules pop an element in the value of the Arg feature of a node and let the node subcategorize for a node unifiable with the popped element. A substitution rule requires that the Arg feature of the node to be subcategorized should be an empty stack, while an adjunction rule concatenates the values of the Arg features of both daughters.

Figure 9 shows examples of rule application. The solid lines indicate the adjoined tree ($\alpha1$) and the dotted lines indicate the adjoining tree ($\beta1$). The adjunction rule is applied in order to construct the branching marked with $\star$, where "*think*" takes as its argument the node having the Sym feature's value of S. By applying the adjunction rule, the Arg feature of the mother node B becomes a concatenated stack of the Arg features of both $\beta1$, '$\boxed{8}$', and $\alpha1$, '$\boxed{5}$.' Note that when the construction of $\beta1$ has been completed, the Arg feature of the trunk node C will return to its former state (A). We can continue constructing $\alpha1$ in the same way as for the case where no adjunction rules have been applied.

3.2. *Conversion of non-canonical elementary trees*

Non-canonical elementary trees are initially divided into multiple subtrees, each of which has at most one anchor, by a procedure called *tree division*, as shown in Fig-

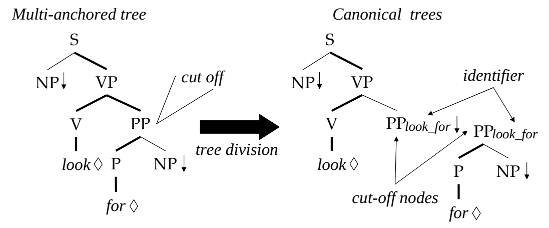


Figure 10. *Conversion of multi-anchored trees into multiple canonical trees*

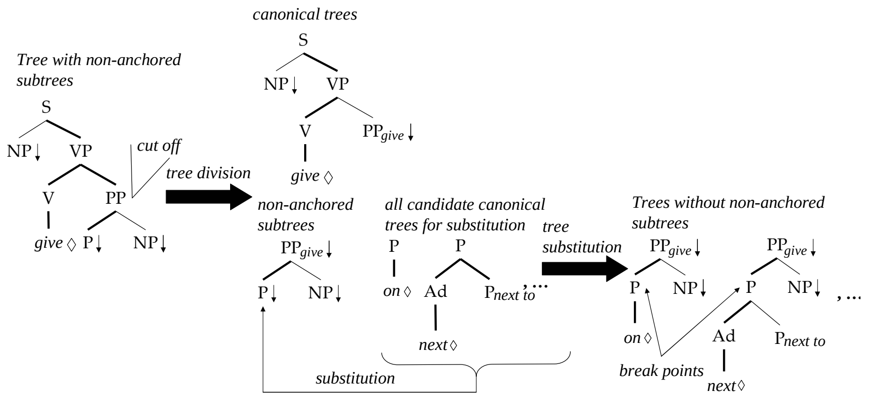


Figure 11. *Conversion of a tree with non-anchored subtrees into canonical trees and trees without non-anchored subtrees*

ures 10 and 11. Nodes that mark the separation of one tree into two are called *cut-off nodes*. A cut-off node is marked by *an identifier* to preserve the co-occurrence relation among the multiple anchors. The tree division converts multi-anchored trees, which only violate Condition 1, into canonical trees (Figure 10), while it converts trees with non-anchored subtrees into canonical trees and non-anchored subtrees (Figure 11).

A procedure called *tree substitution* converts the non-anchored subtrees into anchored trees. It causes a substitution at one substitution node on every deepest branching by every candidate tree for substitution, as shown in Figure 11. We should mention that the substituted nodes are marked as *break points* to remember that the resulting trees are obtained by substituting other trees at those nodes. The candidate trees for substitution are selected from among all the canonical elementary trees and the ones obtained by the conversion given in the previous paragraph. Because the candidate trees for substitution include neither non-anchored subtrees nor auxiliary trees whose root nodes do not originate from the root nodes of initial trees, the trees obtained by this process will satisfy Condition 2. When they take substitution at one substitution node, they also satisfy Condition 1 and are canonical trees; otherwise, they are multi-anchored trees and will be converted into canonical trees by the tree division.

Table 1. *Classification of elementary trees in the XTAG English grammar (LTAG) and lexical entries converted from them (HPSG)*

Grammar	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	**Total**
LTAG	326	763	54	50	1,193
HPSG	326	1,989	1,083	2,474	5,872

Table 2. *Parsing performance with the XTAG English grammar for the ATIS corpus*

Parser	**Parse time (sec.)**
Naive	1.54
lem	20.76

4. Comparison of LTAG and HPSG parsers

4.1. *Comparison of dynamic programming techniques*

We compared a pair of dynamic programming techniques for LTAG [NOO 94] and HPSG [HAA 87] described in Sections 2.2.1 and 2.2.2. Henceforth, *lem* refers to the LTAG parser [SAR 00a], ANSI C implementation of the head-corner parsing.[3] *Naive* refers to C++ implementation of the CKY-style HPSG parser.

We converted the latest version of the XTAG English grammar [XTA 01], which is a large-scale FB-LTAG grammar, into a strongly equivalent grammar by using the grammar conversion described in Section 3.1. Table 1 shows a classification of the elementary trees[4] of the XTAG English grammar according to the conditions described in Section 3.1.[5] In the table, $\mathcal{A}$ shows the number of canonical elementary trees, while $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ respectively show the number of trees that violate only Condition 1, only Condition 2, and both conditions. The second row indicates the number of HPSG lexical entries converted from the LTAG elementary trees.

Table 2 shows the parsing speed results for 452 sentences from the ATIS corpus [MAR 93][6] (average sentence length: 6.32 words). The machine used in the following experiments was a 1.26 GHz Pentium III with 4 GB memory. The results show that the HPSG parser achieved a speed-up of a factor of 13. Figure 12 shows parse time plotted against sentence length, where both axes use logarithmic scales. Since the increase in parse time versus sentence length plotted on logarithmic scales is equal

3. The LTAG parser is available at: ftp://ftp.cis.upenn.edu/pub/xtag/lem/lem-0.14.0.tgz
4. These elementary trees should more strictly be called *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.
5. We eliminated 33 elementary trees because the LTAG parser could not produce correct derivation trees with them; adjunction of these trees was sometimes not performed by the parser.
6. We eliminated 56 sentences because of parser time-outs, and 69 sentences because the LTAG parser had bugs in its preprocessor preventing it from producing correct derivation trees.

