Packing of Feature Structures for Optimizing the HPSG-style Grammar translated from TAG

Yusuke Miyao†, Kentaro Torisawa†, Yuka Tateisi†, Jun'ichi Tsujii†‡
†Department of Information Science, Graduate School of Science, University of Tokyo*
Hongo 7-3-1, Bunkyo-ku, Tokyo 113-0033, Japan
‡CCL, UMIST, Manchester, U. K.
E-mail: {yusuke, torisawa, yucca, tsujii}@is.s.u-tokyo.ac.jp

1 Introduction

This paper describes a method for packing feature structures, which is used for reducing the number of constituents generated during parsing, and for improving the parsing speed. The method was developed for optimizing a parsing system for *XHPSG* (Tateisi et al., 1998) translated from XTAG (The XTAG Research Group, 1995). The XHPSG system is a wide-coverage parsing system for English based on HPSG framework (Pollard and Sag, 1994). This system is also intended to be used for processing large amounts of texts, for the purposes such as information extraction. Current parsing speed of our system is not sufficient enough to achieve this goal.

Our method improves the parsing speed by solving the problem which the XHPSG and the XTAG system have. That is, many lexical entries are assigned to a word, and many constituents are produced during parsing. The experimental results show that our method leads to a significant speed-up. The results also suggest the possibility of optimizing the XTAG system by introducing packing of feature structures and packing of tree structures, although these operations are not currently so apparent.

2 The XHPSG System

This section describes the current status of the XHPSG system and the efficiency problem in the system. Both of the grammar and the parser in the XHPSG system are implemented with feature structure description language, *LiLFeS* (Makino et al., 1998). The grammar consists of lexical entries for about 317,000 words, and 10 schemata, which follows schemata of the HPSG framework in (Pollard and Sag, 1994) with slight modifications. The parser is a simple CKY-based parser.

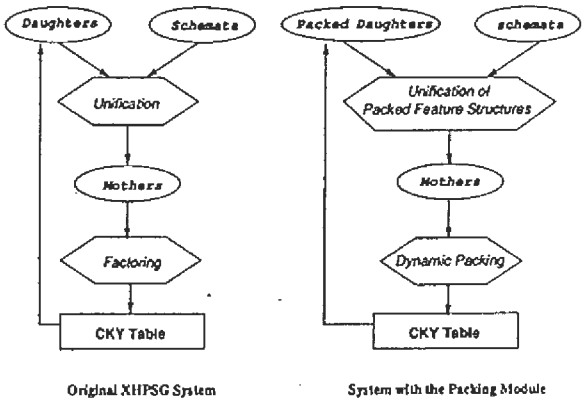

Figure 1: Data flow in the parsers for the XHPSG system.

Currently, the parsing speed of this system is not satisfactory, and we need further improvement of the parsing speed. One of the major reasons of inefficiency is that the XHPSG system assigns many lexical entries to a single word. For example, a noun is assigned 11 lexical entries, a verb is assigned 20–30 lexical entries, and some words are even assigned more than 100 entries.

This characteristic is inherited from the XTAG grammar. The XTAG grammar assigns many elementary trees to a single word, and there is a one-to-one correspondence between a lexical entry in XHPSG and an elementary tree in the XTAG grammar. The XTAG system applies a POS tagger before parsing in order to overcome this inefficiency by reducing the number of lexical entries assigned to a word. However, this method sacrifices the soundness of the

This work is partially founded by Japan Society for the Promotion of Science (JSPS-RFTF96P00502).

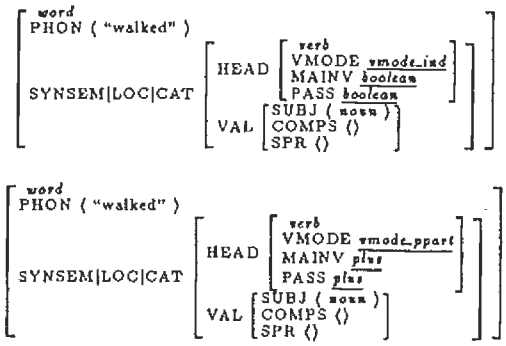

Figure 2: Two of the lexical entries for an English verb "walked". Underlined values are different. Most of the features are omitted for simplicity.

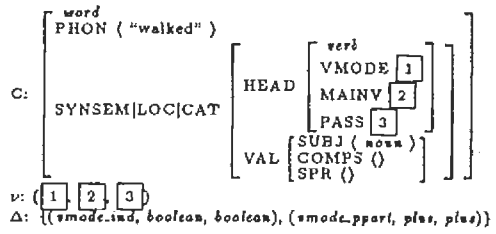

Figure 3: A *packed feature structure* for the lexical entries shown in Figure 2.

parsing process. In the case that the tagger fails to assign the correct POS to a word, correct syntactic structures may not be created even when the grammar potentially covers such structures.

To solve the same problem, we propose a new method described in the next section. The method can gain a similar effect, but does not sacrifice the soundness of parsing.