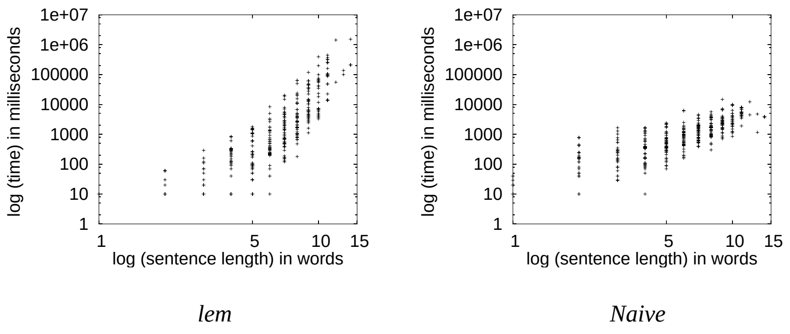


Figure 12. *Parsing performance with the XTAG English grammar for the ATIS corpus*

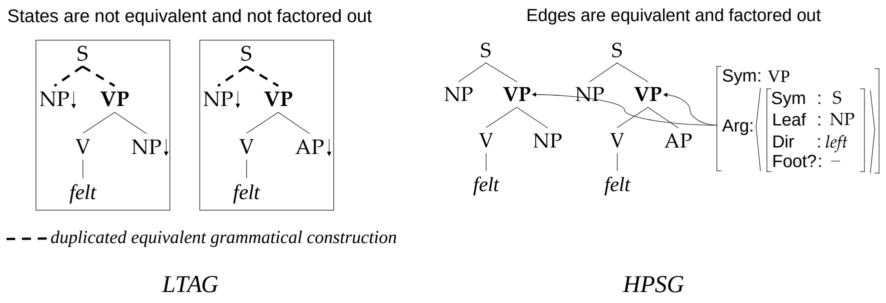
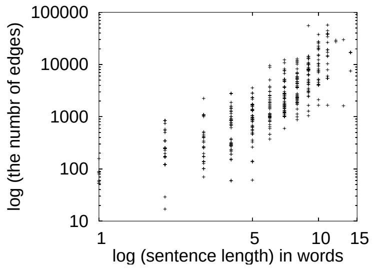

States are not equivalent and not factored out
Edges are equivalent and factored out
S
NP↓
VP
V
NP↓
AP↓
felt
NP
AP
Sym: VP
Arg:
Sym : S
Leaf : NP
Dir : left
Foot?: −
duplicated equivalent grammatical construction
LTAG
HPSG

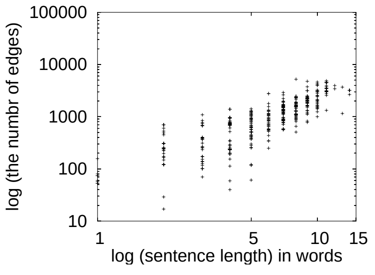
Figure 13. *Difference between factoring schemes in LTAG and HPSG*

to the degree of polynomial order of the empirical time complexity, the graphs show that the order of the empirical time complexity of *lem* is higher than that of *Naive*.

As noted in Section 2.2, both parsers have an architecture that supports factoring, but the ways in which they perform factoring differ. Remember that a state in the LTAG parser is a quadruplet consisting of an elementary tree, a root node, a processing node, and a span over an input, while an edge in the HPSG parser is a tuple consisting of a feature structure and a span over an input. By considering how the HPSG parser handles the HPSG-style grammar converted from the LTAG, we see that the HPSG parser treats a branching as its minimal component and performs factoring of edges when the edges' feature structures are equivalent (the right-hand side of Figure 13). On the other hand, the LTAG parser treats an elementary tree as its minimal component and performs factoring of states when the elementary trees of the states have equivalent root nodes (the left-hand side of Figure 13). Since the root node corresponds to a feature structure whose Arg feature is an empty stack in HPSG, this difference means that the HPSG parser factors out more partial parse trees than does the LTAG parser. As illustrated in Figure 13, the LTAG parser cannot avoid duplicating equivalent grammatical constructions corresponding to fragments of elementary



Naive$_{rf}$



Naive

Figure 14. *Number of edges of a variant of Naive (Naive$_{rf}$) which performs factoring only when the factoring can be performed in lem (left) and Naive (right)*

trees. This difference in factoring schemes leads to the difference in the empirical time complexity.

To verify the above argument, we conducted a parsing experiment on the same corpus by using a variant of *Naive* (hereafter *Naive$_{rf}$*) which performs factoring only when the factoring could be performed by the LTAG parser.[7] Since edges and states generated by the LTAG and HPSG parsers represent partial parse trees, their numbers are reliable indicators of empirical time complexity. Figure 14 shows the number of edges plotted against sentence length, where both axes use logarithmic scales.[8] The increase in the number of edges of *Naive$_{rf}$* was higher than that of *Naive*. Since the parsing scheme of *Naive$_{rf}$* mimics that of *lem*, the difference in parse time between *lem* and *Naive* and the difference in the number of edges between *Naive$_{rf}$* and *Naive* confirm that the difference in the factoring scheme is the major cause of difference in the empirical time complexity.[9]

At first glance, these results are inconsistent with the fact that the theoretical bound of worst time complexity for HPSG parsing is exponential, while LTAG parsing requires $\mathcal{O}(n^6)$ for an input of length n. However, Carroll [CAR 94] demonstrated that theoretical bounds of time complexity with respect to grammar size and input length have little impact on performance for some unification-based parsing algorithms, and attributed the reason to the specification of grammars (*i.e.*, variations in grammar rules, etc.). Sarkar et al. [SAR 00b] studied LTAG grammars extracted from the Penn Treebank and reported that the theoretical bound of computational complexity does not significantly affect parsing performance and that the most dominant factor is syntactic lexical ambiguity, *i.e.*, ambiguity of lexical entries for the same words. Our results are

7. We did not compare the states of *lem* with the edges of *Naive* because an edge of *Naive* does not have a one-to-one correspondence with a state of *lem*.
8. *Naive* and *Naive$_{rf}$* in fact duplicate a part of auxiliary trees when they adjoin to different trees. Although *lem* can avoid this duplication, it has no serious effect on our conclusion.
9. *lem* and *Naive* also differ in their ways of handling linguistic features. However, we suppose that their impact on parsing performance is mild compared to one by the difference in factoring.

therefore convincing because factoring handles ambiguity in partial parse trees, which is mostly caused by the syntactic lexical ambiguity.