3 Packing of Feature Structures

The left hand side of Figure 1 illustrates the data flow of the original parser of the XHPSG system. There are two major operations, *unification* and *factoring*. When we apply a schema to daughters, a unification operation is performed, and a mother is created. A set of mothers are reduced to a smaller set of feature structures by factoring operation[1], and these constituents are put into CKY table.

```
create_PFS(F)
  C := ⊥, ν := (), δ := ()
  for each f ∈ features(F)
    if f ∈ DisjFeatures then
      ν := (follow(C, f))@ν
      δ := (follow(F, f))@δ
    else
      F' :=follow(F, f)
      (C', ν', {δ'}) =create_PFS(F')
      C := C ⊔ [f C']
      ν := ν@ν', δ := δ@δ'
    end_if
  end_for
  return (C, ν, {δ})
```

Figure 4: Algorithm for creating new packed feature structure from a feature structure. '@' denotes the concatenation operation of sequences.

The right hand side of Figure 1 illustrates the parser with the packing module. The unification and the factoring operation in the original parser was replaced by *unification of packed feature structures* and *dynamic packing*. These operations are more efficient than the corresponding one, because multiple applications of schemata are reduced to one unification of packed feature structures, and multiple operations of factoring are reduced to one dynamic packing. In addition, *dynamic packing* reduces the constituents further than the factoring operation.

With a simple example, now we see how feature structures are packed into one. Figure 2 shows two of the lexical entries that the XHPSG system assigns to an English verb "*walked*". These lexical entries correspond to distinct elementary trees of XTAG. They are different in only a few features, while each feature structure has over 100 features. That is, most of them have equivalent values, so that it is redundant to have each of them as two independent feature structures.

For these feature structures, a packed feature structure is described as in Figure 3. C specifies the common part of the original two feature

[1] A factoring operation in a CKY parser for CFG reduces the number of constituents by identifying constituents described by the identical non-terminals. The operation plays a crucial role for avoiding an exponential explosion of the time complexity of the parsing of CFG. In the case of HPSG, the similar effect can be accomplished by the factoring operation, which identifies the constituents with equivalent feature structures in this case. We have observed that parsing time with syntactic grammars can be reduced significantly, though this operation does not lead to a reduction of computational time complexity to polynomial.

```
pack_feature_structures(PFS)
  PFS' := φ
  for each P = (C, ν, Δ) ∈ PFS
    if P' = (C', ν', Δ') ∈ PFS' such that
      C' is equivalent to C and,
      for each i(0 ≤ i ≤ k)
      paths(C, n_i) = paths(C', n'_i)
      where ν = (n_0, ···, n_k) and ν' = (n'_0, ···, n'_k)
    then
      Δ'' := Δ ∪ Δ'
      PFS' := (PFS'\{P'}) ∪ {(C, ν, Δ'')}
    else
      PFS' := PFS' ∪ {P}
    end_if
  end_for
  return PFS'
```

Figure 5: Algorithm for packing a set of packed feature structures.

```
unify_packed_feature_structures(P_1, P_2)
  P_1 = (C_1, ν_1, Δ_1)
  P_2 = (C_2, ν_2, Δ_2)
  Δ := φ
  if success C := C_1 ⊔ C_2 then
    ν := ν_1@ν_2
    for each δ_1 ∈ Δ_1 and δ_2 ∈ Δ_2
      δ := copy((ν_1 ⊔ δ_1)@(ν_2 ⊔ δ_2))..U_1
      Δ := Δ ∪ {δ}
      Cancel the side-effect of ⊔
      occuring during computation of U_1.
    end_for
  end_if
  return (C, ν, Δ)
```

Figure 6: Algorithm for unifying two packed feature structures.

structures. ν expresses the nodes[2] in the feature structure, to which disjunctive structures are incorporated. The nodes are expressed as tags for structure sharings such as $\boxed{1}$. Δ expresses a set of different values, that come to the position specified by the nodes in ν. Hence, the original feature structures are obtained by unifying one of the elements of Δ to the nodes in ν. A packed feature structure holds exactly the equivalent information of the original feature structures with a smaller data size.

4 Algorithms

This section describes three algorithms, (1)conversion of a feature structure to a packed feature structure, (2)packing of packed feature structures and (3)unification of packed feature structures.

The last two algorithm requires packed feature structures as their inputs and the first algorithm is used for convert non-disjunctive feature structures to such inputs to the two algorithms. Figure 4 shows the first algorithm for converting a feature structure to a new packed feature structure. We assume that a packed feature structure is given as a triple (C, ν, Δ) as described in Section 3. The input to this algorithm is a (non-disjunctive) feature structure and a set of features, to which the disjunction is introduced. In the figure, F is a feature structure and $DisjFeatures$ is a set of features. The function follow(F, f) returns the node in F reached by the feature f from a root of F. What the algorithm does is to split F into two parts, the first part is C and the other part is a set of nodes and a set of substructures represented by ν and Δ respectively.

Figure 5 shows the algorithm for packing already packed feature structures. In the figure, $\mathcal{PFS}$ denotes a given set of packed feature structures, and $\mathcal{PFS}'$ denotes a newly created set of packed feature structures. The function $paths(F, n)$ returns a set of all the paths to the node n in F. The algorithm for packing lexical entries is straightforwardly obtained from this algorithm and the previous algorithm.