We should note that there is other work that aims at avoiding the ambiguity caused by syntactic lexical ambiguity. Evans and Weir [EVA 98] have asserted that the compaction of substructures in elementary trees has a great impact on parsing performance. In their research, several elementary trees for each word were converted into finite-state automata, and merged into a single finite-state automaton. Chen and Vijay-Shanker [CHE 97] use underspecified tree descriptions to allow ambiguous node labels in elementary trees. In the conventional HPSG parser, some of this compaction is dynamically executed by factoring. Furthermore, the factoring method enables another kind of compaction that merges equivalent edges headed by different words. Existing methods cannot perform this kind of compaction, since these techniques are applied separately to the elementary trees for each word. Shaumyan et al. [SHA 02] evaluated an automaton-based parser with an LTAG grammar extracted by a method proposed by Xia [XIA 99], and showed results similar to ours. However, the grammar they used had far less syntactic lexical ambiguity than the XTAG English grammar. Our results with the XTAG English grammar are a strong indication of the importance of compaction of substructures in elementary trees.

It should be noted that the above investigation also suggests another way of factoring in LTAG. We can merge two states which have equivalent unprocessed parts, as depicted in Figure 13, into a single state when they cover the same span of input. This kind of factoring that merges edges with equivalent unprocessed parts has been proposed for CFG by Leermakers [LEE 92]. In his parser, the edges in Earley parsing are merged if their rules have a common unprocessed suffix. As exemplified by the application of *Naive* to an HPSG-style grammar converted from LTAG, this kind of factoring is applicable to LTAG parsing. Our study empirically attested to its effectiveness in LTAG parsing, not by implementing complex parsing algorithms, but by simply applying the existing HPSG parser potentially equipped with such a functionality to the grammar converted from LTAG.

4.2. *Comparison of CFG filtering techniques*

Following on from the comparison of dynamic programming techniques, we compared a pair of CFG filtering techniques for LTAG [HAR 90, POL 98] and HPSG [TOR 00, KIE 00] described in Sections 2.3.1 and 2.3.2. We chose the filtering technique of Poller and Becker [POL 98] because it is the most sophisticated algorithm for CFG filtering for LTAG. Hereafter, we refer to its C++ implementation as *PB*. The other filtering technique for HPSG was *TNT* [TOR 00]. We modified the CF approximation of the original *TNT* by instantiating both daughters and restricting the number of rule instantiations, as shown in [KIE 00], to approximate the obtained HPSG-style grammar with CFG. In phase 1, *PB* and *TNT* performed Earley [EAR 70] and CKY [YOU 67] parsing, respectively. Note that the CFG filtering techniques for LTAG used the same CF approximation, as did the CFG filtering techniques for HPSG,

Table 3. *Size of extracted LTAGs (elementary trees) and CFGs approximated from them (above: the number of nonterminals; below: the number of rules)*

Grammar	**G**$_2$	**G**$_{2\text{-}4}$	**G**$_{2\text{-}6}$	**G**$_{2\text{-}8}$	**G**$_{2\text{-}10}$	**G**$_{2\text{-}21}$
LTAG	1,488	2,412	3,139	3,536	3,999	6,085
CFG$_{PB}$	65	66	66	66	67	67
	716	954	1,090	1,158	1,229	1,552
CFG$_{TNT}$	1,989	3,118	4,009	4,468	5,034	7,454
	18,323	35,541	50,115	58,356	68,239	118,464

Table 4. *Parsing performance (sec.) for Section 2 of WSJ*

Parser	**G**$_2$	**G**$_{2\text{-}4}$	**G**$_{2\text{-}6}$	**G**$_{2\text{-}8}$	**G**$_{2\text{-}10}$	**G**$_{2\text{-}21}$
PB	1.4	9.1	17.4	24.0	34.2	124.3
TNT	0.044	0.097	0.144	0.182	0.224	0.542

Table 5. *Number of essential edges generated in parsing of Section 02 of WSJ*

Parser	**G**$_2$	**G**$_{2\text{-}4}$	**G**$_{2\text{-}6}$	**G**$_{2\text{-}8}$	**G**$_{2\text{-}10}$	**G**$_{2\text{-}21}$
PB	791	1,435	1,924	2,192	2,566	3,976
TNT	63	121	174	218	265	536

as described in Sections 2.3.1 and 2.3.2. Comparison of *PB* and *TNT* thus suffices to investigate the effect of the CF approximations for LTAG and HPSG.

We acquired LTAGs by the method proposed in [MIY 03] from Sections 2-21 of the Wall Street Journal (WSJ) in the Penn Treebank [MAR 93] and their subsets.[10] We converted them into strongly equivalent HPSG-style grammars by using the grammar conversion described in Section 3.1. Table 3 shows the size of CFGs approximated from the strongly equivalent grammars. G$_x$, CFG$_{PB}$, and CFG$_{TNT}$ henceforth refer to the LTAG extracted from Section x of WSJ and the CFGs approximated from G$_x$ by *PB* and *TNT*, respectively. CFG$_{TNT}$ is much larger than CFG$_{PB}$. By investigating parsing performance using these CFGs as filters, we conclude that larger size of CFG$_{TNT}$ caused the better parsing performance.

Table 4 shows the parse time for 254 sentences of length $n\,(n \leq 10)$ from Section 2 of WSJ (average sentence length: 6.72 words).[11] This result shows not only that *TNT* was much faster than *PB*, but also that the performance difference between them increased when the larger grammars were used.