Figure 6 shows the algorithm for unifying two packed feature structures. The overall algorithm is similar to the one in (Kasper, 1987), although data structures for disjunctive feature structures are different. Intuitively, we first unify common parts (C_1 and C_2), and next check consistency of each combination of disjuncts in Δ_1 and Δ_2. The operator $\sqcup$ denotes the unification of non-disjunctive feature structures[3]. The unification is regarded as an destructive procedure in the figure. It has a side effect to the input feature structures. For instance, suppose that feature structures stored in the variables F and F' have the nodes stored in the variable n and n' as their substructure and that for some path π $follow(F, \pi) = n$, $follow(F', \pi) = n'$ and $n \neq n'$. After performing the unification $F \sqcup F'$, the values of F, F', n and n' are automatically updated and, as a result of the update, $F = F'$ and $n = n'$ hold. In the algorithm in the figure, this type of side-

[2] Though feature structures are expressed in a conventional matrix-like notation, they can be seen as directed graph with a root whose nodes and arcs are labeled. Features are labels for arcs and the labels for nodes are called types.

[3] Unification of tuples is a tuple of the results of the unification of corresponding elements of the tuples.

Features incorporated from XTAG
PRD, CASE, PRON, REFL, VMODE, MAINV, EXTRACT, TRANS, PASS, PERF, PROG, ASSIGN_CASE, INV
Other features
HEADPHON, MARKING, CONT, TRF

Table 1: Specified features for the experiments.

	Parsing time in avg. (sec.)	
	Test set A	Test set B
Old System(O)	2.31	14.45
New System(N)	1.29	5.88
Improvement Ratio(O/N)	1.79	2.46

The experiments are performed on Alpha Station 500 (500MHz CPU, 256MB Memory), and the times are measured in User Time.

Table 2: Results of the experiments.

effects is assumed to occur for the values stored in the variables such as $C_1, C_2, \nu_1, \nu_2, \delta_1$ and δ_2. The mechanisms for the side-effect and its canceling are similar to the execution mechansims of Prolog, including backtracking. They are also implemented in LiLFeS. The *copy* is a procedure to create a *distinct* feature structure equivalent to the input feature structure and the newly created feature structure is free from the side-effect of the unification against the original input feature structure.

5 Experiments

This section shows the experimental results of the current implementation of our packing method. Experiments are performed by specifying features originated in XTAG and a few other features as in Table 1.

The packing module is implemented with LiLFeS, and is incorporated into the XHPSG system. We compared the parsing times of (1)*Test set A* (337 sentences, 8.37 words/sentence)[4] and (2)*Test set B* (16 sentences, 11.88 words/sentence)[5], between the (1)*New System* (with the packing module) and the (2)*Old System* (without the packing module). The parsers of both systems are simple CKY-based parsers. As Table 2 shows, the parsing speed improves by 1.79 times in Test set A, and 2.46 times in Test set B, which consists of sentences costing much time to parse. In Test set A, the number of lexical entries is reduced by 35.3%, and that of constituents in the CKY table by 46.7% on average.

[4]Test set A is bundled in the XTAG system for checking the grammar.

[5]Test set B is a subset of Test set A. The subset consists of 16 sentences, each of which costs more than 10 seconds to parse.

6 Conclusion and Future Work

We proposed a method for packing feature structures by introducing disjunctions into feature structures. This method reduces the number of lexical entries in HPSG grammars and constituents created during parsing. As a result, we achieved 1.74 times improvement in parsing time for the test corpus bundled in the XTAG system. We expect to gain the similar effect with the XTAG system by applying our packing method, though it is currently not so apparent.

For realizing a practical parsing system, we are currently integrating our packing method with other two optimization techniques: (1)implementation with a native compiler version of LiLFeS (Makino et al., 1998), and (2)compilation of HPSG to CFG (Torisawa and Tsujii, 1996). As a result of the latter optimization, current XHPSG system can parse sentences in the ATIS corpus in 1.12 seconds on average without any POS taggers. Further speed-up is expected by integrating our method to this system.

References

Robert T. Kasper. 1987. A unification method for disjunctive feature descriptions. In *Proc. 26th ACL*, pages 235–242.

Takaki Makino, Minoru Yoshida, Kentaro Torisawa, and Jun'ichi Tsujii. 1998. LiLFeS — towards a practical HPSG parser. In *Proc. of COLING-ACL '98*. To appear.

C. Pollard and I. A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Yuka Tateisi, Kentaro Torisawa, Yusuke Miyao, and Jun'ichi Tsujii. 1998. Translating the XTAG English grammar to HPSG. In *Proc. TAG+ Workshop*.

The XTAG Research Group. 1995. A lexicalized tree adjoining grammar for English. Technical report, IRCS Report 95-03, University of Pennsylvania, March.

Kentaro Torisawa and Jun'ichi Tsujii. 1996. Computing phrasal-signs in HPSG prior to parsing. In *Proc. 16th COLING*, pages 949–955.