To estimate the degree of CF approximation, we measured the number of essential (inactive) edges of phase 1. Table 5 shows the number of essential edges. *PB* produces

10. The elementary trees in the LTAGs are binarized.

11. We used a subset of the training corpus to avoid using default lexical entries for unknown words, because there are various ways to assign default entries for automatically extracted grammars and this would have an uncontrolled effect on parsing performance.

Table 6. *Success rate (%) of phase 2 operations*

Operations	$\mathbf{G}_2$	$\mathbf{G}_{2\text{-}4}$	$\mathbf{G}_{2\text{-}6}$	$\mathbf{G}_{2\text{-}8}$	$\mathbf{G}_{2\text{-}10}$	$\mathbf{G}_{2\text{-}21}$
ok-propagation (*PB*)	38.5	34.3	33.1	32.3	31.7	31.0
rule-application (*TNT*)	100	100	100	100	100	100

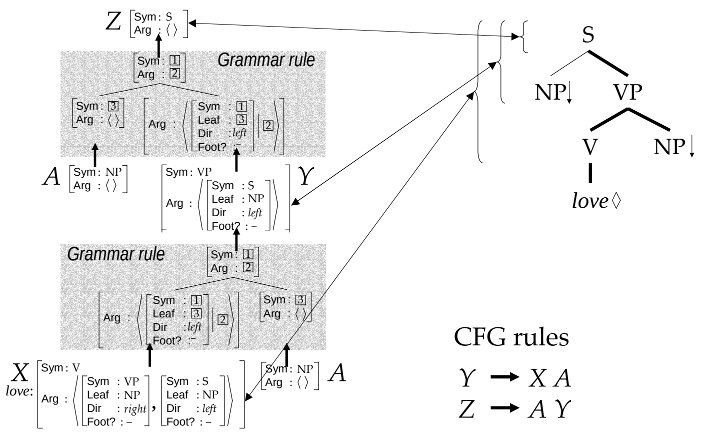


Figure 15. *CF approximation of an HPSG-style grammar converted from LTAG*

a much greater number of essential edges than *TNT*. We then investigated the effect of different numbers of essential edges on phase 2. Table 6 shows the success rates of ok-propagation and rule-application. The success rate of rule-application (for *TNT*) is 100%, while that of ok-propagation (for *PB*) is quite low.[12] These results indicate that CFG$_{TNT}$ is superior to CFG$_{PB}$ with respect to the degree of CF approximation.

We can work out the reason for this difference by investigating how the CF approximation of HPSG approximates HPSG-style grammars converted from LTAGs. As described in Section 3.1, the grammar conversion preserves the whole structure of each elementary tree (precisely, a canonical elementary tree) in a stack, and grammar rules manipulate the top element of the stack. A generated feature structure in the approximation process thus corresponds to the whole unprocessed parts of a canonical elementary tree, as shown in Figure 15. This implies that successful context-free derivations obtained by CFG$_{TNT}$ basically involve elementary trees in which all substitution and adjunction have succeeded. However, as mentioned in Section 2.3.1, CFG$_{PB}$ (as well as a CFG produced by another work [HAR 90]) cannot avoid generating invalid parse trees that connect two local structures where adjunction takes place

12. This means that the extracted LTAGs should be compatible with CFG and were completely approximated with CFGs. Preliminary experiments with the XTAG English grammar [XTA 01] without features were 15.3(parse time (sec.))/30.6(success rate (%)) for *PB* and 0.606/71.2 for *TNT*, for the same sentences.

between them. We used $G_{2\text{-}21}$ to calculate the percentage of ok-propagations that were between two node numbers that take adjunction (the right-hand side of Figure 5) and the success rate for this percentage. 87% of the total number of ok-propagations were of this type but their success rate was only 22%. These results suggest that the global constraints in a given grammar are essential to obtaining an effective CFG filter.

It should be noted that the above investigation also suggests another way of making a CF approximation for LTAG. We first define a unique mode of tree traversal, such as head-corner traversal [NOO 94] described in Section 2.2.1, on which we can sequentially apply substitution and adjunction. We then recursively apply substitution and adjunction on that traversal to elementary trees and generated tree structures. Because the processed parts of the generated tree structures are not used again, we regard the unprocessed parts of the tree structures as nonterminals of a CFG. We can thereby perform another type of CFG filtering for LTAG by combining this CFG filter with a head-corner LTAG parsing algorithm [NOO 94] that uses the same tree traversal.

5. Conclusion

This article presented an approach for comparing parsers for different grammar formalisms, making use of strongly equivalent grammars obtained by grammar conversion. We showed that a parsing comparison between LTAG and HPSG is possible by giving a formal proof of strong equivalence for the grammar conversion procedure [YOS 02] and using this conversion to obtain strongly equivalent grammars. As an application of our approach, we empirically compared two pairs of LTAG and HPSG parsers based on dynamic programming and CFG filtering. We first obtained strongly equivalent grammars by converting the XTAG English grammar and LTAG grammars extracted from the Penn Treebank into HPSG-style grammars. Experiments comparing parsers using dynamic programming showed that the different implementations of the factoring scheme caused a difference in the empirical time complexity of the parsers. This result suggests that for LTAG parsing we can achieve a drastic speed-up by merging two states whose elementary trees have the same unprocessed parts. Another experiment comparing parsers with CFG filtering showed that the CF approximation of HPSG produced a more effective filter than that of LTAG. This result also suggests that we can obtain an effective CFG filter for LTAG by approximating the LTAG with a CFG by applying substitution and adjunction along tree traversal and regarding unprocessed parts of generated tree structures as nonterminals of the CFG.

Acknowledgements

The authors wish to thank Anoop Sarkar for his help in applying his parser and Yousuke Sakao for his help in profiling the TNT parser. The authors are also indebted to John Carroll and the anonymous reviewers for their valuable comments. The first author was supported in part by JSPS Research Fellowships for Young Scientists.

6. References

[BRI 93] BRISCOE E., CARROLL J., “Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars”, *Computational Linguistics*, vol. 19, num. 1, 1993, p. 25-60.

[CAR 92] CARPENTER B., *The Logic of Typed Feature Structures*, Cambridge University Press, 1992.

[CAR 94] CARROLL J., “Relating complexity to practical performance in parsing with wide-coverage unification grammars”, *Proc. of the 32nd ACL*, 1994, p. 287–294.

[CHE 97] CHEN J., VIJAY-SHANKER K., “Towards a reduced-commitment D-theory style TAG parser”, *Proc. of the fifth IWPT*, 1997, p. 18–29.

[CHO 63] CHOMSKY N., “Formal properties of grammar”, LUCE R. D., BUSH R. R., GALANTER E., Eds., *Handbook of Mathematical Psychology*, vol. II, p. 323–418, John Wiley and Sons, Inc., 1963.

[EAR 70] EARLEY J., “An efficient context-free parsing algorithm”, *Communications of the ACM*, vol. 6, num. 8, 1970, p. 451–455.

[EVA 98] EVANS R., WEIR D., “A structure-sharing parser for lexicalized grammars”, *Proc. of COLING–ACL*, 1998, p. 372–378.

[HAA 87] HAAS A. R., “Parallel parsing for unification grammars”, *Proc. of the 14th IJCAI*, 1987, p. 615–618.

[HAR 90] HARBUSCH K., “An efficient parsing algorithm for Tree Adjoining Grammars”, *Proc. of the 28th ACL*, 1990, p. 284–291.

[KAS 95] KASPER R., KIEFER B., NETTER K., VIJAY-SHANKER K., “Compilation of HPSG to TAG”, *Proc. of the 33rd ACL*, 1995, p. 92–99.

[KEL 94] KELLER B., *Feature Logics, Infinitary Descriptions and Grammars*, CSLI publications, 1994.

[KIE 00] KIEFER B., KRIEGER H.-U., “A context-free approximation of Head-Driven Phrase Structure Grammar”, *Proc. of the sixth IWPT*, 2000, p. 135–146.

[KOR 90] KORNAI A., PULLUM G. K., “The X-bar theory of phrase structure”, *Language*, vol. 66, 1990, p. 24–50.

[LEE 92] LEERMAKERS R., “A recursive ascent Earley parser”, *Information Processing Letters*, vol. 41, num. 2, 1992, p. 87–91.

[MAR 93] MARCUS M., SANTORINI B., MARCINKIEWICZ M. A., “Building a large annotated corpus of English: the Penn Treebank”, *Computational Linguistics*, vol. 19, num. 2, 1993, p. 313–330.

[MAX 93] MAXWELL III J. T., KAPLAN R. M., “The interface between phrasal and functional constraints”, *Computational Linguistics*, vol. 19, num. 4, 1993, p. 571–590.

[MIL 99] MILLER P. H., *Strong Generative Capacity*, CSLI publications, 1999.

[MIY 03] MIYAO Y., NINOMIYA T., TSUJII J., “Lexicalized grammar acquisition”, *Proc. of the 10th EACL companion volume*, 2003, p. 127–130.

[NED 98] NEDERHOF M.-J., “An alternative LR algorithm for TAGs”, *Proc. of COLING–ACL*, 1998, p. 946–952.

[NOO 94] VAN NOORD G., “Head corner parsing for TAG”, *Computational Intelligence*, vol. 10, num. 4, 1994, p. 525-534.

[POL 94a] POLLARD C., SAG I. A., *Head-Driven Phrase Structure Grammar*, University of Chicago Press and CSLI Publications, 1994.

[POL 94b] POLLER P., “Incremental parsing with LD/TLP-TAGs”, *Computational Intelligence*, vol. 10, num. 4, 1994, p. 549–562.

[POL 98] POLLER P., BECKER T., “Two-step TAG parsing revisited”, *Proc. of TAG+4*, 1998, p. 143–146.

[SAR 00a] SARKAR A., “Practical experiments in parsing using Tree Adjoining Grammars”, *Proc. of TAG+5*, 2000, p. 193–198.

[SAR 00b] SARKAR A., XIA F., JOSHI A., “Some experiments on indicators of parsing complexity for lexicalized grammars”, *Proc. of the 18th COLING workshop*, 2000, p. 37–42.

[SCH 88] SCHABES Y., ABEILLÉ A., JOSHI A. K., “Parsing strategies with ‘lexicalized’ grammars: application to Tree Adjoining Grammars.”, *Proc. of the 12th COLING*, 1988, p. 578–583.

[SCH 95] SCHABES Y., WATERS R. C., “Tree Insertion Grammar: A cubic-time parsable formalism that lexicalizes context-free grammar without changing the tree produced”, *Computational Linguistics*, vol. 21, num. 4, 1995, p. 479–513.

[SHA 02] SHAUMYAN O., CARROLL J., WEIR D., “Evaluation of LTAG parsing with supertag compaction”, *Proc. of TAG+6*, 2002, p. 201–205.

[TOM 86] TOMITA M., *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*, Kluwer Academic Publisher, 1986.

[TOR 95] TORISAWA K., TSUJII J., “Compiling HPSG-style grammar to object-oriented language”, *Proc. of the NLPRS*, 1995, p. 320–325.

[TOR 96] TORISAWA K., TSUJII J., “Computing phrasal-signs in HPSG prior to parsing”, *Proc. of the 16th COLING*, 1996, p. 949–955.

[TOR 00] TORISAWA K., NISHIDA K., MIYAO Y., TSUJII J., “An HPSG parser with CFG filtering”, *Natural Language Engineering*, vol. 6, num. 1, 2000, p. 63–80.

[VIJ 85] VIJAY-SHANKER K., JOSHI A. K., “Some computational properties of Tree Adjoining Grammars”, *Proc. of the 23rd ACL*, 1985, p. 82–93.

[VIJ 87] VIJAY-SHANKER K., “A study of Tree Adjoining Grammars”, PhD thesis, Department of Computer & Information Science, University of Pennsylvania, 1987.

[VIJ 88] VIJAY-SHANKER K., JOSHI A. K., “Feature structures based Tree Adjoining Grammars”, *Proc. of the 12th COLING*, 1988, p. 714–719.

[XIA 99] XIA F., “Extracting Tree Adjoining Grammars from bracketed corpora”, *Proc. of the fifth NLPRS*, 1999, p. 398–403.

[XTA 01] XTAG RESEARCH GROUP, “A lexicalized Tree Adjoining Grammar for English”, report num. IRCS-01-03, 2001, IRCS, University of Pennsylvania.

[YOS 99] YOSHIDA M., NINOMIYA T., TORISAWA K., MAKINO T., TSUJII J., “Efficient FB-LTAG Parser and its Parallelization”, *Proc. of PACLING*, 1999, p. 90–103.

[YOS 02] YOSHINAGA N., MIYAO Y., “Grammar conversion from LTAG to HPSG”, *The European Student Journal on Language and Speech*, , 2002, available at http://hltheses.elsnet.org/eventpapers/essliproceed.htm.

[YOU 67] YOUNGER D. H., “Recognition and parsing of context-free languages in time n^3”, *Information and Control*, vol. 2, num. 10, 1967, p. 189–208.

A. Formal proof of strong equivalence for the grammar conversion from LTAG to HPSG-style grammar

The proof comprises two parts. Part one proves that strong equivalence is guaranteed for the conversion from LTAG G to canonical LTAG G' by the tree division and the tree substitution. Part two proves that strong equivalence is guaranteed for the conversion from canonical LTAG G' to an HPSG-style grammar G''.

A.1. *Definitions*

Definition A.1 (Lexicalized Tree Adjoining Grammar (LTAG)) *A lexicalized tree adjoining grammar G is a quintuplet (Σ, NT, S, I, A) where Σ and NT are a finite set of terminal symbols and a finite set of nonterminal symbols, respectively, S is a distinguished nonterminal symbol called the start symbol, and I and A are a finite set of initial trees and a finite set of auxiliary trees, respectively.*[13]

Here, an elementary tree $\gamma \in A \cup I$ is a tree whose leaf nodes are labeled by $X \in NT \cup S$ or $x \in \Sigma$, and whose internal nodes are labeled by $X \in NT \cup S$. The symbol of one leaf node in an auxiliary tree $\beta \in A$ is identical to that of its root node, and is specially marked as a foot node. Note that more than one leaf node, called anchors, in an elementary tree γ are labeled with $x \in \Sigma$, and leaf nodes other than anchors and the foot node are marked as substitution nodes.

We hereafter use the notion of *an address* of a node in a tree. An address of a tree is a symbol that indicates a unique node in the tree.

Next, we define *a derivation* for an elementary tree γ. Let us denote a tree that is derived from an elementary tree γ by having substitution and adjunction into γ as γ'. When we produce γ' from an elementary tree γ by applying substitutions and adjunctions of several trees $\gamma'_1, \gamma'_2, \ldots, \gamma'_k$ to γ at k distinct addresses $a_1, a_2, \ldots, a_k$, the production is denoted by $\gamma' \rightarrow \gamma[a_1, \gamma'_1][a_2, \gamma'_2] \ldots [a_k, \gamma'_k]$ where $k \geq 1$, and $[a_i, \gamma'_i]$ indicates substitution at a_i of γ'_i if a_i is a substitution node, or indicates adjunction at a_i of γ'_i if a_i is an internal node. This production is called *a derivation* for γ if $a_1, a_2, \ldots, a_k$ include all addresses of the substitution nodes in γ. A derivation for γ without substitution and adjunction is denoted as $\gamma' \rightarrow \epsilon$. The set of all possible derivations D_G for LTAG G = (Σ, NT, S, I, A) is then denoted as follows:

$$\begin{aligned} D_G = \; & \{\gamma'_i \rightarrow \epsilon \mid 1 \leq i \leq m, \gamma_i \in A \cup I, \gamma_i \text{ includes no substitution node.}\} \\ & \cup \{\gamma'_i \rightarrow \gamma_i[a_1, \gamma'_{i_1}][a_2, \gamma'_{i_2}] \ldots [a_k, \gamma'_{i_k}] \mid k \geq j \geq 1, i > m, \gamma_i, \gamma_{i_j} \in A \cup I, \\ & \qquad a_1, a_2, \ldots, a_k \text{ include all addresses of the substitution nodes in } \gamma_i\} \end{aligned}$$

We use the above notations to define *a derivation tree*, which represents the history of combinations of trees and is a structural description of LTAG.

13. The LTAG definition follows the definition of TAG given by Vijay-Shanker [VIJ 87]. Due to limitations of space, we omit the notion of adjoining constraints and the proof including the notion in this article, and then assume all internal nodes take selective adjoining constraints.

Definition A.2 (derivation tree) *A derivation tree for LTAG G = (Σ, NT, S, I, A), Υ_G, is formed from any subset of the set of all derivations D_G by uniquely relabeling identical elementary trees in the derivations of the subset. A derivation tree Υ_G must satisfy the following conditions:*

– Because γ_i can be adjoined or substituted once, γ_i' can appear once respectively in the left-hand side and the right-hand side of derivations in Υ_G except for the one distinguished elementary tree γ_S, which is the root of the derivation tree Υ_G. The condition implies that trees cannot be substituted or adjoined to more than one node.

– γ_S' can appear once in the left-hand side of the derivation.

– The inequality $i > i_j \geq 1$ is necessary to avoid cyclic applications of substitution and adjunction among elementary trees.

Next, we give the definition of strong equivalence between two grammars G_1 and G_2. Strong equivalence is intuitively that the two grammars generate equivalent *structural descriptions*, which are the most informative data structures given by G; examples of structural descriptions are parse trees by CFG and derivation trees by LTAG. The following definition follows from the one by Miller [MIL 99, p. 7].

Definition A.3 (strong equivalence) *Let the set of all possible structural descriptions given by two given grammars G_1 and G_2 be $T_D(G_1)$ and $T_D(G_2)$. The two given grammars G_1 and G_2 are strongly equivalent if and only if there is a bijective (i.e., one-to-one and onto) mapping from a structural description of G_1, $\Upsilon_{G_1} \in T_D(G_1)$, to a structural description of G_2, $\Upsilon_{G_2} \in T_D(G_2)$.*

In what follows, we assume that structural descriptions of LTAG are derivation trees in which the root node of γ_S is labeled by the start symbol *S* in the definition A.2.

A.2. *Proof of strong equivalence for the tree division and the tree substitution*

The tree substitution of Section 3.2 is exactly the same as the one that Schabes and Waters [SCH 95, pp. 494–495] defined and proved in their procedure of strong lexicalization of CFG. We briefly give a proof sketch that strong equivalence is guaranteed for grammars before and after the tree division.

For the tree division of Section 3.2, we can readily construct a bijective mapping between derivation trees by LTAG G and canonical LTAG G' converted from G by the tree division. Assume that an elementary tree of G, γ, is converted by the tree division into a supertree γ^u and a subtree γ^v, both of which are elementary trees of G'. We can then map any derivation tree by G one-to-one onto a unique derivation tree by G' by replacing every occurrence of γ^u which takes a substitution of γ^v in derivations with γ and vice versa. Note that γ^v must accompany γ^u because γ^v can be substituted only into γ^u and cannot be the root of a derivation tree.

A.3. *Proof of strong equivalence for the conversion from canonical LTAG to an HPSG-style grammar*

We prove that strong equivalence is guaranteed for a conversion from canonical LTAG G to an HPSG-style grammar G'. We first define *an HPSG parse*, which is a structural description of an HPSG-style grammar. We then prove strong equivalence by giving a bijective mapping from a derivation tree by G to an HPSG parse by G'.

Definition A.4 (HPSG-style grammar converted from LTAG) *Given canonical LTAG G = (Σ, NT, S, I, A), an HPSG-style grammar G' converted from G is denoted by a sextuplet (Σ, NT, S, Δ_I, Δ_A, R) where $\delta_i \in \Delta_I$ and $\delta_j \in \Delta_A$ are lexical entries converted from $\gamma_i \in I$ and $\gamma_j \in A$, respectively, and R denotes the substitution and adjunction rules. δ_i is denoted as follows: δ_i = (s_0, (s_1, l_1, d_1, t_1), $\ldots$, (s_k, l_k, d_k, t_k)) where $k \geq 1$, $s_0 \in \Sigma \cup NT$ is the symbol of the mother node of the anchor in γ_i, and $s_j \in \Sigma \cup NT$, $l_j \in \Sigma \cup NT$, $d_j \in \{right, left\}$, $t_j \in \{+, -\}$ are values of Sym, Leaf, Dir, and Foot? features in the j-th element of the Arg feature in δ_i. When the length of the Arg feature of δ_i is 0, δ_i is denoted as δ_i = (s_0, ϕ).*

First, we introduce the notion of *origination* for the Sym and Leaf features in HPSG lexical entries in order to define *an HPSG parse*, which represents the histories of rule applications to lexical entries and is a structural description of an HPSG-style grammar. We hereafter assume that each HPSG lexical entry δ_i is converted from a canonical elementary tree γ_i. We define the origination of the feature in δ_i as $\langle p, \gamma_i \rangle$, which indicates that the value of the feature originates from the symbol of a node with address p in γ_i.

Next, we define *a rule history* for δ_i, which is a history of rule applications to a lexical entry δ_i. Since the grammar rule must pop the value of the Arg feature of one daughter, we assign each rule application to δ_i as an element of the sequence of rule applications for δ_i if and only if the applied rule pops an element that originates from an element of the Arg feature in δ_i. Assume that δ_i is denoted as the one given in definition A.4. When the origination of l_j and s_{i_j} unified with l_j in the grammar rule are $\langle a_j, \gamma_i \rangle$ and $\langle b, \gamma_{i_j} \rangle$, respectively, a sequence of rule applications for δ_i is denoted as follows:

$$\delta'_i \rightarrow \delta_i[x_{i_1}, y_{i_1}][x_{i_2}, y_{i_2}] \ldots [x_{i_k}, y_{i_k}],$$

where $k \geq j \geq 1$, (x_{i_j}, y_{i_j}) is (a_i, δ'_{i_j}) if t_j = $-$ or (b, δ_{i_j}) if t_h = $+$. When $x_{i_1}, x_{i_2}, \ldots x_{i_k}$ include b where $k \geq h \geq 1$ and t_h = $+$ or a_j where $k \geq j \geq 1$ and t_j = $-$ in the sequence of rule applications for δ_i, we call the sequence of rule applications *a rule history* for δ_i. When the length of the Arg feature of δ_i is 0, a rule history for δ_i is denoted by $\delta'_i \rightarrow \epsilon$.

Lemma A.1 *Given an HPSG-style grammar G' = (Σ, NT, S, Δ_I, Δ_A, R), a rule history for $\delta_i \in \Delta_I \cup \Delta_A$ must be the following form.*

i) When the length of the Arg feature of δ_i is 0, $\delta'_i \rightarrow \epsilon$

ii) When the length of the Arg *feature of δ_i is not 0 and $\delta_i \in \Delta_I$, $\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}][a_2, \delta'_{i_2}] \ldots [a_k, \delta'_{i_k}]$.*

iii) When the length of the Arg *feature of δ_i is not 0 and $\delta_i \in \Delta_A$, $\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}] \ldots [a_{h-1}, \delta'_{i_{h-1}}][b, \delta_{i_h}][a_{h+1}, \delta'_{i_{h+1}}] \ldots [a_k, \delta'_{i_k}]$ where $t_h = +$.*

Proof When the length of the Arg feature of δ_i is 0, no rule application is assigned as a rule application for δ_i because it is defined according to elements in the Arg feature. The rule history for δ_i is thus denoted as $\delta'_i \rightarrow \epsilon$.

When $\delta_i \in \Delta_I$, the elements in the Arg feature of δ_i keep their order until the grammar rules consume all the elements. This is because both substitution and adjunction rules do not change the order of the Arg feature, and also do not remove an element of the Arg feature without unifying it with another node. The rule history for δ_i is thus denoted as $\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}][a_2, \delta'_{i_2}] \ldots [a_k, \delta'_{i_k}]$.

When $\delta_i \in \Delta_A$, the elements in the Arg feature of δ_i keep their order until the grammar rules consume all the elements as in the case where $\delta_i \in \Delta_I$. One difference is that it includes exactly one element l_h when $t_h = +$. The rule history for δ is then denoted by $\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}] \ldots [a_{h-1}, \delta'_{i_{h-1}}][b, \delta_{i_h}][a_{h+1}, \delta'_{i_{h+1}}] \ldots [a_k, \delta'_{i_k}]$ where t_h = $+$.

By using lemma A.1, we can define the set of rule histories by G' = (Σ, NT, S, Δ_I, Δ_A, R) as follows:

$$\begin{aligned} D_{G'} = \; & \{\delta'_i \rightarrow \epsilon \mid 1 \leq i \leq m, \gamma_i \in I, \text{the length of the Arg feature of } \delta_i \text{ is 0 } \} \\ & \cup \{\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}] \ldots [a_k, \delta'_{i_k}] \mid m < i \leq n, k \geq j \geq 1, \delta_i, \delta_{i_j} \in \Delta_I\} \\ & \cup \{\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}] \ldots [a_{h-1}, \delta'_{i_{h-1}}][b, \delta_{i_h}][a_{h+1}, \delta'_{i_{h+1}}] \ldots [a_k, \delta'_{i_k}] \\ & \qquad \mid n < i, k \geq h \geq 1, k \geq j \geq 1, t_h = +, \delta_i \in \Delta_A, \delta_{i_j} \in \Delta_I\} \end{aligned}$$

We use the above notations to define *an HPSG parse*, [14] which represents the history of rule applications and is a structural description of an HPSG-style grammar.

Definition A.5 (HPSG parse) *Given an HPSG-style grammar G' = (Σ, NT, S, Δ_I, Δ_A, R) converted from G, an HPSG parse $\Psi_{G'}$ is formed from any subset of the set of all rule histories $D_{G'}$ by renaming identical lexical entries in the rule histories of the subset uniquely. An HPSG parse $\Psi_{G'}$ must satisfy the following conditions:*

– δ'_i where $\delta_i \in \Delta_I$ can appear once respectively in the left-hand side and the right-hand side of rule histories except for the one distinguished lexical entry δ_S where δ'_S appears once in the left-hand side of the rule history for δ_S.

– δ'_i where $\delta_i \in \Delta_A$ must appear only once in the left-hand side of the rule history for δ_i.

– $1 \leq i_j < i$ for the rule history for $\delta_i \in \Delta_I$.

14. Due to limitations of space, we omit the proof showing that an HPSG parse by G corresponds to a unique parse tree derived by G.

– $1 \leq i_j < i$ *where* $j \neq h$, *and* $i_h > i$, *for the rule history for* $\delta_i \in \Delta_A$.

The third and fourth conditions are necessary to avoid cyclic applications of grammar rules to lexical entries.

Lemma A.2 *Let* G *= (*Σ*,* NT*,* S*,* I*,* A*) be canonical LTAG and* G' *= (*Σ*,* NT*,* S*,* Δ_I*,* Δ_A*,* R*) be an HPSG-style grammar converted from* G*. Then, we can map a derivation tree* Υ_G *by* G *one-to-one onto to an HPSG parse* $\Psi_{G'}$ *by* G'*.*

Proof We first show a mapping from $\Psi_{G'}$ to a set of derivations $\Upsilon_{G'}$, and then show that $\Upsilon_{G'}$ is a valid derivation by G. Suppose an HPSG parse satisfying definition A.5. We can map it one-to-one onto a set of derivations $\Upsilon_{G'}$ with the following procedure. For each δ_i where $\delta_i \in \Delta_A$, we eliminate $[b, \delta_{i_h}]$, which corresponds to an application of the adjunction rule, and add the element $[b, \delta'_i]$ to the right-hand side of the rule history for δ_{i_h}. Then, we obtain a set of derivations $\Upsilon_{G'}$ by replacing δ_{i_j} and δ'_{i_j} with γ_{i_j} and γ'_{i_j} in the rule history for δ_i and by regarding it as the derivation for γ_i in $\Upsilon_{G'}$. This mapping is one-to-one because the operation pair of eliminating $[b, \delta_{i_h}]$ and adding $[b, \delta'_i]$ is a one-to-one mapping.

Following the definition A.2, we show that $\Upsilon_{G'}$ is a valid derivation tree by G. First, every substitution and adjunction in the derivations in $\Upsilon_{G'}$ must be valid in G. Since the substitution and adjunction rules preserve the order of the elements in the Arg feature of δ_i, substitution rules always unify the symbol of the substitution node with the symbol of the root node of γ_{i_j}. This unification represents the same constraint as the one imposed by substitution. We can give an analogous argument for an adjunction rule. The substitution and adjunction in the derivations in $\Upsilon_{G'}$ are then valid in G. Second, all addresses in the substitution nodes of γ_i must be included in the derivation for γ_i. This is apparently guaranteed by definition of the rule history for δ_i. Third, γ'_i can appear only once respectively in the right-hand side and the left-hand side of the derivations. This is apparently guaranteed for γ'_i where $\gamma_i \in I$ by definition A.5, and is guaranteed for γ'_i where $\gamma_i \in A$ because δ'_i does not appear in the right-hand side of rule histories, $[b, \delta_{i_h}]$ appears only once in the rule history for δ_i, and the elimination of $[b, \delta_{i_h}]$ accompanies the addition of $[b, \gamma'_i]$ once to the right-hand side of the derivation for γ_{i_h}. Fourth, the elements in the right-hand side of the derivation for γ_i must be $[a_j, \gamma'_{i_j}]$ where $i_j < i$. This is apparently guaranteed for γ'_i where $\gamma_i \in I$ by definition A.5, and is guaranteed for γ'_i where $\gamma_i \in A$ because the addition of $[b, \gamma'_i]$ for the derivation for γ'_{i_h} satisfies $i_h > i$ from definition A.5.

The frontier string is preserved before and after this mapping from $\Psi_{G'}$ to $\Upsilon_{G'}$, because δ_i stores the same linear precedence constraints between δ_i and δ_j for $i \neq j$ as the constraints between γ_i and γ_j. Thus, an HPSG parse $\Psi_{G'}$ by G' is mapped one-to-one onto a derivation tree $\Upsilon_{G'}$ that is valid in G.

We can construct a mapping from Υ_G onto an HPSG parse Ψ_G by inverting the procedure for the above mapping from $\Psi_{G'}$ onto $\Upsilon_{G'}$. The obtained Ψ_G is a valid HPSG parse by G' because we can give an analogous argument for the validity of the rule histories in Ψ_G.